Surfing in Kansas Documentation

Release 1.0

Eric Holscher

November 10, 2013

Contents

Welcome to the home of Eric Holscher on the web.

I talk about software development, mostly in the realm of Django. I am interested in the web, testing, mobile apps, documentation, and much more.

I currently live in Portland, Oregon, and love to explore the outdoors. Whether on bike or foot, I spend as much time as possible out and about, checking out the various beauty that the world holds.

CHAPTER 1

Information

1.1 About me

I currently call Portland, Oregon home. I lived in the woods for over 2 months while walking 800 miles of the Pacific Crest Trail in 2013.

I have previously lived in Lawrence, Kansas for 2.5 years from May 2010-Dec 2012. I lived the rest of my life in Virginia, migrating from the Shennendoah Valley, to Virginia Beach for high school, then to Fredericksburg for college.

I've previously worked at these awesome companies:

- Urban Airship
- Lawrence Journal-World

1.2 Projects

Some projects that I have worked on over the years.

- Read The Docs

A Django Dash project, allowing users to host their Sphinx documentation easily.

- Write the Docs Conference

A conference about all things documentation, held in Portland, Oregon.

- Write the Docs Documentation

A resource for writing better documentation, and spreading the general culture of writing documentation in software.

- Django Kong

A wrapper around twill that allows you to do monitoring and basic functional testing of your sites.

- Django Test Utils

This is a project of mine to help with the testing of Django applications.

1.3 Résumé

Eric Holscher
757 705 0532
Portland, Oregon

1.3.1 Interests

- Django & Python
- Documentation
- Devops
- Scaling systems

1.3.2 Open Source Projects

- Read the Docs - http://readthedocs.org
  - Over 5000 projects
  - Over 5 million page views a month
  - Standard hosting for Python community
- Member of the Python Software Foundation

1.3.3 Conferences

- Helped organize the Write the Docs conference - http://conf.writethedocs.org
  - 200+ people came to talk about documentation
  - Portland, Oregon on April 8-9 2013
  - Has spawned local users groups in multiple cities

1.3.4 Speaking

Spoken at the following conferences on Read the Docs, IRC bots, Testing, and Documentation:

- Open Source Bridge 2012, 2011
- Pyweb Summit 2012
- Djangocon US 2011, 2010
- OSCON 2011
- Djangocon EU 2010, 2009

More information available on Lanyrd: http://lanyrd.com/profile/ericholscher/past/speaking/

1.3.5 Work Experience

January 2013 - July 2013

Pacific Crest Trail

- Trained for 3 months by hiking 50 miles a week
- Hiked 800 miles of the trail before getting injured
- Lived in the woods for 2 months
- Gave my mind and soul room to breathe

December 2010 - January 2013

Urban Airship - Developer & Operations

- Employee 15 at a startup that now has over 150 people
- Assisted moving data centers on the ops team
- Helped scale infrastructure to handle hundreds of millions of messages a day

July 2008 - December 2010

Lawrence Journal-World - Developer

- The birthplace of Django, maintaining the world's oldest Django codebase
- Worked as Lead developer and defacto sysadmin for the "Internal" team

  - Ljworld.com, Lawrence.com, Kusports.com and ~20 other sites
- Worked on the "Commerical" team on Ellington
  - Ported Ellington from Django r1290 to 1.0
  - Wrote lots of tests and supported the product

March 2006 - May 2008

(Hiatus July 2006-January 2007 while in Australia)

CACI - Developer

- Full Life-cycle development of a Portal-type website for the U.S. Navy
- Wrote a javascript validation library, allowing you to add rules in one line, and saving developers from writing custom javascript for each page
- Used Prototype and Scriptaculous Javascript libraries for AJAX and other development

1.3.6 Education

University of Mary Washington

Bachelor of Science in Computer Science

- Graduated May 2008
- Major GPA of 3.5

CHAPTER 2

Activities

Photo: My bike on the Williamette Valley Scenic Bikeway

2.1 Bike Touring

After my PCT injury, I wondered what I could use all of my lightweight camping gear for. I turned to bike touring, allowing a machine to handle the stress that my foot obviously isn't capable of.

2.1.1 August 24, 2013: First overnight

Went on my first bike camping trip to Dodge Park

2.1.2 September 7-10, 2013: First Bike Tour, Portland to Eugene

This was a delightful ride in the Oregon fall, along the Williamette Valley Scenic Bikeway. I took basically 3 days, starting with a MAX ride to 158th street in Beaverton. I then went down to a friend's going away party.

Main Ride

Total: 185 miles
Day 1: 62 miles - Beaverton to Keizer Rapids City Park
Day 2: 68 miles - Keizer Rapids City Park to Peoria Boat Dock
Day 3: 55 miles - Peoria Boat Dock to Eugene

Thoughts

I quite enjoyed bike touring as an endevour. It feels a lot like thru hiking, where you have an overarching goal all the time. Always moving, always more miles. I really enjoyed seeing the country by bike, though the addition of car traffic wasn't a welcome one.

Compared to the wilderness experience of backpacking, I think I enjoy backpacking more. I was along a relatively populated area of Oregon though, so I imagine if I went out east in Oregon I would have a more comperable experience.

The actual scenic bikeway also had a distinct lack of camping. The camp sites I chose were of dubious legality, though nobody questioned me when I camped. As far as I can tell, there is a 80+ mile stretch on the scenic bikeway with no official camping options. This really limited my comfort level, because I was having to worry about where I would camp each night.

Photo: Mount Whitney (14505 ft) at Sunrise

2.2 Backpacking

Backpacking is a hobby that I have picked up since moving to the Pacific Northwest.

2.2.1 Pacific Crest Trail

I walked on the Pacific Crest Trail from April 15 - June 16 in 2013. I had to come home due to a stress fracture in my third metatarsal in my left foot, after walking 800 miles.

I have collected my favorite photos on Flickr

2.2.2 Mount St. Helens

I summitted Mount St. Helens in March of 2013, as a training hike for the PCT.

2.2.3 Mount Hood

I started taking trips around Mount Hood to get my experience in the field. I have done a few different trips, mainly in 2012. A popular one was from Top Spur Trailhead to Cairn Basin.

2.3 Blog Entries

2.3.1 2013

A Walk in the Woods

I've recently schemed some very large changes in my life, and I have been trying to figure out how to blog about it. It all starts with quitting my job at Urban Airship. I sent an email to the staff announcing my departure, and I can't think of a better way to remember this moment in time than to include it in full in my blog. So, here is the letter I wrote to everyone at work when I left the company.

All,

It's with great anticipation and mixed feelings that I announce the end of my ride upon the Airship.

For I am leaving to face a greater challenge and a personal life goal. I will be attempting to walk 2650 miles from Mexico to Canada along the Pacific Crest Trail starting in the spring. Doing a thru-hike of the AT or the PCT has long been something I have wanted to do, and I am in a place in my life where the responsibilities I have are minimal, so it seems like a great time to go walk in the woods for 5 months. The approximate timeline is from April 15~September 30.

Figure 2.1: Pacific Crest Trail

I will be starting training for the hike immediately, and that is hard with a full time job. Luckily, I have been offered a contract gig to work on my side project Read the Docs for as many hours as I want until I leave. This is another life dream come true, the ability to get paid to work on an Open Source project that I have created.

With this confluence of fortune, I leave the Airship for what should be one of the most amazing years of my life.

I would like to thank everyone who I have had the opportunity to work with here at UA, because it has been the best job of my life. Watching the company grow from 15 to over 100 people has been a formative experience in my professional life. I can only hope that I will get another chance to watch such an amazing group of people build such a great company again.

My last day will be Friday, Jan 18th. So, feel free to come accost me while I'm still here and ask me questions, or call me crazy, or give me hugs. I'll also be kicking around in Portland until April, so I should still be around for the PDX Python meetups and other bits and pieces.

My personal email address is eric@ericholscher.com if you want to keep in touch. I will be blogging about my experience on my website at http://ericholscher.com, if you want to follow along. There isn't anything there yet because this wasn't public knowledge, but I will start posting about it soon.

Figure 2.2: Cop High Five

"Live in each season as it passes; breathe the air, drink the drink, taste the fruit, and resign yourself to the influence of the earth." Henry David Thoreau, Walden

Cheers, Eric

tl;dr:

- I'm leaving Urban Airship.
- I'm going to be hiking 2650 miles from Mexico to Canada on the Pacfic Crest Trail starting on April 15, hopefully ending near the end of September.
- I've gotten a contracting gig to work on Read the Docs (my open source side project) for as many hours as I wish until then.
- How much I love you: lots

tl;dr gif:

Figure 2.3: Oregon Trail

Announcing Write the Docs

Documentation is one of the most important parts of a software project. However, a lot of projects have little or no documentation to help their (potential) users use the software. A few years ago we started Read the Docs to help make hosting documentation easier. Part of the reason was that if hosting documentation was a solved problem, it would make people more likely to write docs.

However, there is still a large hole in the documentation world around writing documentation. There are some resources about it, but they are scattered around the internet in random places. Write the Docs is trying to solve this problem by getting all of the people that care about documentation in a room, to improve the art and science of documentation.

Write the Docs is two things. The first and more immediately interesting is that it is a two day conference in Portland, Oregon. Held on April 8-9, **it will bring the community that exists around documentation together.** Through this event we will spread a lot of knowledge about how, why, and when to write documentation.

The second part of the project is a resource for people who are writing documentation. It will solve the problems of someone who has the question: **I want to write docs, but what do I write?!**. It will be a home of best practices around documentation, and a lot more. We hope it will serve as a growing resource of all things documentation. We want to build the canonical source for people who have questions about documentation, and to further the art of documentation in all forms.

Write the Docs will also be a community that you can become involed in. We are announcing a mailing list that will serve as a place to ask questions, and bring together all those who write the docs. In addition, the Write the Docs Sphinx repository is open source, and accepting contributions to the information there.

All of the above is a work in progress. I hope you join me in supporting and contributing these projects. I believe that they will advance the state of the software community in many ways. The vision for the project lays out some of the things that we want to accomplish.

Remember: Docs or it didn't happen :)

Prepping for the Pacific Crest Trail

My time in Portland in winding down, and the trail is approaching in my minds eye. I leave on the 14th of April, and 6 in the morning. Setting out on a jet-plane to San Diego. I will spend the night on the 14th and be transported to the trailhead at sunrise the following day.

Prep

I have been attempting to walk 10 miles a day, and gain 1000 feet of elevation. I have been using a Fitbit to keep objective data about my day, and I've been doing pretty well in this department. My longest hike was a 20 mile hike that went from Terwilliger Blvd in Southwest Portland, along the west hills, into Forest Park. That day exhausted me, but it was good to see my body could do 20 miles in a day, seeing that's what I need to average to complete my journey.

Gear

I have my gear pretty much dialed in. My gear list is available on Postholer, and is mostly up to date. I'm pretty happy with all of my gear selections, having done a couple of test hikes overnight.

Online Presence

I have been gathering my online assets together, and consolidating. I have moved my blog from a Django app into a Sphinx repo hosted on Read the Docs. That will make it much easier to update, and I don't have to worry about database backups or anything. It's all based out of a git repo, and hosted on Read the Docs, which is highly available.

This means I only have 1 personal server left, and that is running my IRC bouncer. I will likely keep this around just for fun, but I will likely knock down the buffer size on my bouncer so that it doesn't overwhelm my phone when I log on in the woods. I'll want to be able to chat with people and keep logs of the channels I'm in, but I don't need all the scrollback after a week of walking.

Mental Prep

The big part of the journey currently is just getting my head into a good state. The concept of hiking the PCT is becoming very real, and it's scary and exciting as hell. I know it will be an amazing experience, but it's quite the physical feat. It's also something different; something I have never done anything like before. It's that nervous feeling that you get when you are pushing your limits.

I am trying to step back from the emotions I'm experiencing to appreciate the experience of it. Much like watching a movie, I need to appreciate the fear and apprehension that I'm experiencing as an artifact of doing something important and necessary. There is common wisdom that goes along the lines of "The things that scare you the most are the most important to do," and I'm trying to keep that mindset in the front of my mind.

Help me improve documentation

tl;dr: I don't have (or want) a job, help fund me to make the documentation world better.

A year ago, I *asked* people to help cover the hosting costs of Read the Docs. It was successful and the site hosting has been funded ever since. Simply being hosted isn't enough. There are many other aspects to having a healthy community, including development, support, and advocacy. The approach this time around is aiming to support those aspects as well.

In the last year, I have gone on to do a lot more to help improve the documentation world. I want to continue to do this work, with your help.

Work I am doing

- **Developing and maintaining Read the Docs**

  - A website that hosts documentation for ~4000 projects, and has over 5 million pageviews/month.
- **Co-organizing and producing Write the Docs conference**
  - A community, non-profit conference that gathers 250 people in Portland to talk docs.
- **Helping organize regional Write the Docs meetups**
  - We currently have meetups in SF, NYC, and Boston, hopefully more coming soon!
- **Writing documentation about documentation**
  - A growing resource of information about writing docs.
- **Writing presentations about documentation**
  - The end goal being a set of presentations that people can use to promote documentation at meetups and conferences.
- **An unreleased project called Grok the Docs**
  - The idea here is to improve documentation through data.

Why it matters

I have talked to many people around the tech world over the last year, and they all agree that the state of documentation is lacking. I want to help improve this situation in many ways.

Read the Docs is the defacto documentation hosting tool for Python. It has also spread into many other communities, hosting documentation for PHP, Javascript, Science, Games, Databases, and more. It removes barriers to people writing documentation, allowing them to simply get started down a well paved path.

Through my work on Write the Docs, we are building a community around documentation. This community is helping to push the state of the art. Joining together allows us to present a unified face for improving documentation in the open source, and proprietary software worlds.

Grok the Docs is a project that is still in development. The goal is to use analytics data to improve the browsing experience. It is very much a research project, but I really hope Grok the Docs will improve the experience of reading and writing documentation.

Budget

I quit my job earlier this year to hike the Pacific Crest Trail. I have returned home (after only 800 miles, but that's a different story), and want to try living life a bit differently.

I am asking for support on Gittip to help me on this endeavor. Currently, my living arrangements cost around **$1200/month**:

- $500 for Rent & Utilities
- $300 for Food
- $250 for Health Insurance
- $80 for Cell Phone

This works out to around **$300/week**, which is the way that Gittip keeps track of money. I hope to achieve independence from money so that I can continue to do work on this important problem.

Note: This is a baseline existence. Simply enough to make sure I don't die or starve. Going to conferences, buying hardware, taxes, and other expenses will need to be figured out too.

Why Gittip?

Gittip supports me without telling me who is giving money. This allows me to maintain and develop projects without regard to whose interests might be impacted by my work. This gives me the freedom to work on important things, without regard for who might stop giving me money.

I also don't want to promise certain outcomes with the money. Platforms like Kickstarter and Indiegogo are great for raising money, but they require that you offer something in return. I am approaching this more as patronage. **If you like the work I have done, sponsor me to continue doing more good things.** My interests will change over time, and I want to have freedom to change direction.

Support me

I don't want to have to get a job. I want to continue doing work that I love, and that I think matters.

If you think that the work that I'm doing is important, I hope that you support me on Gittip.

This is an experiment, and I have no idea what is going to happen. So please feel free to give me feedback over email or Twitter.

I started today with $1.75 in gittip, the current tally is:

A letter to an old friend

A letter I wrote to an old friend Josh, after I had come home from the PCT. It was sent on August 7th, 2013. It captures my thoughts on the trail pretty well.

Text

Howdy,

Indeed the trail satisfied many parts of the soul. Sadly I hurt my foot and had to end the trail before I reached the end, but I still spent enough time out there to understand some of the lessons it has to teach.

I always like to think back to how life was for me before college. It's crazy how much of a different person I was going in and coming out. We chiseled away at the possibilities of humanity, and ended up with a pretty good statue at the end.

Being on the trail is the closest that I have had to that feeling in my adult life. You are given the time to think without constraint, and bullshit all day long with amazing people. It feels much like college, in that the friends I've made will be life long.

It has changed what I want to do with myself for the foreseeable future, seeking out more situations where the intensity and breadth of the human soul can come to bear. It felt like the most natural thing in the world to slip into walking everyday, and living a simple life.

Simple life with good people, a profound and fundamental way to enjoy the world. It showed me how full of bullshit and tedium the "normal" world can be, and how a reduced subset of choice can really expand your happiness.

The months I spent on the trail were some of the happiest of my life. I will look to hopefully relive them again in a few years, but like many things in life, the first time is likely to be the sweetest. Hopefully I won't go through life looking backwards, trying to feel it again, and will find solace and peace again.

Coming back to reality has been hard, but Portland is a good place to come back to. I have been surrounding my self with good food, mostly from Farmers Markets. Lots of berries and salads, things you can't eat on the trail. I am also looking into doing some bike touring, once my foot heals, so that I can still explore. Exploring on bike will be a different experience, but one I think I'll quite enjoy.

I now also have the experience to go and live in the woods for a week at a time. This freedom opens a lot of possibility for adventure in the future. Having amazing tools, the knowledge , wherewithal, and drive to go out into nature again makes me happy.

May there be many more adventures for us both.

Cheers,
Eric

Writing a Beginners Guide to Documentation

A few days ago I *started a campaign* to improve documentation. Today I have the first results to show from this work.

It started first with a presentation that I presented at PDX Python here in Portland. The talk was very well received, so I decided to write it up.

So, I present A beginners guide to writing documentation. It is still very much a work in progress, so I hope that you can provide feedback. The idea behind the presentation and document is to allow people modify and present it themselves. I am hoping to build documents and presentations for other aspects of documentation as well. If you want to give this presentation, I would love for you to email me.

As I said, this work was done as part of my ongoing work on Documentation. If you think this work is important, you should support me on Gittip.

Sphinx Live Preview

For a long time, there has been a live preview site for reStructuredText: http://rst.ninjs.org/. It is really fantastic for learning the language. The immediate feedback is really valuable in helping you expirment and see how things work.

I think that this tool would be great for people to have with Sphinx as well. I know a lot of people use Sphinx, and then end up on that page, and random things don't work because they are extensions to reStructuredText. So, I went ahead and forked the app to support Sphinx.

I present: http://livesphinx.herokuapp.com/.

It is still very much a work in progress. The output should be themed nicely, and support syntax highlighting. If you have any feedback or requests, please tweet or email me.

This project was done as part of my *ongoing work* to improve documentation. If you think this work is important, you should support me on Gittip.

Google Summer of Code Book Sprint 2013

Or how I learned to stop worrying and write a book.

Last week, I flew down to Mountain View, Ca for the Google Summer of Code Book Sprint. It is an event that brings open source projects together to write a book in a week. 20 people and 3 projects produced books over a sunny and well-fed week. Kindly hosted by Google, we spent 5 days hanging out at their campus writing away. The three projects involved were:

- OpenMRS - A medical records system used in many developing countries, originally created to help with AIDS in Africa.
- BRL-CAD - A CAD program for developing 3D models, one of the first open source projects from the United States Military.
- Mallard - An XML based documentation framework, from the Gnome Documentation Team.

I was what they called a "free agent", someone not involved in a specific project that would help out. Free agents are useful for providing an outside viewpoint to the projects.

Writing a book was an amazing experience. It has always felt out of my reach, but time limiting it to a week (really 3 days), made the goal more approachable. I hope this post will help you conceptualize the process of how we wrote the book, and possibly think about doing it yourself. After this experience I think it is something that anyone can do, as long as you keep the scope small.

You can see the finished book for OpenMRS online.

Monday

Heresy On Monday, we got together to play some games to break the ice. We were asked to write our most polarizing documentation viewpoints on an index card, and they were read out loud in front of the room. People arranged themselves along a spectrum of agreement and disagreement. Physically arranged themselves, by walking to different corners of the room. Then we presented reasons for our views on the topics mentioned.

Some examples are:

- Developers MUST write documentation
- WYSIWYG editors are evil

This exercise acted to show how diverse a crowd we had. Some people were developers, some were tech writers, some were students, some were teachers, people ranged all across the spectrum of experience. You learned to respect where people were coming, which might be very different from your background.

Audience and Outcomes After the ice breakers, we did an exercise where each member of the team was broken into a separate group, and had to write down their views for the book. We were told to have **up to 3 audiences in mind, and 3 take aways someone would have from reading the book**. This allowed people to flesh out their idea of the book without other members of their team present. The idea behind this exercise was to eliminate the group-think that happens when a group works to shape an idea. It also allowed us free agents to get an idea of what book people were writing, and which we might be able to help out with.

Once we all fleshed out our ideas individually, the groups got back together and talked through what they thought the book should be. This is the stage where free agents chose their team as well, so that they could see the vision for the book.

As a project group, we then chose the audiences and takeaways for the book. This is the part where I joined the OpenMRS group. They were writing an introductory book for developers who wanted to get involved in the project, and I felt I could help out as a developer coming to the project fresh.

We assumed everyone would be new to OpenMRS. We then broke this down into 3 types of things people might be new to:

- Developers new to Health IT
- Developers new to Open Source
- Developers new to Software Development in General

The outcomes we wanted people to come away with were:

- Understand how to become a member of the OpenMRS community
- Be able to get the OpenMRS environment set up
- Feel confident doing development on OpenMRS modules

Tuesday

Table of Contents On Tuesday we got together in the morning to come up with a table of contents based on our audience and outcomes. Having the audience and outcomes written down really helped guide and focus the book. At each step someone could ask "is this really serving our intended audience?" We only had 2 and a half days to actually write, so we needed to aggressively trim the content to something that we could accomplish in that time.

Our table of contents ended up looking like:

- **Introduction**
  - Welcome to OpenMRS!
- **Saving Lives With Software**
  - The Need for Health IT
  - Our Response
  - OpenMRS Today
- **Community**
  - Working Cooperatively
  - Collaboration Tools
- **Technology**
  - Architecture
  - Data Model
  - Development Process
  - Get Set Up
- **Case Study**
  - Creating your First Module
- **What's Next?**
  - Get Involved
  - Get Support
  - Developer Checklist

Promotion Plan We also talked through a plan to release the book into the community. There was an understanding that if you don't promote the book, the time spent writing it might go to waste. Having a way to build momentum for the project in the community would ensure the book continued to live on after this week.

Our original promotion plan looked like:

- Blog post announcing the book on the project blog
- Tell developers in the project about it, so they can recommend it to people
- Add it to all of our beginner documentation

- Talk with existing developers to make sure the information in the book is correct
- Add a JIRA project so we can track issues with the book
- Add a survey so that we can get feedback on the book
- Make sure that updating the book is added to release processes

Compare and Contrast After coming up with ideas inside our own teams, we sent a member to each other team to hear what they had come up with. We were encouraged to steal their ideas if they had something interesting, and to provide feedback if we saw something missing. This worked really well at removing group think again, and making sure that you didn't have a huge blind spot in your plans.

Start writing After lunch on Tuesday, it was time to start writing. This part was referred to as "content production", there was a specific focus on just getting pen to paper. Editing would come later. We worked until 8 in the evening, and then headed back to the hotel.

Around the pool that evening we spent time hanging out and talking about ideas. In particular I talked to the Mallard team, comparing and contrasting it to Sphinx.

Wednesday

Content production continued Wednesday. The goal was to have a complete book by Wednesday night, and then spend Thursday refining and editing it down.

Thursday

Thursday was spent writing until around lunch, then the afternoon was spent editing. We formed groups of 2 or 3 which looked over a section at a time. Each section had an average of 3 chapters, and you looked to make sure the flow of all the chapters made sense together. We would each read a chapter and then talk over each of the issues that we found.

At 6pm on Thursday we called the books done, and all celebrated.

Friday

On Friday we got together to do a postmortem on the process. We talked again about the promotion plan, assigning items to specific people to make sure they got done.

This was all along the theme of continuing momentum forward. We now had a list of tasks, with people who were responsible for getting them done. This made me feel a lot more confident that our work would live on, and really make a difference in the community.

Take aways

I think the mixing of ideas behind groups was really key to success in this endeavor. Group think is potent, and having someone with an outside perspective come in can really reveal your blind spots.

Along these lines, the evenings hanging out by the pool talking through your work was really important. You can't sit and write 24/7, and having a place to escape and let you ideas breathe really allows you to form them. I think throughout the week everyone was thinking about their book pretty non-stop, but were weren't necessarily writing non-stop.

I come away from this experience with a lot of inspiration and perspective. Writing a book is something that anyone can do, with a little help from their friends.

Announcing Grok the Docs

> Are my docs working?
> Are folks getting what they need?

I've asked myself these questions a lot. Historically I have put Google Analytics on my doc pages, and called it good. I would browse over the data every once in a while, gleaning basically zero information out of it.

Read the Docs hosts a lot of documentation, and I want to help these folks understand how their docs are being used. So I have been working on a project for the last month called Grok the Docs.

Grok the Docs is a bit different, with the main difference being it embeds the information in the page for you. This is interesting because it adds context to the data. I believe that context is the first step from turning data into information. The main interface to Grok the Docs are keyboard shortcuts within the documentation page. So you can access information about the current page you're on, while you're browsing. Check out the *Example* below to see it in action.

Surfacing analytic data in the page is great for the maintainer and user alike. The maintainer can see what parts of their docs are being heavily used, and which parts aren't being used as much. Users can see where other people are ending up, which is probably where they want to go too.

This is very much just a tech demo currently. I would love feedback from folks about how I could improve the display of the data. Currently it's something that you need to enable.

It would be great if you have ideas for other additoinal functionality that could be added. This is very much an experiment currently, so I'd love to hear any thoughts you have. Please email or tweet me if you have feedback or ideas.

Once the code is more baked and solid, the plan is to turn it on for all Read the Docs users. After I do a full rollout across Read the Docs, I'll consider opening it to other people. The code is currently closed source, and will likely remain so. The idea is that this might become a product that can suppliment my Gittip income. If it fails as a product, I will then open source it. That said, it will always be free for documentation on Read the Docs.

This project was done as part of my *ongoing work* to improve documentation. If you think this work is important, you should support me on Gittip.

Example

This shows how user might harness this data.

A new theme for Read the Docs

We have been hard at work improving Read the Docs over the past month. A large amount of back end work has been going on, and now we have a brand new documentation theme to showcase it.

We have looked at how people use documentation, and built a beautiful and highly functional new interface for browsing documentation. We created a solution that looks great and works well.

Creation

Dave Snider approached me about a month ago, interested in helping improve the documentation ecosystem. We talked about what could improve the Read the Docs experience for the greatest number of our users, and a new theme seemed like a great place to start.

The New Theme

Full site The full documentation page is now beautiful and streamlined. We got rid of the visual clutter and integrated a full-project Table of Contents in the sidebar. When you go into a specific page, a page-level contents get embedded in the sidebar as well. This allows you to keep context inside the documentation when you land on a page from a search.

Old

New

Flyout Read the Docs provides a lot of functionality for documentation projects. The flyout is the avenue to accessing that functionality. We need to pack all our functionality into this space.

In the new theme, the flyout is integrated into the bottom left of the theme.

Old The old flyout let you:

- Change versions
- Go back to Read the Docs

New The new flyout lets you:

- Change versions
- Go back to Read the Docs
- See the current version
- Show if the current version is out of date
- Download docs for offline viewing
- Contribute edits on GitHub or Bitbucket
- Do a full-text search (Coming soon)

Mobile The new theme really shines on mobile. We provide a beautiful interface for phones and tablets, while staying completely functional.

Using it

There are two ways that you can use this theme on Read the Docs. The first is to simply leave your `html_theme` variable set to `default`. This is now the default Read the Docs theme. You can also set `RTD_NEW_THEME = True` in your project's `conf.py`, and we will use our theme when building on Read the Docs no matter what `html_theme` is set to.

After you change these settings, simply rebuild your docs and the theme should update. More information about the theme can be found on the theme documentation page

If you want to continue using the old theme, simply set `RTD_OLD_THEME = True` in your `conf.py`.

Conclusion

This theme is a great addition to the documentation ecosystem. It is highly functional and beautiful, allowing users to easily navigate and find what they need.

We have a few more tricks up our sleeves, but theme is ready to launch today. Over the next few weeks we'll be adding a bit more functionality to it, which should be even more delightful.

I hope that you enjoy using it. If you have any feedback, please open an issue on GitHub for the repo. To follow announcements for Read the Docs, follow us on Twitter

If you want to support work like this, help fund development on Read the Docs on Gittip.

2.3.2 2012

Why Read the Docs matters

Documenting projects is hard, hosting them shouldn't be. Read the Docs was created to make hosting documentation simple. I think that we have solved this problem well, but now we need to start thinking about the larger picture.

Along with hosting, Read the Docs was created with 2 other main goals. One was to encourage people to write documentation, by removing the barrier of entry of hosting. The other was to create a central platform for people to find documentation. Having a shared platform for all documentation allows for innovation at the platform level, allowing work to be done once and benefit everyone. Having run the site for over a year now, I think there is a third thing that we should be striving for. That is to make the quality of documentation better.

I think that **we can help a documentation culture flourish** within the open source world. Django is a shining example of what a project with great documentation can do, and it has a community that values docs more than the norm. I think we can help **spread this culture throughout the Python world, and beyond**. This has already started, and I want to think about how something like RTD can help.

What we can do to help

I think that having a **guide for writing useful documentation** would be a great step towards helping people along the path of documentation enlightenment. Jacob Kaplan-Moss has started down this road with his blog series and Pycon 2011 talk on this subject. I think that we could start by collecting these into a section of the site.

We could build on top of that great start with simple guides for how to get started with Sphinx, best practices for documentation, and providing a general place to learn more about how to write good documentation. Since we host a lot of documentation, we could point to live examples of techniques, and provide helpers for people to enable the techniques.

I have started a reStructedText Philosophy document that is meant to help people understand the ideas behind how reST works, so that it isn't as mystifying. This reST cheatsteet also appears to have similar goals. These are a very basic start, and I think some more along these lines would really help a lot of people get over the barrier to starting and continuing to write good documentation.

I think that we could also help **create contributors** to projects, if we could find an easy way to provide patches to documentation. If you could go to the project documentation, and fix small typos, or help add a paragraph in the tutorial, it would lower the bar to helping.

However, it isn't a wiki. These changes would be represented to the project author as pull requests in their VCS, and they would still be responsible for tending the garden. This gets rid of the "Just Edit The Wiki" solution of documentation, and also helps new contributors get started in an easier fashion.

The Plone community has built a proof of concept, linking to Github's edit pages for the current document. I think we can integrate this at the platform level, and make it available to everyone.

Want to help?

Read the Docs is open source. You can help by writing docs for the site, writing code for the site, or just writing documentation in general. People can also help just by using the site, and reporting bugs. Telling us how to make the site better helps everyone in the long run. Come join us on Freenode in the #readthedocs channel as well.

Another area that we're hurting is in the design front. We have been adding features over time, and the design of the site is getting a bit strained. Having someone with a good sense of design help re-think and re-architect some of the features and ideas that we've been working on I think would help a lot.

A lot of the RTD contributors will be at Pycon 2012, where we will be having a sprint on the site. If you want to get started contributing, that is a great place to come and get started.

The festival that felt like a hug

A story about XOXO Festival in 3 acts. I will start first with something that set the tone, then talk about the importance to me, and then what I hope comes from it in the future.

Act 1: Interjecting awesome

One moment stood out to me in the swirl of ideas and amazing that was XOXO. It was at the beginning of the conference, when the organizers were on stage. They were talking about how they wanted the conference to be experienced. The sentence that I think changed the entire conference experience for me (paraphrased):

> This should be a conference where you can go up to a group of people you don't know, and they will include you in their conversation.

This seems like something very simple, but it set an important social contract. Normally my introverted self will balk at the idea of joining a group of unknown people. Especially at a conference with so many people who I look up to and admire. However, this idea, set forth by the organizers, dispelled this apprehension, and instead I viewed it as my responsibility to interject.

This was a fundamental change in how I experienced the conference. I spend most of my time at conferences talking to people I've known for years, rarely breaking into new groups. At XOXO, though, since I knew only a few people beforehand and felt compelled to meet new folks, I spent the entire conference striking up conversations with complete strangers. This was a profoundly different and amazing experience.

Act 2: Bring out your trolls

Consuming the world through twitter is not a way to be inspired. Getting together in a room and seeing people who have changed their world, and the world for others, is an amazing experience. It allows you to perceive and appreciate people's aspirations.

I started XOXO in a funk that can only be explained as cynical. I had heard of Kickstarter and the ilk, but never really invested or taken the time to fully let the idea wash over me. As the talks started, and I heard Kickstarter over and over, it at first felt like a promotion and a buzz word. However, through the genuine excitement and joy of bringing something new into the world, my skepticism turned into inspiration.

Greed being destructive was a theme behind the conference, and I think this is the primary thing that won me over. People were creating things because they wanted them to exist in the world, and they had to do it. It wasn't about making money, or getting famous, but because they had a drive to change a part of life. This drove the jealousy and skepticism from my heart, and started the search for the thing in life that I was meant to change.

Act 3: Radiating change

I think that this conference was an amazing view into a world that could exist. At a high level it was a distancing from the classical tech world that is so focused on money. A place where we can be open, share our ideas, successes, and failures. Somewhere that people can actually introduce something into the world and have support for it.

During the talk on Kickstarter, Yancey mentioned that Portland has been the most successful city on Kickstarter. Something like $7.5M has been given to creators in the rose city. On the technical side, we have a burgeoning, but not well formed start up community. This means that we can form this community into something that is different than has existed in other places.

As Paul Graham once said, each city sends you a message. I think that this conference was in some ways a call to action, that a place like XOXO needs to exist in a more permanent manner. I think that Portland has a chance of doing this going forward. I can't, and won't, try to spell out how this could be done. I will say that I can't imagine another city that is better poised to do it.

I want Portland to be the place where you come, and think you can change the world.

Help fund Read the Docs

Currently Read the Docs is funded mainly through Corporate sponsorship. The Django and Python Software Foundations (non-profits), Mozilla, Lab305, Revsys, and others have helped keep the site running. However, this requires finding sponsors to help donate to the site every 6 months or so to keep things running.

I want to try out a new idea that is effectively a subscription to the website. When people pay for something, they expect certain things. A promise of support, uptime, and other work are basically being transfered in the mind of the person providing payment. I know some places try to explicitly denounce this transaction, but it is still there.

This is where Gittip comes in. It has the idea of funding a person to do work through anonymous donations. The thinking behind this is that the person recieving the money now has no sense of obligation to the person giving money. This allows them to take the money and continue to work on Open Source without feeling pressured to work on the things a specific person giving them money cares about.

I think this same idea can apply to software projects. Read the Docs doesn't cost a huge amount of money to run every month - it costs a lot less than keeping a person alive and happy. So, I think that the first success story for Gittip funding something could easily be a project instead of a person. This funding model would then allow Read the Docs to support itself over time - without having to try and get support and investment again.

Read the Docs currently costs about $300/mo to run. This includes 6 servers (2 web, LB, Build, Database, Util/Monitoring), over 350GB of data transfered, over 100GB of repositories, and it serves over 3 million page views every month. We expect these costs to slowly rise as we get more and more traffic, but that is the goal we are currently aiming to hit. Head to the Read the Docs gittip page if you want to help out.

This is the beauty of Gittip - when 75 different people are giving you $4 a month ($1 a week), one can stop giving and it doesn't totally destroy the funding. It allows other people to pick up the slack, and to sustain a dependable revenue stream for the project.

This is an experiment that I am going to try running to see if we can get individual sponsorship for the project, instead of depending on corporate sponsors for the sole source of support. Once this is achieved, we will look at other ways to spend the sponsorship we get from corporations, perhaps in more traditional efforts to advance the code base.

If this sounds interesting to you, head over to the Read the Docs gittip page, and start donating to the project.

Update: Wow! We reached the goal of $75 in around 14 hours. Thanks everyone who has donated to help keep the site running! It looks like we might reach above our weekly goal. For now, that money will just be left in an account to help pay for future growth of the site. If we end up making way more than we need, we'll find something awesome to do with it (CDN?!).

Interesting projects on Read the Docs: Teaching

As the maintainer of Read the Docs, I spend a lot of time looking through random projects, and getting inspired. People have been doing lots of interesting things with the project, and I'd like to highlight some of them.

This edition is focused on teaching. All of these projects are trying to teach something, and doing it in different ways. Some are community contributed guides that have many authors, where some are a single person trying to distill their experience into something valuable for others.

The projects mentioned here will be featured on the homepage of Read the Docs until I do another posting, where those new projects will take their place.

Little books of R

The Little Books of R were some of the first books that I was aware of on Read the Docs. They are great little manuals on things that you can do with the R programming language, often used for modeling and graphics.

There are a few different books, including how to use R with:

- Biomedical Statistics
- Time Series Analysis
- Multivariate Analysis
- Bioinformatics

These books are perfect examples of what publishing online provides. Short and sweet, to a specific niche, and easily available.

Ops School

Ops School is an attempt at providing a cirriculum for someone interested in learning Systems Administration. **It's answering the question of "What do I need to know to get a junior sysadmin job" that many people have before starting into a career.**

It takes a lot of knowledge that takes years to gather from experience and attempts to distill it down into a form that is easily referencable. As this project matures, it will provide a valuable resource for a lot of information around the running of systems.

The project is still in development, and is actively seeking contributors.

The Hitchhiker's Guide to Python

The Hitchhiker's Guide is a great project from the Python community. It's goal is to provide knowledge on best practices on the daily usage of the Python language. **It is trying to answer the question "What do I need to know that I don't know I need to know."** Known more colloquially as *unknown unknown's*, this knowledge is the hardest to gain. Having a guide from the community about the things that you should probably know about is invaluable as a new, or even experienced member of that community.

This project is also in active development.

Thoughts on RESTful API Design

These thoughts are a great collection of experience from someone who has built production REST APIs. If you are tasked with creating an API for a web site, it's a great read. **It provides a good framework for understanding how the API fits in with the rest of the application, as well as what makes a good API.**

Conclusion

I love the idea of Read the Docs as a medium of teaching, as well as just documenting software projects. **Please let me know of other interesting projects that you've found, and I can include them.** Raising awareness of these great resources is valuable, and hopefully it will reach more people who can learn from them.

2012 Year in Review

Wow, what a year. 2012 was a great year in my book. I took 2012 off from a lot of the professional development activies that have taken up my adult life thus far, and really focused on personal development. I think I did a great job with that, and I have a pretty awesome list of things I accomplished this year. I think I also started to get the full enjoyment out of Oregon this year, in all its seasons.

Physical achievements

The biggest achievement of the year has to be the Petal Pedal I did in June. That's a 100 mile bike ride through the Willamette Valley in Oregon. I trained for it through most of the spring with rides that started out around 20 miles and ended up in around 70-80 miles. Previously the longest bike ride I had done was probably around 10 miles. The official ride also had 5000 ft of eleveation gain, which added another element of difficultly. The ride was beautiful though, and went through a bunch of flower fields and Silver Falls State Park.

Earlier in 2012, I also started taking up skiing again. I went up to Ski Bowl for night skiing a bunch. I went ahead and bought a season pass for Ski Bowl for the 2013 season.

As summer came to Portland, I was outside pretty much all the time. It was a most excellent summer weather-wise and I took full advantage of it. I did a bunch of backpacking up on Mount Hood, with a number of weekend trips to various places including Carin Basin and Paradise Park. I also went car camping a bunch, including at Rock Creek in the Santiam State Forest, and Trillium Lake on Hood. I also stayed in a Yurt on the Oregon Coast for the first time at Cape Lookout.

The summer also included a couple fun road trips. I got down to see Crater Lake, the only national park in Oregon. We camped there for a night and drove around the rim. It is one of the most amazing places I have ever been. It's a crystal clear blue lake, but it's in the rim of an extinct volcanoe, at around 5000 ft elevation. It's a really amazing geological place and highly recommended. After that, we drove up to Bend, and camped at Lava Lake, which is also a beautiful place. We saw tons of ground squirrels and other wildlife there. We also went through Bend and walked around, and checked out Smith Rock. The trip concluded with a sunset walk up to Mirror Lake on hood, and a race against the sun to get back to the car.

I made it back home for 2 of my good college friends getting married (to each other) in DC and visited family in Virginia. I got some surfing in down in Virginia Beach, which was great. It's something I miss about Oregon, because the water is so cold here. Made it out to the Oregon Coast, but it's mainly just good for looking instead of swimming in.

The summer also included some awesome hikes and other activities. I went out to Hood River a couple times, and actually Wind Surfed and Paddle Boarded for the first time. We also drove around the east side of Hood to the Fruit Loop and gorged on cherries and other delicious fruit. Speaking of fruit, I ate a ton of berries again this year, with Oregon having some of the finest summer farmers markets in all the land.

Hiking was a big draw this year. I went on a number of hikes including the PCT from Timberline Lodge all the way to Cairn Basin on a couple different days. Eagle Creek, Angel's Rest, Dog Mountain, and a few more in the Columbia River Gorge. I hiked the Wildwood trail through Forest Park a number of times, however I didn't thru-hike it's 30 miles, will have to save that for next year.

As fall came around, things started to slow down. I still did a bunch of hiking through the fall, and riding my bike. I started rock climbing around this time to keep up the activity level, and have been progressing nicely at indoor climbing at the Portland Rock Gym. I can't wait to test my skills outside, but I already noticed my balance and confidence improving on the hikes I did in the fall.

As winter came into view, I escaped down to the Turks & Caicos islands for Christmas. Where I did 7 dives, got my Advanced Open Water cert, and did a bunch more snorkeling. It was great hanging out with my family in the warmth, and spending time in the water. When I got back, I went on my first snowshoeing adventure ever on Hood. I really like snowshoeing, it's basically just like hiking in the snow, just it takes a lot more work!

This year was also full of Ping Pong. We have a table at work, and it's rekindled my love of the game. I played some when I was a kid, but we have gotten pretty good and serious at work, and it's been a pleasure upping my skills again.

In list form

A list of firsts for the year:

- Rock Climbing
- Backpacking
- Wind Surfing
- Paddle Boarding
- Snow Shoeing
- Long Distance Biking

Things that I love that I did more of this year:

- Ping Pong
- Hiking
- Camping
- Skiing
- Diving
- Snorkeling
- Biking

Things I love that I didn't do so much of:

- Surfing
- Traveling
- Soccer
- Frisbee

New things I want to try next year:

- Bike Camping
- Longer backpacking trips

- Climbing outside
- Climbing Mount Hood and/or St Helens
- Thru hiking Forest Park
- White Water Rafting
- Kayaking

Conclusion

I'm sure I missed some stuff, but it was a year full of firsts, and I think I have gotten the most out of Oregon that I could. I chose to focus on my personal life this year instead of my professional life, and I think it's worked out for the best. I have had an amazing year full of excellent activies outdoors.

I also saw a bunch of music this year, and met a bunch of awesome new people, and did a bunch of great things professionally, but I think the points worth remembering will be the parts above.

As a wise man once said: **Work to Live, Don't Live to work**.

2.3.3 2011

Handling Django Settings Files

I have seen a lot of talk over the past couple years about how to handle different settings files and databases, synced between production and development. I have happened onto a way of doing it that makes me happy, and figured I would share it with the world.

File structure

I use a file structure that looks like this:

```
project/
    settings/
        __init__.py (empty)
        base.py
        sqlite.py
        postgres.py
```

The base.py contains all of the configuration options that are shared among the databases. INSTALLED_APPS, etc. All of the DATABASE settings should be specified in the more-specific files. As well as things that differ by environment, like remote servers, cache settings, cookie domains, and other things.

This allows you to run the sqlite settings file, and have it be set to localhost, or whatever your development settings are. Then in production you just run against the postgres settings.

A good example of this being used in practice is on Read the Docs.

But wait, there's more!

manage.py for dev

./manage.py is great for development. It is the easiest way to get started, and it automatically sets up your paths and stuff. With my setup, I actually explicitly set manage.py's settings file to the sqlite file.

This means that whenever you are using manage.py, you are in a development context. So, what do you do about production?

django-admin.py is for production

In production, you set your DJANGO_SETTINGS_MODULE to the postgres settings file. So whenever you use django-admin.py, you will be running against the production database.

I really like this scheme, because it gives you a logical distinction between production and development in your code, and in your interface on the CLI. When you are developing, you are using manage.py and editing the sqlite settings file. The reverse for production.

Read the Docs Updates

Documentation writing will always be hard work. It's a much different mind-set than programming, and people that write good code might not necessarily write good docs. However, this is a known issue, and something that can't really be solved.

What you can do is make it easier to write documentation. Every step along the way that you can give yourself an excuse to not write documentation is another undocumented open source project.

Luke Plant has a great post up about how important documentation is, and I completely agree. I imagine a lot of the people using Django are using it because of the documentation. I think as members of the Django community, we need to build a culture of documentation within the greater Python world. Python does tend to have better documentation than a lot of languages, but it's still not nearly what it could be.

Read the Docs exists to make it easier to host your Sphinx documentation. Over the weekend, Bobby, Jonas, and I added a bunch of new features to the site. I think it's getting to the point where there isn't an easier or better way to host the documentation for your Django project, and we're only going to keep improving it!

A different Eric added a really nice Getting Started guide for RTD, that shows how easy it is to get your projects hosted with us.

Anyway, on to the new features that we added.

New Features

Versions

Versions of projects are easily one of the biggest requested features on the site. For a long time we just supported building the latest versions of your documentation. Now we support versions of your documentation that are tagged in your VCS (hg/git only).

A lot of larger projects need versioning because they support one or two versions, as well as developing in the trunk. Django was the main project we were thinking of, but some other projects have put this to good use. A couple of examples are:

- Django's latest stable build
- Fabric 0.9.3,
- django-admin-tools's awesome integration that has all it's versions hosted with us.

PDF Support

Sphinx has interesting support for PDF generation through Latex. In my testing it was pretty unreliable, but I was able to rangle it into working well enough to expose in the UI. So now almost every project will have a "Download PDF" button. This code has version support as well, so we can offer PDFs of certain versions.

- Django's trunk documentation PDF
- Django CMS 2.1.0-rc2 PDF
- Varnish trunk PDF

Another interesting part of this feature is that this building code has been abstracted out, so we can support epub, plain text, and all the other Sphinx output formats that people want.

Badges on the project pages

We killed the RTD header on hosted documentation pages in favor of a Badge in the lower right hand corner. The header clashed with a lot of the themes, and the badge is nice because it gives us a place to put functionality that is always visible, but is obviously not part of the hosted documentation. We want to build some more functionality into the badge, like switching between versions and linking back to the project's RTD page, once we build a good UI for it.

Sponsorship

Revsys has agreed to sponsor the hosting costs for RTD. Jacob Kaplan-Moss has always been a big proponent of documentation, and I'm glad that he and Frank Wiles are helping us keep Read the Docs around and get better. We tried to make the sponsorship subtle and not intrusive, so please let me know if it bothers you and we can try and figure something out.

Conclusion

I think that these features are really starting to make RTD a compelling platform for hosting your documentation. We are planning more awesome features that will make RTD even better. I'm really excited about the project and I hope that you either host your docs with us, or find docs that we host useful.

Using Reviewboard with Git

Reviewboard is a great tool for managing the process of Code Reviews. It has pretty good git support, but it might not be obvious what the best way is to use it. At work, I have a couple of different ways of pushing up code for reviews, which I'll talk about.

This guide is assuming you are using your own bare repositories, on the server hosting the Reviewboard instance. It's mainly here so that I can remember how to do this next time I need to. Also, thanks to Travis Cline for the initial pointers for this post.

Setting up Reviewboard

Once you have Reviewboard installed, you need to add a Repository in the admin, which is located at `/admin/db/scmtools/repository/`. The required fields have the following values:

- Name: The name of the project
- Hosting service: Custom

- Repository type: Git
- Path: The path to the local checkout of the git repository (ex. /var/lib/git/name)
- Mirror Path: The **Url** to the repository (ex. ssh://git@your.server.com/var/lib/git/name)

The Repository Documentation has more about why you need this screwy configuration.

post Review

Before we get started, you're going to want to get the post-review tool that works along with Reviewboard. The easiest way to get it is to `pip install RBTools`.

Pointing to the right Reviewboard Instance

The easiest way to make sure your pointing at the right Reviewboard instance is the *.reviewboardrc* file in your home directory. You only need to add one line to that file, which is:

```
REVIEWBOARD_URL = "https://path.to.your.instance"
```

If you are working with multiple instances that map to different repos, you can set the Reviewboard instance for the specific repo as well:

```
git config reviewboard.url https://path.to.your.instance
```

Reviewing a Branch

Alright, now you are all setup to start posting reviews. The easiest way to do that is to branch off of master, and start committing. If you are following something similar to this awesome branching model, this should be your normal workflow. Once your branch is ready to be merged back into your project, you want to get it reviewed. You can send a review equivalent to "this branch diffed against master" like so:

```
post-review --guess-summary --guess-description
```

Reviewing one commit

Another thing I find myself doing a lot is working on my master, and only having one commit to review. In theory this should probably be done on a bugfix branch, but such is life. There are other good use cases for only reviewing the latest commit as well. It's done like so:

```
post-review --guess-summary --guess-description --parent=HEAD^
```

Reviewing arbitrary number of commits

It's also possible to review any number of previous commits. It looks a lot like the previous command:

```
post-review -o --guess-summary --guess-description --parent=HEAD~4 #To review last 4 commits.
```

If you are familiar with git, you will understand that there is a lot more that you can do with the –parent argument. I'll leave the possibilities up to your imagination.

Other useful post-review flags

The are a couple of other useful post-review flags, that I use from time to time.

- -d This basically outputs all of the git commands that post-review is using to generate the diffs. It's a great way to figure out what's going wrong when it can't find a diff.
- -o: This opens a web browser to the posted review once it's done. Great for easily making the review public.

I hope this makes it a little easier for you to set up a git repository with reviewboard.

Read the Docs Update

It's been a while since I last talked about Read the Docs, and there has been a lot of activity. This is an update on the latest and greatest new features.

PSF Funding

The biggest news that has happened is that we have been given a grant from the Python Software Foundation to help host the site. Thanks PSF! They have blogged about it, and I am grateful that they have given us support. With the funds they have offered, we have been able to make Read the Docs a lot faster, and more robust. I will outline some of the changes below.

This also means we won't be going away any time soon!

New Theme

We have a fancy new theme for documentation on Read the Docs! If you have the 'default' theme for your project, it will show up on the build of your docs on the site. I think it is pretty great, thanks to the designers who spent their time making it awesome. A really great feature is that the **new theme is mobile ready**. Go ahead and view a project using it on your phone, or make your browser smaller and you will see the fanciness. Having a custom theme will give us a base to build lots of other neat features on top of.

Better architecture

We had some connectivity trouble in between our servers a while back, and this prompted me to make the site respond better to these conditions. Every time you view documentation on a subdomain of readthedocs.org, your request will never hit Django. So all of these requests will work without a database. We have also added a second application server with a load balance in front, which means that one of the app servers could go away and your documentation would still get served.

That leaves our load balancer as the main single point of failure at the moment. We're using Varnish for the load balancer, and we've implemented strong caching of data. Varnish will cache your docs for up to a week, and it will be actively purged when you rebuild your docs. This means that your docs will usually be served out of memory, and without dependence on any other server but that one. We have plans to elininate Varnish as a single POF, and then it would only be our hosting provider that would be a single point of failure (famous last words).

Intersphinx support

Intersphinx is an awesome feature of Sphinx that allows you to reference remote sphinx documentation easily. RTD now supports it for every project that we host.

Improved rtfd.org

I've always had big ideas for rtfd.org, since it can act as a short-url for things. *projectname*.rtfd.org has always redirects to the projects docs, but now we have something a lot better. Inspired by Jacob Kaplan-Moss and his work on django.me, we now support human-edited deep-linking within documentation hosted on RTD.

Taking another page from Jacob's book, we seeded the index of our projects with their Intersphinx data, so a lot of references will automatically work. This works best with API reference docs, but anything people have put links to in their documentation should have been picked up. A couple of examples:

- http://pip.rtfd.org/git
- http://celery.rtfd.org/Task
- http://sqlalchemy.rtfd.org/relationship

If you go to a non-existent link on rtfd.org, you will be prompted to enter a suggested URL. This will help build the data, and make it more useful for everyone.

rtd command line utility

RTD has had an API for a while now, and with the addition of the support for rtfd.org, I thought it would be neat to make it easier to access docs from the command line. With a simple `pip install rtd`, you will get an rtd utility that will open docs on RTD. It supports 2 arguments, the first being a project name, and the second being a slug to append for the rtfd.org functionality. So like the example above:

```
-> rtd pip
Pip Installs Packages.
Opening browser to http://pip.rtfd.org/
-> rtd celery Task
Distributed task queue
Opening browser to http://celery.rtfd.org/Task
```

It hits the RTD API to see if the project is on the site, and only opens your browser if it doesn't exist. I hope that in the future we'll make it easy to upload a project from the shell, and more.

More docs

Since we are a documentation site, we've always had documentation. I've been adding more as time has gone on, and most of the features I'll be talking about today are already documented. I also broke the documentation up into sections for users of the site, and developers on the codebase, so it should be easier to find for everyone to find what they are looking for.

Conclusion

I think that RTD can be doing a lot more to help out the community with regards to documentation. I'll write another post about that soon. But if you are interested in helping out with the effort, all of the code is open source and we love people to contribute. Feel free to jump in #readthedocs on Freenode as well, if you have any questions or thoughts.

2.3.4 2010

A simple Perl IRCBot

A couple things I want to talk about. First of all, I will be participating in project52; which is a competetion to write a blog post in every week of the year. The last 2 years I have done the november post-a-day, and gotten about 25 of the 30 required posts. So hopefully writing twice that number of posts in 12 times the amount of time will be easy, right? Anyways, this is the first post in that series, so stay tuned for more regular and hopefully useful content :)

And since I don't like posting just basic updates, here is some perl code that I just recently dug up.

The code

I was thinking about adding IRC integration to a side project that I have been working on lately. I remembered that I had written something similar while in high school, and I've decided to throw that code up online, and clean it up a little bit. I don't really expect anyone to use it, but I think it's pretty neat, considering I wrote it 6 years ago.

The code is up over at github with basic installation instructions. It comes with a client in perl and python.

The idea is that the IRCBot has a basic TCP server in it, that you can use to send it messages across a network. So you can send a message crafted in the form of `password&server&channel&my sweet message`, and the bot will display it on the correct channel.

It uses POE, which I believe is perl's analogous idea to Twisted. I assume something like this is possible for Python, but I figured since it was already written, I should go ahead and use and release it.

The story

So the reason that this code exists is because I was a huge nerd in high school. I went to a computer class in the afternoons, where we each had our own computers, but the networks were locked down. This was in a time before I was good enough at SSH Tunneling, so I went ahead and wrote this code as a way to use HTTP to get into IRC.

The code released was half of it, it would site on IRC, and log all activity into a Mysql database. It also had a TCP server built in, so that I could ping it from another process/server and send messages out to other channels.

The second half was a basic web interface, which would pull and display all of the logs from the Mysql DB. It was broken down by channel, giving historical logs of each. It also had a firehose display that showed all incoming messages and what channel/server they were from. Along with this there was a form that hit a CGI script that sent a TCP request across to the client process and sent messages into the IRC channels. This allowed me to have two way communication into IRC from my web browser.

I happily used this to chat with people on IRC for my last year in high school. :)

Django Inspect: A generic introspection API for Django models

Django itself has shipped with a "semi-private" introspection API, `_meta`, for a long time. I have created a drop-dead simple wrapper on top of this.

The value of introspection keeps growing on me as I realize how it makes making truly reusable applications possible. It is an interesting intersection of duck-typing and interfaces. Basically, you can create functionality that will work with **any Django model**, as long as it has the correct values on them.

However, I present this as a very useful proof of concept, and I think a lot can be done with these ideas to improve on them. I recommend you just pull down the code and play with it. The ideas are very simple, but the power it gives you is vast.

The code is on Github with a very simple test suite showing usage.

What does it do?

The API is very simple. You pass in a instance of a Django model, and you can get off the values that you care about.

```
from django_inspect import base
intro = base.Inspecter(comment)
self.assertEqual(intro.content.field, 'comment')
self.assertEqual(intro.content.value, 'First here, too!')
```

This example is using a Django comment, as you can see. When you get a comment object, you want to see what the actual "content" of it is. Normally, this requires special casing in your code, or somewhere else. However, here we see it's just `intro.content.value`, to get the value, or `.field` to get the name of the content field.

The idea is that you have pluggable "parsers" that have names, which then map to fields on the model. By default, the name of the parser is checked, and any mappings you have passed in. Then it will go ahead and execute any custom logic that is associated with that parser.

So for this example, the "content parser" knows about comments, so it knows to check for the "comment" field for it's main content. This lets this mapping of content to fields to live inside the parser, and lets the user of the Inspecter to just say "I want the content".

Again, it's just easier if you read the code, it's really pretty simple.

Handling third party apps

Django Inspect also has the concept of mapping models to fields. So you can create a simple dictionary and pass it into your Introspection class, and it will map those keys to the corresponding fields. An example is worth a thousand words:

```
DEFAULT_MAPPINGS = {
'comments.comment': {
    'content': 'comment',
    'pub_date': 'submit_date',
     }
}
```

I call this the "Mingus use case". For example, if Mingus wanted to be able to introspect any of it's reusable apps for what it's "pub_date" or "content" fields were, it could ship with a mapping for all of the reusable app models, and then you would be able to write generic code that would work across all of those apps.

This is partially inspired by South's support for app migration directories

A complex example

Say I am using Nathan Borror's Fantastic Basic Blog. It has it's Blog Post model defined as such:

```
class Post(models.Model):
    STATUS_CHOICES = (
        (1, _('Draft')),
        (2, _('Public')),
    )
    title = models.CharField(_('title'), max_length=200)
    slug = models.SlugField(_('slug'), unique_for_date='publish')
    author = models.ForeignKey(User, blank=True, null=True)
```

```
    body = models.TextField(_('body'), )
    tease = models.TextField(_('tease'), blank=True, help_text=_('Concise text suggested. Does not ap
    status = models.IntegerField(_('status'), choices=STATUS_CHOICES, default=2)
    allow_comments = models.BooleanField(_('allow comments'), default=True)
    publish = models.DateTimeField(_('publish'), default=datetime.datetime.now)
    created = models.DateTimeField(_('created'), auto_now_add=True)
    modified = models.DateTimeField(_('modified'), auto_now=True)
    categories = models.ManyToManyField(Category, blank=True)
    tags = TagField()
    objects = PublicManager()
```

When I go ahead and create an inspecter class for this, I will be able to define what fields I want to map onto what. So for example, here the 'content' field of the blog post is actually called "body". I could create a simple mapping for this model, or I could modify the default parser to make the "content" field look for "body" models.

```
BLOG_MAPPING = {
'blog.post': {
    'content': 'body',
    'pub_date': 'publish',
     }
}

from django_inspect import base
ins = base.Inspecter(post, BLOG_MAPPING)
```

Now the following fields should have the following values:

```
ins.content.field: 'body'
ins.content.value: <Whatever my blog post is about>
ins.pub_date.field: 'publish'
ins.pub_date.value: <When my blog post was published>
```

Lots of room for improvement

There are a lot of interesting API niceities that could be added in on top of this code. I want to keep it really simple, however there is room for improvement. A couple that I have thought of:

- Expose this as a Proxy Model, where you would get a proxy model of your model with the introspection bits attached onto it.
- Make a descriptor so that you can have a pass through values do magical things on the Parsers
- Allow for complex parsers by having the parsers know about each other
- Make the Inspecter class know more about the parsers and be able to do more interesting things there
- Ship it with a default set of mapping that work for most reusable apps out there. Also have a "standard" way for apps to define mappings.
- Lots more

The whole idea of releasing this is to get feedback on what the actual API should look like. I think it's pretty awesome currently for the simple case, but for more advanced use, it's going to need to grow some features.

Conclusion

The whole idea behind this is that if your code is named or modeled sanely, it should "Just Work". However, if you have a crazy data model, or have to depend on wonky third party apps outside your control, it is incredibly simple to

map and introspect those models as well.

The other powerful idea is the application of semantics to models. I can query your model for the "content" or "tease" field, and be able to define exactly what that is. This lets me build interfaces and applications that "know" more about their data, even when that data is unknown at the time of writing.

This gives the application developer the power to write truly generic applications that will work with any suitable model. At least I hope so :). I have some other ideas that fall out from the implications of this introspection code that I will be talking about at Pycon, and probably doing a lightning talk. So feel free to find me and I probably won't shut up about it.

Large Problems in Django, Mostly Solved: Documentation

** [This is part of the 'Large Problems in Django Series <http://ericholscher.com/tag/largeproblems/>'_, see previous entries about: 'APIs <http://ericholscher.com/blog/2009/nov/11/large-problems-django-mostly-solved-rest-api/>'_, 'Search <http://ericholscher.com/blog/2009/nov/2/large-problems-django-mostly-solved/>'_, and 'Database Migrations <http://ericholscher.com/blog/2009/nov/6/large-problems-database-migrations/>'_]**

Django is well known across the open source community for it's stellar documentation. While the tech behind the documentation plays only a little role in how good it is, the tools behind both Python and Django's documentation is Sphinx. Luckily, we can all use Sphinx to document our projects, and I'd like to talk a little about why you might want to.

Why use Sphinx

Network Effects One of the big reasons is because it is becoming the standard documentation tool in the Python community. Once your projects documentation is in Sphinx, most everyone will know how to contribute to it. You will also be able to contribute to other projects easily as well. You can look through the Python and Django docs for examples of how to do neat things, and it is really the best solution to the problem.

It uses Restructured Text If you are writing plain text about python, more than likely you should be using Restructured Text. All docstrings are parsed for it, and you only need to learn this one markup language for all of your plain text needs. It even works great for blog posts, with most Django blogging engines supporting it. It is also easy to extend, and is generally a useful thing to know how to do.

Write once, compile to HTML, PDF, etc. By writing in Restructured Text, you write your documentation with metadata about what all of your text means. This then allows it to be transformed intelligently into other formats. This is how Django can provide HTML and PDF versions of the documentation all from the same source format. By rendering through LaTeX, you are given a large amount of flexibility in the style of your PDF output, allowing for really nice designs with a little effort.

Your docs are beautiful Sphinx has native support through Pygments for syntax highlighting most languages that exist. It also ships with support for themes, with the community providing themes out of the box to make your documentation look great. This is another place where having a critical mass of people behind the project makes your docs better.

Cross References With simple markup rules applied to your documentation, you get indexes and cross referencing for free. This makes your documentation much more discoverable, and useful for people who are browsing it. The Django documentation makes extensive use of this, making it easy to jump to the definition of a setting where ever it is referenced for example.

Lots and lots more. Please just go look at Sphinx, and read a little more about it. The Overview at the Sphinx page gives you a nice example of actual Sphinx docs, and points to lots of little tidbits of information. Sphinx has made documenting your project a real joy, and I can't recommend it enough.

The role of designers in the Django community

* *UPDATE**: There is a new thread about the roles and implementation of a Design Czar up on the Django Developers mailing list. Please contribute there as well, if you have thoughts and ideas.

There has been a recent discussion on the Django Developers mailing list about the role of designers in the Django community. I think that this is an interesting discussion that can come from this, and I would like to explain my thoughts on the issue.

This discussion came up in the context of redesigning the Django Admin, which everyone knows and loves. The UI is growing a bit out-dated, and there was talk of working to clean it up. This then turned into a discussion about how design proposals and improvements aren't taken as seriously as they should be by the community. I think there are a number of reasons that this happens, and I would like to take a look at them. My purpose here is to start a discussion about how to better integrate designers into the community, because they are a vital part of making our world more beautiful and efficient.

I don't trust myself to judge your work

The normal process for changes that go into Django is that a proposal is sent to the mailing list. There is a discussion that happens around them, and then if the code is produced, and it works, it gets committed. For design changes, I don't reply to these messages, because I don't have the skills or knowledge to judge the work. I think that a lot of people on these lists are in the same boat.

When someone sends a proposal to the list, and it doesn't get any replies, that feels like rejection. This happens more than it should, but it isn't anyones job to respond to these messages and say "sorry, I'm not qualified to critique your work". This happens with code proposals too, but I think it may happen more with design. This leads to designers forsaking the mailing list, and this problem perpetuates itself, by not drawing designers into the community.

Design is not special, except when it is

Part of the problem that seems to have come forward is that there is a feeling that design is "special". That it should be treated somehow differently in the process. As we know from history, even with all good intentions, different is never equal. So I think that we should work to fit design into the current scheme of how things work, instead of trying to adopt new ways of dealing with it.

When I look at the current Core Developers of Django, I don't see many people who are designers. As I said above, that fact that very few of the current core developers are well versed in the design realm, really hurts inclusion of design changes. This creates a lot more friction in the process of getting design changes into the code base.

I don't know if this idea is crazy, but should we have the concept of a "core designer". These would be people that the community trusts and knows have good taste, that would be an obvious person to make these design choices. I think that there is a problem when I have a design change for Django, and I really don't know who to talk to. There is an obvious authority (BDFL) for code changes, but I don't know if Adrian and Jacob are really the correct people to making these judgment calls on design?

I realize that this is open source, and "core designers" would be the same as developers, just people who care about the direction of the projects design. However, I think that having more design oriented people in the community in a more direct fashion would make it more obvious that design changes are welcomed and seriously considered.

I don't know how far we need to go down the path of making this explicit. However, most of the documentation about contributing is explicit about "code". This is another of those lines, where I don't know if it makes sense to

be explicit about design, having a "design" section in the contributing documentation, or if the implicit knowledge of core designers will make it obvious that we mean design changes there too.

The actual process

I don't want to talk about the actual design process, because well, I really don't know how it works. I think that once we integrate designers into the community better, the process for design will naturally fall out better.

Conclusion

I would like to point out that Django has some of the best designers of any open source community out there. I am lucky to work with a number of them on a daily basis, and I really think that they make our community special. So thank you guys for sticking with us.

This is a place where I could see Django leading the way in how to integrate design into the open source development process. Let's make a grand experiment, and see how it works out.

If you have thoughts, please join the discussion on Django Developers, so that the correct people hear your thoughts.

Large Problems in Django, Mostly Solved: Delayed Execution

[This is part of the Large Problems in Django Series, see previous entries about: Documentation, APIs, Search, and Database Migrations]

A lot of Django applications have tasks that they need to perform out of process. When you are executing a web request, if you try to do all the work that you need before returning to the user, your site will be increasingly slow. The answer to this problem is to fire off a request to do those tasks, while returning to the user in a reasonable amount of time. Celery refers to itself as a "Distributed Task Queue", and is the current best of breed in the Python realm.

Why Use Celery

Easy For the most basic functionality, all you need to do is:

- move your function into your tasks.py
- wrap it with a @task decorator
- call it with task.delay(*args) just like before.

Now, your task is magically running out of process and you can get on with whatever it is your code is meant to be doing.

Network Effects This is currently the best and most complete application in Python that does these things. A lot of people are using it, which means that features will be added consistently. There is also pretty good support in the #celery IRC channel, which usually has around 40-50 people in it. It is being actively developed and all other things being equal, using a tool with a community around it is much better.

Concurrency The celeryd daemon supports multiprocessing, which allows it to run multiple tasks at once. You can get "cheap concurrency" this way, by loading it up with tasks and having it execute them. You can also run multiple instances of celeryd across multiple servers, you can get your tasks that run concurrently across servers. Running multiple instances is also a good way of insuring redundancy in case one of your daemons goes down.

Monitoring One of the scary things about having remote execution of tasks is that if your daemon goes away, your site will appear not to function. Celery has an accompanying project called celerymon which provides monitoring services for Celery.

No more hacky cron jobs I don't know about you, but most of the time when I want something to be run in the background, cron is my go to choice. I'm ashamed to admit that I've written code that is meant to run in a cron job every minute checking for something to have happened. However, celery has most of the features that cron has, while giving you real support for deamonizing and delaying tasks. Being able to retry tasks is a great benefit is has over cron, so when something fails, you can run it again later.

Great documentation The celery docs are great, including everything from basic setup and example instructions to howtos. We put it into production at work, and the docs for using redis as a "ghetto queue" were great and worked the first try.

Lots more I highly recommend that you check out celery. Unless you are doing a small website like a blog, you more than likely have a use case for Delayed execution of tasks. **It's one of those things that once you have celery set up and running, you find more and more ways to use it over time. It is one of the best ways to increase the responsiveness of your website.** I've found that it can also clean up some of the other infrastructure you might have in place to do similar things.

Announcing Read The Docs

This year's Django Dash just came to an end, and I'm really excited about the project that we built. I'm sure the other teams are feeling just as stoked, because there is an amazing amount of awesome work that was done in the last 48 hours.

I'm really happy with the work we did, I think it is close to production quality. Last years project didn't even get a blog post because it was "almost done". This year I'm putting it out there because I think it is genuinely useful and pretty damn awesome.

Our team consisted of, besides myself, Charlie and Bobby who are an amazing dev and designer, respectively.

Read The Docs

Our Django Dash project solves a real problem in the Open Source community I think. I have a love affair with Sphinx, and it's really started to catch on as a cross-platform documentation tool. Our idea was to provide hosting for people's documentation, in a central place with nice tools built around it.

I know whenever I create a project, I have this moment where I think about documentation, and hosting it is a problem that is hard to solve. **I currently have a cron job running on my server pulling my docs every 5 minutes**, this is no way to host documentation.

We created Read The Docs to solve this problem. It will automatically build your documentation for you, if you put in a github or bitbucket URL. You can also use it to create Sphinx documentation on the site with some basic editing tools that we created.

Cool Features

Open Source This year's Django Dash required submissions to be open source, which I think is great. It gets a lot of knowledge from smart people into the world, and I think focuses the projects more on community problems. Feel free to take a look (again, written in 48 hours, be kind) our source and contribute, or laugh at us :)

Build your own docs If you have a really basic project that doesn't need a whole bunch of documentation, you can use our documentation builder to create the docs right on our site. We will host them for you and automatically rebuild them whenever you update them. This solves a problem for people with simple documentation needs. My teammate Charlie made some docs for his own project on the site

Host existing docs If you already have documentation in your project, but are hosting it in a crappy way, let us host it for you. We will update it in real time (see below) whenever your update it, and we have lots of neat features planned that will make it silly not to use our hosting. For example, here is a mirror of pip's docs.

Web Hooks Web hooks are pretty amazing, and help to turn the web into a push instead of pull platform. We have support for hitting a URL whenever you commit to your project and we will try and rebuild your docs. This only rebuilds them if something has changed, so it is cheap on the server side. As anyone who has worked with push knows, pushing a doc update to your repo and watching it get updated within seconds is an awesome feeling. If you're on github, simply put `http://readthedocs.org/github` as a post-commit hook on your project. Otherwise your project detail page has your post-commit hook on it.

Bookmarking I have a problem with Django's documentation, and it's that it is so big, I often find a page and then forget where I was when I need that information again. We added simple bookmarking so that you can find pages that you were on before. Check out the recently bookmarked pages

View Tracking Another feature is that we track which doc pages are viewed the most. This is a great hueristic to what pages are important and useful, and I think will be an interesting UI feature once we hopefully get more and bigger projects on the site. Not suprisingly, the project's own docs are currently the most viewed

Planned cool features

Full Text Search Once we have a critical mass of documentation, being able to search across all of it, based on tags and other attributes will be a killer feature. I'm really excited about the possiblities here, and think this will be the first big new feature that we will implement.

Quick browser editing When I find a typo in your documentation, there should be a 1-click process to be able to make an edit and send you a diff. We want to be able to support this with a nice UI. I think it will really increase the quality of documentation if there is a super easy way to update and edit existing documentation from it's rendered interface.

Mobile View

With we will be able to make a mobile theme that we can serve based on user agent. This will be another killer feature to hosting your docs on our site, because you'll get a kick ass mobile version for free.

Future I really hope that this utility becomes used by the community, because I think it is needed. I understand that large projects would want to and should host their own documentation, but for the 90% of projects that are small, I think this is a great solution.

Lessons Learned From The Dash: Easy Django Deployment

This is going to be a series of posts that talk about what I learned from the Django Dash. I think it's a really fun competetion that is also a great learning experience. I hope that this series catch on, and other people write about some of the things that they learned in the Django Dash.

What I learned

The thing that I learned about during my dash project was the awesomeness that is Gunicorn. It is an awesome HTTP server that I think has really solved the "how do I deploy Django" problem.

Here are the steps involved in deploying a site using the gunicorn:

- pip install gunicorn
- Add 'gunicorn' to your installed apps
- ./manage.py run_gunicorn -b 127.0.0.1:1337 –daemon

It really is that simple. Gunicorn is the fastest way to having a production ready web server serving your site that I've found in the Django realm. However, Gunicorn by itself isn't production ready. It is recommended to deploy something in front of it. We used Nginx, which is another super simple web server.

Here is basically the simplest possible configuration of nginx that will work for your gunicorn backend server.

```
server {
        listen 80;
        server_name  example.com;
        access_log  /var/log/nginx/example.log;

        location / {
                proxy_pass   http://127.0.0.1:1337;
        }
}
```

After you restart Nginx, you should be able to hit your server at port 80 and have it be serving your Django web app. This allowed us to get our application into production during the dash in about 10 minutes, which was a great time saver.

I'd be curious if people have had any trouble with Gunicorn in deployment, because as far as I've seen its production ready. As a "first Django deployment" set up I think it's hard to beat. I've also noticed that is uses significantly less RAM than an Apache/mod_wsgi set up (I know this can be configured away, but by default it's much better). This is great for the memory constrained deployment platforms a lot of us are running on.

A better webhook for code hosting

I have written a couple of different services that have needed to be required when your repository has had code committed to it. The normal path of getting this to happen is to ask your users to add your special URL to their list of post-commit hooks for their repository. However, once you have 3 or 4 or 10 services that need to do this, it becomes cumbersome. **If I am a user that has 5 repos and I want to use 5 services, this is 25 times that I need to copy/paste some URLs into a form on a website.**

I think that a publish subscribe model is better here, because that way the end user doesn't need to constantly be caring about who is listening to their commits. I think that Pub Sub Hub Bub sounds like it does what I want. However, I think it should be baked into the tools.

I am imagining an opt-in service for your repository (and other things), that gets pushed to when a user commits, and I can subscribe to. So an example workflow would be

- User adds their repository to the PSHB or whatever service (push.github.com/eric/my_repo)
- I POST to the Hub with my URL I want to be pushed to on commit
- When a User commits, github pings the PSHB Hub, which pushs the commits to anyone listening.

This allows my service to listen in to your repositories updates without having to force you to go through a bunch of hassle. This just feels like a better fit for the current webhook model that we have.

I think that this is what PSHB does, but it is more focuses on RSS/ATOM, instead of just being a replication hub for JSON data. I assume that PSHB could be shoehorned into this task, and it would make the atmosphere of apps around code a lot easier to write.

As a side note, it would be pretty awesome if this same service allowed bitbucket, github, google code, and launchpad to post data into it, but sanitized it so the listener on the other side only had to support one format of data.

Edit: Looks like Superfeedr has already done this.

Lessons Learned From The Dash: Nginx SSI

Continuing from my previous post about Django Dash, I will be talking about another thing that I learned from the dash. This isn't as big of a post, but just something that we ran into that caused us some trouble.

We are hosting documentation for other projects, and we needed a way to put a toolbar on the top of the pages so users can still get around our site. We started out by hacking this into the sphinx template as static html, which was annoying because it didn't let us determine if the user was logged in, owned the project, etc. So we decided to load the header dynamically.

Using Nginx Ghetto ESI

We were deploying on Nginx, and luckily this post about Ghetto ESI with Nginx laid it out pretty well. We only ran into one problem with this approach, and it was minor.

The implementation is that we are hacking the SSI tag into the Sphinx template's we are rendering at the top of the page.

```
{% block relbar1 %}
<!--# include virtual="/render_header/" -->
{{ super() }}
{% endblock %}
```

Then you simply add a `ssi on;` into your Nginx configuration for your site. This makes the page call /render_header/ to fill out the top of the page when the user hits a documentation page.

The problem

The problem with this is that this doesn't work in your local testing environment. So Joshua's post earlier has a piece of middleware that you can include in your Django project to emulate the Nginx include behavior.

We turned this on, but every once in a while our pages were getting randomly cut off halfway through the response. We looked into it a little bit, and figured out that it was because of the response's Content Length header was still set to the old value. So our updated middleware simply added one line to the reponse.

```
response['Content-Length'] = len(response.content)
```

This allowed our pages to render correctly in testing, and then in production Nginx will hit the include before Django sees it, so the middleware never processed. **If you are changing the content of your response in middleware, make sure that you update the Content-Length header.**

New features on Read The Docs

Since the Django Dash ended, We've been working on adding some requested new features to Read The Docs. There are a couple of major ones that we have added that I'd like to talk about.

hg and svn support

We've added support for all of the version control systems that people have requested. When you sign up or edit a project, you can now tell us which VCS you are using, and we'll use that to check out your code to build your documentation.

There are two libraries that I wish existed: One to smartly parse urls into the correct repository, and a standard VCS abstraction that lets me treat all VCS' as the same. These would integrated presumably, so I could do `vcs clone <url>` and `vcs update`, and it would "Just Work"

Whitelisting support

By default we don't execute any python code when you import your project. This is a security precaution that we take, so that means disabling all extensions by default. A lot of people are using autodoc and some other extensions, so we have added the ability to whitelist projects so that they are built without any sanitization on our part.

A sweet logo

Our designer Bobby made a sweet logo for the site, and has been adding lots of little visual tweaks that I'm not qualified to talk about :) However, it seems that whenever I look at the site, it gets a little prettier, that's how I usually know that design is being done.

Subdomain and CNAME support

This is a really exciting one for me, because I've been learning more and more about sysadminery lately, and this was a fun little mixing of the two. For any project, you can now access the projects documentation at `<slug>.readthedocs.org`, for example, pip's documentation is now at pip.readthedocs.org.

Now that we have subdomain support, this makes supporting CNAME's really simple. So if you have your own domain name, and you'd like those docs to point to us, it's simple. All you need to do is add a CNAME record for that domain in your DNS settings to point at your subdomain URL. Pip is another good example here, pip-installer.org now is hosted on RTD. Other notable examples are djangotesting.com and djangowoodies.com :)

All of this support is implemented in middleware and only ends up being about 25 lines of code. There are going to be some complications when we try to add multiple version support, and internationalization, as you can't really specify those well on subdomains. I see us having a "default" project version, as well as letting you have other versions hosted as well.

RTFD.org

"RTFM" is a well known term in the programming community. Luckily when we were scheming up names for our project, we noticed that rtfd.org was available. We went ahead and bought it, and now we're supporting `<slug>.rtfd.org` and `rtfd.org/<slug>` redirects that go to your RTD page. This is a nice little keystroke saver, as well as a fun was to refer people to your documentation. This is implemented simple in an Nginx server directive. I'm sure it can be improved upon, but it's working well at the moment.

I think adding the ability to have "smart slugs" here would be interesting, so it could actually perform a search or something, and return the top result, kinda like LMGTFY. This could be a neat feature to add on.

LaTeX support

LaTeX is a pain to get setup, so if you want to support rendering LaTeX, we now support that as well. The sympy has been testing their docs on RTD, and have helped me clean up a bunch of bugs. The Geometric Algebra section shows off some of the LaTeX goodness.

* *Update:** Just for kicks, we currently have 120 users and 80 projects currently hosted on Read The Docs. At least a couple of these are using RTD as their official documentation host. I'm pretty happy with the uptake that's already happened in the last 2 weeks (wow, that little?!). Thanks everyone for checking it out!

* *Update 2** We also have a new IRC channel if you need help or have questions, it's **#rtd on irc.freenode.net**.

Conference Fun

It's conference season and I realized that I haven't talked about any of the ones that I've been to or am going to, so I figured it would be a good post.

Djangocon US

Djangocon US is just around the corner, and I'm getting excited about going back to Portland for another year. The conference is being held in the second week of September (7-9th) with sprints afterwards. I will be speaking again at Djangocon, talking about the awesome applications that the Python community has put together. Modeled and named after my blog series "Large Problems, Mostly Solved", and the full desciption is available on the Djangocon site. The full schedule of speakers has a little bit for everyone.

My friend Danny has been highlighting some of the talks and other reasons he's also excited about going this year on his blog. It's a great oppertunity to meet up with all the members of the community and see them present ideas that they have been working on, and it really makes you appreciate the scope of the Django world. I highly recommend going for anyone who is interested in or doing Django developement.

Strange Loop

Strange Loop is a conference right around the corner in St Louis, Mo. It is a general developer conference, without any kind of specific focus besides being awesome. The keynotes that I'm super excited about are Douglas Crockford (Author of Javascript: The Good Parts) and Guy Steele (whos Growing a Language talk is excellent).

I think a crew of us from Lawrence are going to go, and the full speaker list doesn't fail to disappoint. I think it will be a really interesting experience, with lots of different communities coming together.

Djangocon EU

Djangocon EU is the Django communities conference in Europe. It was held in Berlin this year, and was a smashing success. It was the first conference run by the community, and I think the organizers did a fantastic job with the food, talks, venue, and general galavanting around Berlin that ensued.

I realized that I forgot to post the slides to my talk afterwards, so I'll go ahead and post it here for posterity. It is a talk about Continuous Integration, Testing, and some of the tools and utils in that realm. If you haven't seen it before, I hope you learn something:

Making the most of your Test Suite

View more presentations from ericholscher.

I know that I love software conferences, especially Open Source oriented ones (I've never been to another kind :). So I hope to see (or that I saw you) at one of these awesome events.

Djangocon Talk

I just gave a talk title "Large problems, Mostly solved", which you will recognize if you've been reading this blog for a little while. I took my past series of Large problems posts, and expanded on them into a full talk.

The video and slides are posted below. Thanks for the A/V team kicking ass this year and getting the videos posted in amazing time. Also many thanks to the conference organizers and volunteers who managed to make it all happen, it was a great time!

Large problems, Mostly Solved

View more presentations from ericholscher.

Virtualenv Tips

Virtualenv is a project that is indispensable for most Python devs these days. I am writing down some tips here so mainly for personal reference, and because I found them useful.

Virtualenv & Upstart

For my ReadTheDocs project, I was wanting to run celery as a background process that processes documentation building. I'm running Ubuntu, so their built-in upstart service seems like a logical choice. I really like upstart because of it's simple configuration, but it is rather undocumented (this Stanzas) page is a useful starting point).

Carl Meyer pointed out to me that in order **to get inside the context of a virtualenv, you don't need to munge your pythonpath or anything, but simply run the correct script from inside the virtualenv**. So a simple `/path/to/virtualenv/bin/django-admin.py celeryd` was all that was needed to get inside the virtualenv's context.

This also true of the python executable inside your python directory. `/path/to/virtualenv/bin/python` will allow you to run any python script inside of that virtualenv's context.

I also wanted to be running my jobs as the user for that site, so `sudo` is the correct tool for that. The final file ended up looking like this:

```
description "Celery for ReadTheDocs"

start on runlevel [2345]
stop on runlevel [!2345]
#Send KILL after 20 seconds
kill timeout 20

script
exec sudo -i -u docs django-admin.py celeryd -f /home/docs/sites/readthedocs.com/run/celery.log -c 2
end script

respawn
```

The only other interesting bit there is the `-i` option to sudo, which means it will run the command as a login shell, picking up the environment for the user. This means it has the correct path and everything set, so that `django-admin.py` works without an explicit PATH.

Adding site-packages in after initial creation

Frank Wiles ran into this problem on IRC, where he wanted to add in the site-packages after creating a virtualenv with `--no-site-packages`. It turns out to be really simple, in that you only have to remove the `no-global-site-packages.txt` in the `lib/python2.x` directory inside the virtualenv. After that virtualenv will go ahead and fallback to the global site packages happily.

I'd imagine this would work the other way as well, if you want to not have your site-packages included, you could add this file into your virtualenv.

Use virtualenvwrapper

Virtualenvwrapper is a nice set of extensions around virtualenv. It gives you handy command line helpers, like `workon` which autocompletes the names of your virtualenv's. It has its own Tips and Tricks page that has some neat ideas about how to improve your virtualenv experience.

Deploying Virtualenv

Deploying with virtualenv and apache has been well covered. I recommend this Caktus post that gives some good examples.

The main idea however, is that you make sure your the virtualenv's pythonpath is on your pythonpath, or that you are running the virtualenv's python when you run your webserver. For apache, in your wsgi file, you generally do something like:

```
site_packages = os.path.join(PROJECT_ROOT, 'env/lib/python2.6/site-packages')
site.addsitedir(os.path.abspath(site_packages))
```

For a gunicorn deployment, you would do something along the lines of `/path/to/virtualenv/bin/python manage.py run_gunicorn`.

Your tips

I'd love to hear your tips about how to use virtualenv in the best way possible. I know that my workflow is probably lacking, and these aren't all or even many of the neat things you can do with virtualenv.

Building a Django App Server with Chef: Part 3

Alternate title: Show the world what you've got.

This is Part 3 of my Chef tutorial. Today we're talking about deployment. You can check out the first 2 parts of the series:

- Part 1: Chef Beginnings
- Part 2: Python environment buildout

Today's code will be in the git repo under the tag blog-post-3.

What we'll need

We'll be taking the Django application that we have on the server and actually deploying it. Let's make a list of what we'll need:

- A web server to sit in front and proxy requests
- A WSGI server
- A way to keep both of these things running
- A caching layer

We'll be using Nginx, Memcached, Upstart, and Gunicorn. This is my preferred deployment stack as of late, mainly because of the simple setup.

Let's get started

A web server

Getting Nginx up and running should be old hat by this point. We're going to need the package and service Resources, which will tell Chef to install and run it.

`cookbooks/main/recipes/nginx.rb`

```
package "nginx" do
    :upgrade
end

service "nginx" do
  enabled true
  running true
  supports :status => true, :restart => true, :reload => true
  action [:start, :enable]
end

cookbook_file "/etc/nginx/sites-enabled/readthedocs" do
  source "nginx/readthedocs"
  mode 0640
  owner "root"
  group "root"
  notifies :restart, resources(:service => "nginx")
end

cookbook_file "/etc/nginx/nginx.conf" do
  source "nginx/nginx.conf"
  mode 0640
  owner "root"
  group "root"
  notifies :restart, resources(:service => "nginx")
end
```

As you can see, we're providing our own nginx.conf and a readthedocs site configuration. I'm not going to paste these in, as they are pretty application specific, but you can look at them on Github if you're curious. I also wrote about it a while back.

The only new part here is the `notifies` command, which is pretty nifty. It basically means that whenever you change the nginx.conf file, it should restart Nginx, which is a really nice feature.

A WSGI Server

Yesterday, when we installed the deploy_requirements.txt with pip, it pulled in gunicorn. So we actually have Gunicorn already installed in our virtualenv, waiting for us to use it. The only difference is I actually committed a change to the ReadTheDocs source so that it will pull Gunicorn from the git master, which I'll explain below.

Upstart

Note: I use upstart because it ships with Ubuntu, so you don't need to install a separate package. However, it has pretty horrible documentation, with the Stanzas doc probably the best clue as to what it supports.

Here is where things get interesting. I spent a bunch of time trying to get gunicorn and upstart to play nicely yesterday night, but it wasn't working. I went on the #gunicorn IRC channel on Freenode today, and talked with benoitc. He was awesome and patched gunicorn to work with Upstart for me.

Here is the upstart script that we're using to keep gunicorn running:

`cookbooks/main/files/default/gunicorn.conf`

```
description "Gunicorn for ReadTheDocs"

start on runlevel [2345]
stop on runlevel [!2345]
#Send KILL after 5 seconds
kill timeout 5
respawn

env VENV="/home/docs/sites/readthedocs.org"

#Serve Gunicorn on the internal rackspace IP.
script
exec sudo -u docs $VENV/bin/gunicorn_django --preload -w 2 --log-level debug --log-file $VENV/run/gu
end script
```

As you can see, an Upstart script is a pretty clean way to do this. If you've ever tried to write an old SysV-style init script, this will look beautiful. You'll notice that we aren't passing the –daemon parameter to gunicorn, this is because upstart will background the process for us, and keep track of everything, so we don't need gunicorn's daemonizing behavior.

It should be pointed out how awesome it is that we can run a production ready WSGI server with a single line of bash. If you've ever set up a mod_wsgi install, needing to fuddle with your apache.conf and a WSGI file and everything makes it a chore. This is quite simply the easiest way to deploy a WSGI application.

Then we need some additions to `cookbooks/main/recipes/readthedocs.rb`:

```
cookbook_file "/etc/init/readthedocs-gunicorn.conf" do
    source "gunicorn.conf"
    owner "root"
    group "root"
    mode 0644
end

service "readthedocs-gunicorn" do
    provider Chef::Provider::Service::Upstart
    enabled true
    running true
    supports :restart => true, :reload => true, :status => true
    action [:enable, :start]
end
```

Here you can see we're doing a similar thing to the other service declarations. We however need to tell Chef to use Upstart for this service, instead of defaulting to init.d. Other than that, everything here should look similar to the other files and services we've set up.

Memcached

As you would expect, installing memcached is just like nginx:

`cookbooks/main/recipes/memcached.rb`

```
package "memcached" do
    :upgrade
end

service "memcached" do
  enabled true
  running true
  supports :status => true, :restart => true
  action [:enable, :start]
end

cookbook_file "/etc/memcached.conf" do
  source "memcached.conf"
  mode 0640
  owner "root"
  group "root"
  notifies :restart, resources(:service => "memcached")
end
```

The memcached.conf is so short, I might as well include it here:

```
-d
logfile /var/log/memcached.log
-m 64
-p 11211
-u nobody
-l 127.0.0.1
```

Memcache's config file is pretty neat, because it's basically just a list of arguments to pass to the daemon when it's started. A little bit like a pip requirements file is just commands to pass to pip install when it's run.

Wrapping up

Now that you have these awesome new recipes, and additions to old ones, we need to make sure they're actually being run. Your run_list in your node.json file should now look something like this:

```
"run_list": [ "main::default", "main::python", "main::readthedocs", "main::memcached", "main::nginx"
```

At this point, it's pretty neat. I can run a `fab install_chef update`, wait about 5 minutes, and go from a freshly paved server to a fully functioning app server.

Tomorrow we'll be adding some monitoring and auxiliary niceties. This includes setting up Munin, Celery, generating the /etc/hosts file, and throwing in a little .bashrc magic to make the user experience nicer.

There were a couple of questions yesterday about databases and other things. My current problem is running an application server, which is what I've accomplished. However, with my new-found love affair for chef, I will definitely be making my Database/Utility box into a chef configuration really soon. I might not write it up in so much detail, but hopefully you've learned enough from this series that I can just publish the code.

Building a Django App Server with Chef: Part 4

Alternate title: There's no place like home!

This is Part 4, the final part, of my Chef tutorial. Today we're talking about the odds and ends left over to make the server nice to use. You can check out the first 3 parts of the series:

- Part 1: Chef Beginnings
- Part 2: Python environment buildout
- Part 3: Deployment

Today's code will be in the git repo under the tag blog-post-4.

What we'll need

So we have our app server up and running, and ready for traffic. Now we just need to add some other bits around the outside for it to be fully functioning and nice to use.

- Monitoring with Munin
- Background tasks with Celery
- A firewall for security
- A /etc/hosts file for talking with other nodes
- A .bash_profile file so that when you shell in you'll have a nice environment

Let's get started.

Monitoring with Munin

For doing monitoring with munin, we're going to need to learn our final Chef concept, which is Templates. You should be pretty familiar with them already, except they use Erb, which is a template language that lets you embed Ruby.

We're only going to be configuring the Munin node here. This assumes that you have a munin server running on another machine that you want to give access to monitor your new app server. These configs depend on you putting an entry like this in your node.json, which points at the IP of the master server:

```
"munin_servers": ["10.177.243.34"],
```

Then here is how you would write the Recipe.

`cookbooks/main/recipes/munin.rb`

```
package "munin-node" do
    :upgrade
end

service "munin-node" do
  enabled true
  running true
  supports :status => true, :restart => true, :reload => true
  action [:enable, :start]
end

if node.attribute?("munin_servers")
  template "/etc/munin/munin-node.conf" do
    source "munin-node.conf"
    mode 0640
    owner "root"
    group "root"
```

```
    variables :munin_servers => node[:munin_servers] || []
    notifies :restart, resources(:service => "munin-node")
  end
end
```

The template Resource here is the interesting part. We're surrounding it with a conditional, that makes sure that we're defined a 'munin_servers' key in our node.json. Then we're saying that we're going to render the munin-node.conf file with the source template 'munin-node.conf'. This template will be given the extra varibale 'munin_servers', which is passed in using the variables attribute.

Template are placed inside the cookbook in a similar place to files.

`cookbooks/main/templates/default/munin-node.conf`

```
<% @munin_servers.each do |server| -%>
allow ^<%= server.to_s.gsub(/\./, '\.') %>$
<% end -%>
allow ^127\.0\.0\.1$

host *
port 4949

log_level 4
log_file /var/log/munin/munin-node.log
pid_file /var/run/munin/munin-node.pid
background 1
setsid 1
user root
group root

ignore_file ~$
ignore_file DEADJOE$
ignore_file \.bak$
ignore_file %$
ignore_file \.dpkg-(tmp|new|old|dist)$
ignore_file \.rpm(save|new)$
ignore_file \.pod$
```

The interesting part here is the iteration over the munin_servers list. It's just doing a simple ruby loop, and then outputting the IP address that it contains into the format that munin's configuration file expects.

Note: This data-driven template rendering is a really powerful idiom, and one of my favorite parts about Chef. This allows you to add a new server to your pool, and have all of your configuration files updated automatically across all your server. This is hugely powerful, and one of the primary wins of Configuration Management. This will be shown to better effect in the /etc/hosts file later.

Installing Celery

Installing celery is much akin to Gunicorn that was discussed yesterady. The dependencies were installed from our pip requirements file, and we just need to make it run in upstart. We'll be doing that with the following setup.

Additions to `cookbooks/main/recipes/readthedocs.rb`

```
cookbook_file "/etc/init/readthedocs-celery.conf" do
    source "celery.conf"
    owner "root"
    group "root"
    mode 0644
```

```
    notifies :restart, resources(:service => "readthedocs-celery")
end

service "readthedocs-celery" do
    provider Chef::Provider::Service::Upstart
    enabled true
    running true
    supports :restart => true, :reload => true, :status => true
    action [:enable, :start]
end
```

cookbooks/main/files/celery.conf

```
description "Celery for ReadTheDocs"

start on runlevel [2345]
stop on runlevel [!2345]
#Send KILL after 20 seconds
kill timeout 20

script
exec sudo -i -u docs django-admin.py celeryd -f /home/docs/sites/readthedocs.org/run/celery.log -c 2
end script

respawn
```

There isn't anything new or interesting here. Just more of the same as before, to get another piece of infrastructure up and running.

A ghetto firewall install

I'm a big fan of not enabling services that aren't running as a fundamental security practice, but having a basic firewall to make sure that those are the only ports open isn't a bad idea either. I'm not a great expert, so this is probably the weakest part of my knowledge in this series, so take it with a grain of salt.

My favorite firewall utility is ufw. It makes managing your firewall really simple. Here is my super basic way to configure my firewall, it pretty much sucks :)

cookbooks/main/recipes/security.rb

```
package "ufw" do
    :upgrade

service "ufw" do
  enabled true
  running true
  supports :status => true, :restart => true, :reload => true
  action [:enable, :start]
end


bash "Enable UFW" do
user "root"
  code <<-EOH
  ufw allow 22 #SSH
  ufw allow 80 #Nginx
  ufw allow 4949 #Munin
```

```
  EOH
end
```

As you can see, we're just enabling SSH, Nginx, and Munin. If we need to install any more packages, we'll need to expicitly open a port, which is usually good to remind me that I'm doing it.

/etc/hosts

Whenever I'm in the cloud, I find keeping track of my other servers to be a pain. You generally want to use the internal backplane to communicate between your servers, so I use the /etc/hosts file to make that simple.

We're going to depend on an entry in your node.json that looks something like this:

```
"all_servers": {"Golem": ["10.177.234.234", "173.203.234.234"],
                "Chimera": ["10.177.234.234", "204.232.234.234"],
                "Hydra": ["10.177.234.234", "173.203.234.234"] }
```

Which is a mapping of all your servers, with their internal and external IPs. This will be useful to have for lots of different recipes, and it would be nice to autogenerate this, but when you only have a few servers it isn't so bad.

The rest of out hosts configuration looks like this:

Addition to `cookbooks/main/recipes/default.rb`

```
if node.attribute?("all_servers")
  template "/etc/hosts" do
    source "hosts"
    mode 644
    variables :all_servers => node[:all_servers] || {}
  end
end
```

`cookbooks/main/templates/default/hosts`

```
127.0.0.1     localhost localhost.localdomain

<% @all_servers.each_pair do |name, ips| -%>
<%= ips[0] %> <%= ips[1] %> <%= name %>
<% end -%>
```

As you can see, when we add a server to the all_servers hash, it will propogate out to the /etc/hosts file of our app server. This makes me really happy, and showcases some of the more advanced use cases of Chef.

Customizing your shell

Now that we have the server all set up, it won't be much good if it isn't nice to use when we shell in. So here is how I go ahead and add in some nicities to bash for when you log in.

Addition to `cookbooks/main/recipes/readthedocs.rb`

```
cookbook_file "/home/docs/.bash_profile" do
    source "bash_profile"
    owner "docs"
    group "docs"
    mode 0755
end
```

`cookbooks/main/files/default/bash_profile`

```
. .bashrc

export PIP_DOWNLOAD_CACHE=/tmp/pip
export DJANGO_SETTINGS_MODULE=settings
export PYTHONPATH=$PYTHONPATH:~/sites/readthedocs.org/checkouts/readthedocs.org
export EDITOR=vim

. sites/readthedocs.org/bin/activate

cd ~/sites/readthedocs.org/


alias chk='cd /home/docs/sites/readthedocs.org/checkouts/readthedocs.org'
alias run='cd /home/docs/sites/readthedocs.org/run'
```

First off, we're sourcing the .bashrc file, so that we get all the nice things it provides, like a colored PS1. Then we're setting some environment variables so that django-admin.py and pip work nicely. Then we activate our virtualenv and switch into it's base directory, so we're always starting where we want to be on login. Then we just have a couple of aliases for easy navigation around.

I like how this makes the user experience of shelling into the server a lot nicer, and makes the manual workflow that you'll eventually have to fiddle with really nice.

Conclusion

So that's the end of this tutorial. I hope that it was instructive in learning Chef, as well as providing some insights into the deployment of a Django application. Tomorrow (or if I'm too tired, next week), I'll be providing some thoughts on how I think chef treated me, and how I feel about the build out.

Site upgrades

Today I went ahead and flipped the switch on a couple of server migrations I've had queued up. One of these updates is moving ReadTheDocs over to its own dedicated server, that I built up over the week in my Chef Tutorials.

Over at RTD, you won't notice too many changes, other than it should be FASTER! I had a bunch of sites running on an underpowered server, and now I have it set up nicely, and running on it's own machine, it's chugging along great.

The other change is that I migrated my blog (what you're reading!) over to Mingus. I was running an oold copy of django-basic-blog, which is what Mingus is based off, so the migration was easy. I moved it over from my legacy Slicehost account onto my new server infrastructure that I've been building. There is also a slight refresh of the theme of the sight, mainly the Mingus defaults poking through on top of my old theme.

The other bits you might notice is that my code snippets should now be syntax highlighted. It seems pygments gets confused sometimes, but it's better than it was.

Please report any bugs that you see on either of the sites above to me on Twitter, or here in the comments. Thanks!

Correct commands to check out and update VCS repos

In my work in ReadTheDocs, we now support all of the major VCS repositories: svn, bzr, hg, and git. At this point in time we're only checking out the repos to their default branches, and then trying to trying to update them again to another revision. While writing this code I have had at least 3 different bugs that caused the repos not to be updated correctly. So I'm going to detail here the exact code that allows me to do this for each of these types of repos, hopefully so that when you or I need to do this in the future, we can at least start from here.

Let me know if any of these are wrong, because they probably are.

Git

Checking out a repo:

```
git clone --depth=1 <remote_url> <local_filepath>
```

Updating a repo:

```
git --git-dir=.git fetch
git --git-dir=.git reset --hard origin/master
```

I'm specifying the –git-dir here because my master repository is git as well, and I don't want to risk the git commands cascading up and applying to the outer repository. I'm also specifying –depth=1 so that I don't clone the entire repository, but only the latest commit. I don't need the history, so I'm doing this. As you can tell, I'm more familiar with git than the other VCS systems here.

Svn

Checking out a repo:

```
svn checkout <remote_url> <local_filepath>
```

Updating a repo:

```
svn revert --recursive .
svn up --accept theirs-full
```

I ran into problems here where I was calling revert without recursive and it wasn't doing anything! You need to do this from the top-level of the repo, and it will make sure all the state lower in the repo is reverted.

Bzr

Checking out a repo:

```
bzr checkout <remote_url> <local_filepath>
```

Updating a repo:

```
bzr revert
bzr up
```

This one is nice and easy.

Hg

Checking out a repo:

```
hg clone <remote_url> <local_filepath>
```

Updating a repo:

```
hg pull
hg update -C .
```

Again, a slightly different syntax to make sure that you're deleting all the files, and with the update command.

I hope this helps people in the future at least get to the point where they can pull down code and update it. My next task is figuring out how to support branching in all of the different repositories which is going to be fun, because they have different filesystem structures.

Using Haystack to index non-database content

Over on ReadTheDocs, I wanted to build search around the documentation that we're hosting. I chose Haystack and Solr for this, because it's the best way to do search in Django these days. However, I've only ever used Haystack to index content that is in the database. I thought about trying to add all the rendered HTML from the documentation into the database, but that was a non-starter.

I ended up adding a ImportedFile model to the database, which would contain the metadata for the HTML file:

```
#!python
class ImportedFile(models.Model):
    project = models.ForeignKey(Project, related_name='imported_files')
    name = models.CharField(max_length=255)
    slug = models.SlugField()
    path = models.CharField(max_length=255)
    md5 = models.CharField(max_length=255)
```

This allows me to link the SearchIndex in haystack to a model. Then the interesting part is in the Haystack SearchIndex, where I override the prepare_text method, allowing me to read the data in from the filesystem instead of from the database.

```
#!python
class ImportedFileIndex(SearchIndex):
    text = CharField(document=True)
    author = CharField(model_attr='project__user')
    project = CharField(model_attr='project__name')
    title = CharField(model_attr='name')

    def prepare_text(self, obj):
        full_path = obj.project.full_html_path
        to_read = os.path.join(full_path, obj.path.lstrip('/'))
        try:
            content = codecs.open(to_read, encoding="utf-8", mode='r').read()
            return content
        except IOError:
            print "%s not found" % full_path

site.register(ImportedFile, ImportedFileIndex)
```

This means that I don't have to bloat my database with all my rendered HTML, but have the full HTML stored in Solr which works for querying.

Required Reading

At work, we have a wiki page that's called Required Reading. Named after that oh-so-lovely tradition in high school or college of having books that you needed to read over the summer before you started class. The idea being that they are relics of the culture of the company, and if you read everything, you will understand a lot about how work (and play) is done.

I think this is something useful that most companies should have. It was basically split into two parts: Silly and Serious. The silly parts are so that you can understand all of the inside jokes and references that people are bound to

make. The serious parts are more philosophy and thoughts behind how we write code and do things.

Silly

Being a python shop, this Youtube Monty Python playlist is a must watch. It's got most of the famous Python gags, and being knowledgeable about the languages namesake makes you a more rounded human being. Speaking of namesakes, Django Reinhardt is a great jazz player, so I recommend you learn about him as well. He was quite the bad ass.

After that, there are just some of the classic online videos that everyone should watch:

- DJ Ango
- Bill O'Reily DO IT LIVE (remix) for when you're doing it live.
- Nerf guns gone wild. Again.
- SCHNAPPI. Cutest Crocodile Ever.

and such.

Serious

The serious posts are a little more relevant, because they are more about the philosophy of how to code. I have been looking through a lot of really great posts on this subject lately, and these are some of my favorites:

- Code like a Pythonista (which links to pep8 and pep20)
- Pylons Unit Testing Guidelines
- The Art of Unix Programming

Then there are some of the more topical guides that give a good knowledge and understanding of some of the fundamental things that you do at work. The unit testing guide above is a good example, but there are a few as well:

- Richardson Maturity Model (for REST)
- HTTP Caching
- Understanding the GIL - Talk by David Beazley
- Pro Django ($$)

I think that having a guide to the culture of the company is really useful for people that are getting started. Plus I think it's a good way to remind people of what the values are of your team. Hopefully silliness is valued as much as good work, and you have a place to go when you want to see silly bits of the past.

I'd love to hear if other people have other ways of introducing culture to their companies. I'd also love to see some other really good topical guides on things that you may love, as I'm sure there are tons more out there.

Celery Tips

Following on yesterday's post about Virtualenv Tips, I will be talking about celery tips. Yesterday I talked about how to run celery with upstart easily, and today I'll be expanding on that below as well as talking about how to set it up using supervisord.

Note: Also interesting, I wrote a Big list of django tips back in 2008, that still has a lot of good information.

Running celery in development

When you run celery in production, you should be using a queue on the backend. However, when you're running celery in development, it's nice to execute the code paths, but not actually need a queue. This is where the CELERY_ALWAYS_EAGER setting comes in handy. It makes celery run the code in process, but will make sure your code paths work correctly.

I talk about this and more in my djangocon talk.

Killing long running tasks

On ReadTheDocs I would run into problems with celery tasks never returning. Luckily, celery has a way to handle this. The CELERYD_TASK_TIME_LIMIT setting lets you set the number of seconds that a task can run until it is killed. This is nice to make sure that a run-away task won't take down all your backend processing.

Use the JSON serializer for interoperability

I was talking on IRC to Eric Florenzano and he mentioned that you should use the json serializer if you want to be able to add celery tasks from other languages.

This allows you to use another language to put a message that looks like a celery task in the queue, and it should just work.

Explictly set the number of clients

When you run celery, it defaults to having the number of workers equal to the number of cores the machine has. If you are running multiple queue workers on the same machine, it is a good idea to use less. You can set this with the CELERYD_CONCURRENCY setting, or passing `-c<num>` on the command line.

Running against multiple databases with supervisord

At work we run a bunch of different sites on multiple databases. When we were figuring out how to deploy celery, we needed a good way to make sure that celeryd was always running, and we needed multiple celery daemons for each of our databases. We have written our tasks to run against multiple sites on the same database in order to reduce the number of daemons we had to use.

Celery ships with a couple of daemon configurations out of the box, support for init.d style init scripts, and support for supervisord. We first looked at the init.d approach, but there didn't seem to be a good way to have it run multiple settings files without creating multiple scripts, which seemed hacky. So we went with superisord for the task. Below is our configuration, if you are curious.

/etc/supervisord.conf By default, the conf file is in the top-level /etc/ directory. We kept it this way, but I kind of wish it was in it's own directory. This is basically the exact script that celery ships with

```
unix_http_server]
file=/tmp/supervisor.sock ; path to your socket file

[supervisord]
logfile=/var/log/supervisord/supervisord.log ; supervisord log file
logfile_maxbytes=50MB ; maximum size of logfile before rotation
logfile_backups=10 ; number of backed up logfiles
loglevel=info ; info, debug, warn, trace
```

```
pidfile=/var/run/supervisord.pid ; pidfile location
nodaemon=false ; run supervisord as a daemon
minfds=1024 ; number of startup file descriptors
minprocs=200 ; number of process descriptors
user=root ; default user
childlogdir=/var/log/supervisord/ ; where child log files will live


[rpcinterface:supervisor]
supervisor.rpcinterface_factory = supervisor.rpcinterface:make_main_rpcinterface ;

[supervisorctl]
serverurl=unix:///tmp/supervisor.sock ; use unix:// schem for a unix sockets.

[include]
files = supervisord/celeryd.conf
```

Then we created a supervisord directory which we included in the above file (in the last line) that contains the celery specific configuration. On this machine the only thing that supervisord is watching is celery, so that has kept our configuration simple.

/etc/supervisord/celeryd.conf Inside of our celeryd specific configuration we went with mostly stock options except how we are setting up the DJANGO_SETTINGS_MODULE. We need to change the environment in which we are deploying, so that the celery daemon runs against the correct database.

```
[program:celery-cms]
environment = PYTHONPATH='/home/code',DJANGO_SETTINGS_MODULE='ljworld.standard'
command=/home/code/django/bin/django-admin.py celeryd --loglevel DEBUG -c2
user=nobody
numprocs=1
stdout_logfile=/var/log/celery/cms_supervisord.log
stderr_logfile=/var/log/celery/cms_supervisord.err
autostart=true
autorestart=true
startsecs=10

[program:celery-weeklies]
environment = PYTHONPATH='/home/code',DJANGO_SETTINGS_MODULE='desotoexplorer.settings'
command=/home/code/django/bin/django-admin.py celeryd --loglevel DEBUG -c2
user=nobody
numprocs=1
stdout_logfile=/var/log/celery/weeklies_supervisord.log
stderr_logfile=/var/log/celery/weeklies_supervisord.err
autostart=true
autorestart=true
startsecs=10
```

The really nice part about using supervisord is that our fabric script for deploying changes to celery is just deploying the code and then running `supervisorctl restart celery-cms`.

I hope today's post was useful, and I'm again curious for any other awesome celery tips!

Django Testing Mailing List

I have a couple of testing related applications in the Django community. I don't have a good way of communicating with the users of these apps, namely about releases, or helping with support questions. So I am starting the Django

Testing Mailing List for people that are interested in my testing projects: django-test-utils, django-crawler, and django-kong.

This is kind of an experiment of a combined mailing list for a couple of different projects. Hopefully people that are interested in one will be interested in the others, so we'll see how it goes.

If you have a testing related app for Django that doesn't have a good place to discuss, feel free to make it your home as well.

Note: This is more a place to discuss issues my django applications. If you are interested in testing, the Testing in Python list and the #python-testing IRC channel on Freenode are both great resources as well.

Running Hudson matrix builds on multiple machines

When I was setting up the Django Hudson instance, I ran into a problem that seems like it should be pretty easy to solve. However, I couldn't figure out a way. So at this point it's looking like we're going to have to use buildbot to build out what we want instead of Hudson. Wondering if I missed something obvious, or if this is a missing feature.

Our setup

We have our builds segmented currently by Django version and database. So to get something up and going, we have a Django trunk and 1.2.X build going against sqlite. We use a matrix build to run this against python 2.4-2.6, which means 3 builds in total for each Django version.

However, **I can't find a way to make the matrix build choose the slave to run on based on what version of python it supports**. I want to be able to randomly add slaves to the Hudson configuration, and have them pick up builds based off of their capabilities.

The problem

Hudson supports the idea of tagging, so we came up with a tagging scheme for the slaves. So it seems that we should be able to tell a test to run on any slave tagged with the versions of python we want. Hudson also supports this, but it runs all the tests across all the different slaves each time. I need it to **have a pool of workers capable of running a set of tests, but run each test on only one of the members of the pool**.

I was wondering if we're doing it wrong, or if other people know the correct way to run tests across a set of slaves, based on the capabilities of the slave? This seems like a pretty basic requirement for people running any kind of sizable Hudson configuration.

Using ZNC, an IRC bouncer

I use IRC for work, play, and generic open source questions and support. I think it's a pretty integral part of my existence as a developer. Today I'm going to write about why using an IRC bouncer makes IRC a ton better and show you how to get one setup.

ZNC

ZNC is the bouncer that I was recommended when I started, and I only have good things to say about it. On ubuntu installing ZNC is really simple.

```
sudo apt-get install znc-extra
```

This will pull down ZNC and it's extra modules which offer you some nice features. So now that you have ZNC installed, you need to make a configuration. You can do this with the `znc --makeconf` command, which I've pasted a session of at the bottom of this post. However, here is the config that I run now.

```
Listen       = +6666
ConnectDelay = 30

<User eric>
        Pass = md5#blah
        Nick = ericholscher
        AltNick = ericholscher_
        Ident = eric
        RealName = Eric
        QuitMsg = Peace.
        StatusPrefix = *
        ChanModes = +stn
        Buffer = 5000
        KeepBuffer = false
        MultiClients = true
        BounceDCCs = true
        DenyLoadMod = false
        Admin = true
        DenySetVHost = false
        DCCLookupMethod = default
        TimestampFormat = [%H:%M:%S]
        AppendTimestamp = false
        PrependTimestamp = true
        TimezoneOffset = 0.00
        JoinTries = 0
        MaxJoins = 5

        LoadModule = chansaver
        LoadModule = log

        Server = irc.freenode.net +7000

        <Chan #django>
        </Chan>
</User>
```

Note: This might be from a slightly older version of ZNC, so you might have to modify this to work on the newest version of ZNC in the ubuntu 10.10 repos. Using the –makeconf option with the same answers will also make an up to date conf.

Important options

The `Buffer` setting is the number of lines to keep in the buffer when you're not connected. If you have your ZNC not keep your buffer, with `KeepBuffer = false` setting, I tend to keep the Buffer setting pretty high. You don't want to miss messages when you're disconnected, so it will just stream these messages back to you when you reconnect.

The other use of ZNC that I know people have is to connect from multiple clients, like checking IRC on an iPhone. So long as you have the `MultiClients` setting set, you can do this and it is transparent to anyone else in your channels. However, when you connect, you always want a scrollback of the context of the last things said. In this case, you'd want to have `KeepBuffer = true`, but have a small `Buffer` playback, so you don't spam your phone.

The `+6666` on the Listen portion of the config means that it is listening on port 6666, using SSL. So make sure if you turn on SSL, your client is connecting with SSL as well.

Good Modules

The modules that I have enabled are the chansaver and log modules. The chansaver module writes out your ZNC config every time you join or part a channel, so when you restart your bouncer it will have the rooms you're in. The log module logs all the channels your in, which is nice because it allows you to have comprehensive logs of your work channels.

ZNC also comes with a webadmin that I have never used, but hear is quite nice. It allows you to configure everything through a web interface. ZNC has a lot more power than I've shown here, including connecting to multiple backend servers, having multiple users, inter-user chat, and lots of other interesting things. Once you've gotten hooked, you can share your server with your friends, and play around with modules.

Using it

So now to connect to your proxy, in your IRC client, instead of connecting to your normal server (eg. irc.freenode.net 6667), you would connect to your bouncer instead. This will be running on the IP and port that you have configured in your ZNC config. My Limechat config looks like this:

Figure 2.4: Limechat Config

You should then be able to just connect to that server, and your client will show all the channels you're connected to. You can try logging off and back on a couple of minutes later and see that it plays back what you've missed.

This really changes how you interact with IRC I find, because you can keep tabs on everything that is going on when you aren't connected. I can be in the middle of a conversation, disconnect and move to a meeting room downstairs, and pick right back up where I've left off.

znc –makeconf session Here is a copy of the makeconf session I talked about earlier. Where there is no visual input, it's just me accepting the defaults.

```
eric@Chimera:~$ znc --makeconf
[ ** ] Building new config
[ ** ]
[ ** ] First lets start with some global settings...
[ ** ]
[ ?? ] What port would you like ZNC to listen on? (1 to 65535): 6666
[ ?? ] Would you like ZNC to listen using SSL? (yes/no) [no]: yes
[ ** ] Unable to locate pem file: [/home/eric/.znc/znc.pem]
[ ?? ] Would you like to create a new pem file now? (yes/no) [yes]: yes
[ ?? ] hostname of your shell (including the '.com' portion): irc.ericholscher.com
[ ok ] Writing Pem file [/home/eric/.znc/znc.pem]...
[ ?? ] Would you like ZNC to listen using ipv6? (yes/no) [no]:
[ ?? ] Listen Host (Blank for all ips):
[ ** ]
[ ** ] -- Global Modules --
[ ** ]
[ ?? ] Do you want to load any global modules? (yes/no): yes
[ ** ] +-----------+-------------------------------------------------------+
[ ** ] | Name      | Description                                           |
[ ** ] +-----------+-------------------------------------------------------+
[ ** ] | partyline | Internal channels and queries for users connected to znc |
[ ** ] | webadmin  | Web based administration module                       |
[ ** ] +-----------+-------------------------------------------------------+
[ ** ] And 10 other (uncommon) modules. You can enable those later.
```

```
[ ** ]
[ ?? ] Load global module <partyline>? (yes/no) [no]: no
[ ?? ] Load global module <webadmin>? (yes/no) [no]: yes
[ ** ]
[ ** ] Now we need to setup a user...
[ ** ]
[ ?? ] Username (AlphaNumeric): eric
[ ?? ] Enter Password:
[ ?? ] Confirm Password:
[ ?? ] Would you like this user to be an admin? (yes/no) [yes]:
[ ?? ] Nick [eric]:
[ ?? ] Alt Nick [eric_]:
[ ?? ] Ident [eric]:
[ ?? ] Real Name [Got ZNC?]:
[ ?? ] VHost (optional):
[ ?? ] Number of lines to buffer per channel [50]: 500
[ ?? ] Would you like to keep buffers after replay? (yes/no) [no]:
[ ?? ] Default channel modes [+stn]:
[ ** ]
[ ** ] -- User Modules --
[ ** ]
[ ?? ] Do you want to automatically load any user modules for this user? (yes/no): yes
[ ** ] +-------------+-------------------------------------------------------------------------+
[ ** ] | Name        | Description                                                             |
[ ** ] +-------------+-------------------------------------------------------------------------+
[ ** ] | admin       | Dynamic configuration of users/settings through irc                     |
[ ** ] | chansaver   | Keep config up-to-date when user joins/parts                            |
[ ** ] | keepnick    | Keep trying for your primary nick                                       |
[ ** ] | kickrejoin  | Autorejoin on kick                                                      |
[ ** ] | nickserv    | Auths you with NickServ                                                 |
[ ** ] | perform     | Keeps a list of commands to be executed when ZNC connects to IRC.       |
[ ** ] | simple_away | Auto away when last client disconnects                                  |
[ ** ] +-------------+-------------------------------------------------------------------------+
[ ** ] And 33 other (uncommon) modules. You can enable those later.
[ ** ]
[ ?? ] Load module <admin>? (yes/no) [no]: yes
[ ?? ] Load module <chansaver>? (yes/no) [no]: yes
[ ?? ] Load module <keepnick>? (yes/no) [no]: yes
[ ?? ] Load module <kickrejoin>? (yes/no) [no]:
[ ?? ] Load module <nickserv>? (yes/no) [no]:
[ ?? ] Load module <perform>? (yes/no) [no]:
[ ?? ] Load module <simple_away>? (yes/no) [no]: yes
[ ** ]
[ ** ] -- IRC Servers --
[ ** ]
[ ?? ] IRC server (host only): irc.freenode.net
[ ?? ] [irc.freenode.net] Port (1 to 65535) [6667]:
[ ?? ] [irc.freenode.net] Password (probably empty):
[ ?? ] Does this server use SSL? (probably no) (yes/no) [no]:
[ ** ]
[ ?? ] Would you like to add another server? (yes/no) [no]:
[ ** ]
[ ** ] -- Channels --
[ ** ]
[ ?? ] Would you like to add a channel for ZNC to automatically join? (yes/no) [yes]: yes
[ ?? ] Channel name: #django
[ ?? ] Would you like to add another channel? (yes/no) [no]:
[ ** ]
```

```
[ ?? ] Would you like to setup another user? (yes/no) [no]:
[ ok ] Writing config [/home/eric/.znc/configs/znc.conf]...
[ ** ]
[ ** ] To connect to this znc you need to connect to it as your irc server
[ ** ] using the port that you supplied.  You have to supply your login info
[ ** ] as the irc server password like so... user:pass.
[ ** ]
[ ** ] Try something like this in your IRC client...
[ ** ] /server <znc_server_ip> 6666 eric:<pass>
[ ** ]
[ ?? ] Launch znc now? (yes/no) [yes]:
[ ok ] Opening Config [/home/eric/.znc/configs/znc.conf]...
[ ok ] Binding to port [+6666] using ipv4...
[ ** ] Loading user [eric]
[ ok ] Loading Module [admin]... [/usr/lib/znc/admin.so]
[ ok ] Loading Module [chansaver]... [/usr/lib/znc/chansaver.so]
[ ok ] Loading Module [keepnick]... [/usr/lib/znc/keepnick.so]
[ ok ] Loading Module [simple_away]... [/usr/lib/znc/simple_away.so]
[ ok ] Adding Server [irc.freenode.net 6667]...
[ ok ] Loading Global Module [webadmin]... [/usr/lib/znc/webadmin.so]
[ ok ] Forking into the background... [pid: 15983]
[ ** ] ZNC 0.092+deb3 - http://znc.sourceforge.net
```

Building a Django App Server with Chef: Part 1

Alternate title: Fucking Chef, How does it work?

When I started looking at Chef, I found it incredibly complex and lacking in fundamental documentation. This is going to be my experience understanding Chef while setting up a single server. This strategy can be used across multiple servers, with a little tweaking.

I'd like to thank Jacob for the ideas and some of the inspiration behind the code and ideas. The code for this blog post can be found on github. It will be expanded as I write updates.

Today's code will be using tag blog-post-1 in the repo.

Goal

I'm hoping to write a blog series that goes from explaining what Chef is, to having a working Django server, and to release all this code so that you can tweak and use it with your own servers. I'll be doing this to set up a new and configuration managed server for ReadTheDocs.

There are a couple of other good chef intros available. However, they only get you to the super basic first step of setting up the server. Hopefully this series will be more in depth and useful to actually getting a real server running under Chef.

Basic terminology

Chef has some basic terminology that you need to understand before we get into things. I'm going to purposefully leave out a ton of stuff, because it isn't really important for me now.

- Cookbook (cookbooks/*)

A cookbook, not surprisingly, contains recipes. With my configuration, we're only going to use one cookbook, that has multiple recipes. This is probably wrong and horrible, but it's simple, which is the goal for now.

- Recipe (cookbooks/*/recipes/*.rb)

A recipe is the basic building block of Chef. It is what does the meat of the work that you want done.

- Resources

A resources is an abstraction that defines a specific piece of your configuration. It can be a file, a user, or just about anything you want to talk about on your system.

- Attributes (node.json)

Attributes are just a blob of JSON that Chef uses to pass around data. It will be (slightly) different for each server that you want to set up. I really like this approach, because I think a data-driven approach is the correct way to solve this problem.

Bootstrapping chef

We'll be running what is called chef solo. This means that chef will be run independently of a server, and just execute on our one host. This seems to be the easiest way to get things running.

When you first run `chef-solo`, it will look for a `solo.rb` file. The documentation about configuring chef-solo does a decent job of explaining this. By default it looks in /etc/chef/solo.rb, so lets go ahead and use that. It has 2 required fields, a `cookbook_path` and `json_attribs`. `cookbook_path` tells chef where to find the "cookbooks" it uses to run your code, and `json_attribs` tells it where to load in your data dictionary.

Note: The docs say that `file_cache_path` is required, but it just defaults to `/var/chef/cache`.

For simplicity, we're going to keep our cookbooks and json data file in `/etc/chef`. On my local machine I keep everything in a `~/projects/chef` directory, and then sync that to the remote box.

In production you'll probably want to set up your remote server to pull from a repository somewhere, so that you can have a stable deployment base, but again, syncing from the local filesystem is simple and works.

Bootstrapping your new server

I'll be using a fabric script to run the commands on a remote server, which also allows me to run them again later on a new machine in an automated fashion. So this is the first simple script that we'll use to bootstrap our new server, `fabfile.py`:

```
from fabric.api import env, local, sudo
env.user = 'root'
env.hosts = ['204.232.205.196']

env.chef_executable = '/var/lib/gems/1.8/bin/chef-solo'

def install_chef():
    sudo('apt-get update', pty=True)
    sudo('apt-get install -y git-core rubygems ruby ruby-dev', pty=True)
    sudo('gem install chef', pty=True)

def sync_config():
    local('rsync -av . %s@%s:/etc/chef' % (env.user, env.hosts[0]))

def update():
    sync_config()
    sudo('cd /etc/chef && %s' % env.chef_executable, pty=True)
```

A couple notes: the chef executable is version-dependent, but that's because I don't know enough ruby to query it dynamically. You will also need to change the value of the env.hosts to a server that you actually control.

This will install the ruby dependencies, and get chef up and running for you. You'll need to install fabric (`pip install fabric`, presumably in a virtualenv), and then run `fab install_chef` to get it up and running. Then you can go ahead and run `fab sync_config` to get your chef configuration onto the remote server.

Now we need to go ahead and figure out how to make chef actually do something. You'll see in the `cookbooks/main/recipes/default.rb` file we have a simple `package` declaration. This simply means to make sure that the package is installed on the remote system. This is as simple as it gets, so lets go ahead and run it. With the `fab update` command, it will sync your local directory, and run chef on the remote server.

```
-> fab update
[localhost] run: rsync -av . root@204.232.205.196:/etc/chef
[204.232.205.196] sudo: cd /etc/chef && /var/lib/gems/1.8/gems/chef-0.9.12/bin/chef-solo
[204.232.205.196] err: stdin: is not a tty
[204.232.205.196] out: [Tue, 09 Nov 2010 01:42:01 +0000] INFO: Setting the run_list to ["main::defaul
[204.232.205.196] out: [Tue, 09 Nov 2010 01:42:01 +0000] INFO: Starting Chef Run (Version 0.9.12)
[204.232.205.196] out: [Tue, 09 Nov 2010 01:42:01 +0000] INFO: Installing package[curl] version 7.21.
[204.232.205.196] out: [Tue, 09 Nov 2010 01:42:04 +0000] INFO: Chef Run complete in 2.574963 seconds
[204.232.205.196] out: [Tue, 09 Nov 2010 01:42:04 +0000] INFO: cleaning the checksum cache
[204.232.205.196] out: [Tue, 09 Nov 2010 01:42:04 +0000] INFO: Running report handlers
[204.232.205.196] out: [Tue, 09 Nov 2010 01:42:04 +0000] INFO: Report handlers complete
```

You now have Chef running on your server. That was pretty easy, eh? For tomorrow's lesson, we'll be making it actually do something, like installed nginx and gunicorn, and keeping track of config files.

Building a Django App Server with Chef: Part 2

Alternate title: Actually doing something useful.

Yesterday we covered the basics to getting started with Chef. You should have a remote server configured with chef, and have curl installed! Now lets go ahead and get some useful bits for your Django application.

What we'll need

So this is going to be based around the way that I set up my servers, so if this is different than you, I'm sorry. However, I think it is a pretty solid way of managing them. A lot of the ideas here are stolen from Travis when he set up the server for Pypants.

So lets assemble a list of things we're going to want in order to get a super basic Django configuration running:

- A user to run our code as and who's home directory we'll store the data.
- A basic global python ecosystem, including setuptools and pip
- A virtualenv to store all the project-specific packages and code in
- A copy of the project that we'll be running

Let's get started.

The finished code for today is located on github, with the tag blog-post-2. It is a copy of the completed steps, so feel free to peek through that and come back here for clarification (or to ask questions).

Setting up our user

For RTD, I run everything under the user docs. So we'll go ahead and set up that user so that we can get our site set up. We're going to go ahead and replace our "default" recipe, because right now it isn't doing anything much useful. The relevant part is below:

cookbooks/main/recipes/default.rb

```
node[:base_packages].each do |pkg|
    package pkg do
        :upgrade
    end
end

node[:users].each_pair do |username, info|
    group username do
       gid info[:id]
    end

    user username do
        comment info[:full_name]
        uid info[:id]
        gid info[:id]
        shell info[:disabled] ? "/sbin/nologin" : "/bin/bash"
        supports :manage_home => true
        home "/home/#{username}"
    end

    directory "/home/#{username}/.ssh" do
        owner username
        group username
        mode 0700
    end

    file "/home/#{username}/.ssh/authorized_keys" do
        owner username
        group username
        mode 0600
        content info[:key]
    end
end

node[:groups].each_pair do |name, info|
    group name do
        gid info[:gid]
        members info[:members]
    end
end
```

There's a lot of stuff going on here, so lets go over it. First you might notice that there's this node variable, the node data structure is the JSON that you have in your node.json file. It is looping over the keys and values with ruby's each_pair and pair functions.

The base_packages bit is a cool example of the power of the chef configuration. We have a list of packages that we want to install in our Attributes, and we're looping over them and setting using the package Resource.

I realize I skipped over the run_list part yesterday, but it basically is just a list of recipes to run. Each of the resources in the default.rb file should be pretty self explanatory. The Chef Resource Documentation is really comprehensive, and will probably be the most referenced document that you use. The main resource's that we used were **group, user, file, directory**, let's take a look at the User declaration in particular.

Everything there should be pretty obvious, as it's the information that goes into /etc/passwd for the user. However, the `supports` keyword isn't obvious at first. This is part of the Common Attributes that can be set on all Resources. It's a way of passing along configuration options to the Resource. manage_home actually just makes it so that the users home directory is created when the user is created.

So we're going to have to go ahead and put some data in there for it to work with. Our node.json will now look like this:

node.json

```
{
  "run_list": [ "main::default", "main::python", "main::readthedocs" ],
  "base_packages": ["git-core", "bash-completion"],

  "users": {
    "docs": {
      "id": 1001,
      "full_name": "Docs User",
      "key": "ssh-rsa key-goes-here eric@Bahamut"
    }
  },

  "groups": {
    "docs": {
      "gid": 201,
      "members": ["docs"]
    }
  }
}
```

Adding a Basic Python Environment

Now lets go ahead and add a python recipe to build out some basic python stuff that we'll be needing.

cookbooks/main/recipes/python.rb

```
node[:ubuntu_python_packages].each do |pkg|
    package pkg do
        :upgrade
    end
end

# System-wide packages installed by pip.
# Careful here: most Python stuff should be in a virtualenv.

node[:pip_python_packages].each_pair do |pkg, version|
    execute "install-#{pkg}" do
        command "pip install #{pkg}==#{version}"
        not_if "[ `pip freeze | grep #{pkg} | cut -d'=' -f3` = '#{version}' ]"
    end
end
```

Additions to node.json

```
"ubuntu_python_packages": ["python-setuptools", "python-pip", "python-dev", "libpq-dev"],
"pip_python_packages": {"virtualenv": "1.5.1", "mercurial": "1.7"},
```

Here we're adding some global packages that we need. We're going to install setuptools and pip so that we can install further python packages. python-dev and libpq-dev are so that we have the headers for libraries that need to compile against postgres and python. We'll also be installing virtualenv and mercurial globally so that we can create our virtualenv and install packages from mercurial.

Creating a virtualenv

We're going to introduce the first new Chef concept here, which is called a Definition.

- Definition (cookbooks/*/definitions/*.rb)

  A definition is a custom Resource that you build to abstract a set of operations. Pretty simple

This is a definition that Jacob published and then I updated to make the permissions correct. It allows you to set up a virtualenv:

`cookbooks/main/definitions/virtualenv.rb`

```
define :virtualenv, :action => :create, :owner => "root", :group => "root", :mode => 0755, :packages
    path = params[:path] ? params[:path] : params[:name]
    if params[:action] == :create
        # Manage the directory.
        directory path do
            owner params[:owner]
            group params[:group]
            mode params[:mode]
        end
        execute "create-virtualenv-#{path}" do
            user params[:owner]
            group params[:group]
            command "virtualenv #{path}"
            not_if "test -f #{path}/bin/python"
        end
        params[:packages].each_pair do |package, version|
            pip = "#{path}/bin/pip"
            execute "install-#{package}-#{path}" do
                user params[:owner]
                group params[:group]
                command "#{pip} install #{package}==#{version}"
                not_if "[ `#{pip} freeze | grep #{package} | cut -d'=' -f3` = '#{version}' ]"
            end
        end
    elsif params[:action] == :delete
        directory path do
            action :delete
            recursive true
        end
    end
end
```

As you can see, it takes a bunch of arguments, then just wraps up a bunch of Resource definitions in a nice little package. There is a little bit of magic with the pip freezing things, but it's basically just how we're checking to make sure that a package isn't instead before we install it. We are using only using the **directory and execute** Resources here.

Now we're going to use this virtualenv Definition, and create the home virtualenv for our site. I like to keep my virtualenv's in `~/sites/<domain>`, so this will go into `/home/docs/sites/readthedocs.org/`. Since this is becoming specific to the site we're building, it's going to go into a readthedocs recipe:

`cookbooks/main/recipes/readthedocs.rb`

```
directory "/home/docs/sites/" do
    owner "docs"
    group "docs"
    mode 0775
```

```
end

virtualenv "/home/docs/sites/readthedocs.org" do
    owner "docs"
    group "docs"
    mode 0775
end
```

This will set up a basic virtualenv in our directory.

Getting our site set up

To get our site set up, we need to pull in the source code, and make sure our virtualenv has all the requirements. This code is a little bit hacky, and could probably be abstracted out a bit, but it will work for now. We're going to go ahead and add some things to our readthedocs Recipe.

Additions to `cookbooks/main/recipes/readthedocs.rb`

```
directory "/home/docs/sites/readthedocs.org/run" do
    owner "docs"
    group "docs"
    mode 0775
end

git "/home/docs/sites/readthedocs.org/checkouts/readthedocs.org" do
  repository "git://github.com/rtfd/readthedocs.org.git"
  reference "HEAD"
  user "docs"
  group "docs"
  action :sync
end

script "Install Requirements" do
  interpreter "bash"
  user "docs"
  group "docs"
  code <<-EOH
  /home/docs/sites/readthedocs.org/bin/pip install -r /home/docs/sites/readthedocs.org/checkouts/read
rg/deploy_requirements.txt
  EOH
end
```

I like to have my runtime files in the `venv/run` directory, so we'll go ahead and create that directory. Then comes the fun part.

We are checking the Readthedocs source out of github with the git Resource. Chef only supports git and svn as far as I can tell, so luckily I'm using git.

Then we're going to install from the pip requirements file. This is using the script Resource, which allows you to inline a bash, ruby, python, or more script inside your Recipe. This is using a hard coded bash script to install the requirements, which sucks, but will work for now.

Note: Chef appears to buffer output and not show itself as doing anything when running the script Resource here, so it will look like your build will hang while it installs your pip requirements file for the first time.

Done for now

Alright, this post has gotten long enough, so we're done for today. But we're in a pretty awesome spot, I think. We now have our app server set up with a runnable version of our code. You can go ssh in and play around, you should be able to run simple manage.py commands inside the virtualenv and whatnot (after a syncdb).

Tomorrow we'll talk about deploying our code with Nginx and Gunicorn. I've been having trouble with Upstart, so we might switch our deployment to Supervisord, but we'll see how it goes.

Don't forget to check out the finished code on Github to see the actual running examples.

2.3.5 2009

Django now has fast tests

As of Changeset 9756, Django's test suite is A HELL OF A LOT FASTER. This was one of the 1.1 Features and probably the one I was looking forward to the most. Django Unit Tests now run inside of a transaction.

Karen Tracey did most of the work on this finishing one, and we owe her a huge thanks, and a lot of our time :). Her commit message sums up the work better than I can.

```
Fixed #8138 -- Changed django.test.TestCase to rollback tests (when
the database supports it) instead of flushing and reloading the database.
This can substantially reduce the time it takes to run large test suites.

This change may be slightly backwards incompatible,
if existing tests need to test transactional behavior,
or if they rely on invalid assumptions or a specific test case ordering.
For the first case, django.test.TransactionTestCase should be used.
TransactionTestCase is also a quick fix to get around test case errors
revealed by the new rollback approach, but a better long-term fix
is to correct the test case. See the testing doc for full details.

Many thanks to:
* Marc Remolt for the initial proposal and implementation.
* Luke Plant for initial testing and improving the implementation.
* Ramiro Morales for feedback and help with tracking down a
  mysterious PostgreSQL issue.
* Eric Holscher for feedback regarding the effect of the
  change on the Ellington testsuite.
* Russell Keith-Magee for guidance and feedback from beginning to end.
```

The amazing thing is that you don't have to do anything in order to get the benefit from this change. People's test suites will now be running about 8-12x (ed: was 40x, but that was a bit much) faster than before. Depending on whether you have a larger portion of doctests or unit tests, you will get different speedups. The Database backend you are using is also import; MySQL/MyISAM doesn't support transactions, and tests running in SQLite were already much faster, so they don't get as much of a percentage gain.

Ellington's test suite, which was taking around 1.5-2 hours to run on Postgres, has been reduced to 10 minutes. I tested changing some of our more expensive doctests to unit tests (and thus getting transaction support), and it looks like we can get our suite to run in 3-4 MINUTES.

Prior to this change, before every doctest or unittest test case was run, Django would do an entire flush and syncdb of the database! This was time consuming, and not totally necessary. Now all unit tests are wrapped in a transaction. This is a much faster operation on the DB, and that is where the speed gains come from.

People using doc tests aren't having their tests run inside a transaction, but that is the same behavior as before the change. If you are curious more about the implementation of this patch, there is a huge discussion on ticket 8138 that

shows some of the interesting things encountered during this process.

This patch applies cleanly to Django 1.0 as far as I know, so if you are running tests on any kind of large code base I would recommend either applying the patch in #8138, or upgrading to trunk. This also means that you have one less excuse for not writing or running tests ;)

Review of Pro Django by Marty Alchin (1/2)

5 second review: **Reading this book will make you a much better Django Programmer.**

In full disclosure, I was sent a review copy of this book by the publisher, but I had already pre-ordered it on Amazon, and the copy I am reviewing is that copy. The review copy is now the office copy :)

I will do an overview, and then a specific breakdown of what I thought after reading each chapter. This is currently only the first half of the book, but the book is already worth it's price (perhaps in gold). This covers chapters 1-6, which is basically the normal URLs/Models/Views/Templates stuff. The later sections appear to cover more specialized and advanced use cases. You can download the table of contents also.

Also note, that you can see my further reviews and some other peoples over on Readernaut

Overview

Pro Django is certainly not for the beginner. It assumes a pretty decent amount of Django experience (or at least the ability to learn/reference it easily). I think that this is a good thing, because it doesn't get bogged down in silly details and explanations when what we want is upper level content. If you have been doing Django stuff for a couple months, I think this book is amazingly good at pushing you to the next level of knowledge in the field.

I certainly recommend reading James Bennett's Practical Django Projects (once version 2 gets released) before this one, as it is an easier introduction. These two books together provide an amazing 1-2 knockup punch to getting you to be a great Django developer. Now onto the review.

Chapter 1: Understanding Django

This chapter goes over the basics of Django and the philosophy that it has. It is a great introduction into how to be a good member of the Django community. It is available for free on the Apress website, and I recommend reading it, because I felt inspired by the prologue, and I think that Marty explains the value of the community around Django very well.

Links from the books pages are provided as links from prodjango.com, which is nice at first, but the obfuscation of the URLs is a pain later in the book. It is used as a way to hide information ("The django irc channel", with a link) instead of #django in irc.freenode.net. Also, I have read some of the URLs linked to, but knowing where they go helps me decide if i need to read them.

Chapter 2: Django is Python

With one of the best descriptions that I have read of metaclassing in python, this chapter starts off guns a-blazing. It tries to show some of the more advanced Python features that Django uses, and stresses that Django is merely Python. I loved the explanation of the gentlemen's agreement on interfaces, including file-like objects. The explanation of metaclassing is finished up with a great example of implementing a plugin system for a password validator. The code ends up being around 40-50 lines, flowing through my brain like a river, and beautiful. It was also explained in a way that allowed me to see the abstract value of the ideas, and not simply how it is useful in this specific implementation.

It also includes a really heady implementation of decorators that blew my mind a little bit. With my current intermediate level of python knowledge, it provided a reference for things that I mostly knew, and went into depth in the parts that were in need of it. A great start to the book. As of page 45, my brain is already going full throttle and I'm looking forward to what this book has in store.

Chapter 3: Models

I found this chapter to be a little less amazing than the previous, but still solid. It didn't quite flow as well for me, and I had to think about things a lot more. I didn't have a great understanding of how all of the subject matter fit together. This might just be the fact that I'm not as acquainted with the topics.

It has a cool example of creating a field that stores pickled data and returns it as a normal object. As the chapter progressed the value of the explanations earlier in the chapter start to gel a bit, but not totally. I think that this is one that re-reading, at least for me, will be really valuable after I'm finished. It provided a great reference to all of the different fields, and why we should care about them. I really like the continuing theme of "this is a brief explanation of what X is, and this is an in depth explanation of why you should care"

Chapter 4: Urls and Views

This chapter does a good job of showing how URLs and Views are inextricably linked. Your URLConf basically just hands off data to views, and it goes over all the interesting things that you can do in this space. It flowed well, was just the right length, and left me really appreciating the value of decorators around views.

The concept of using a decorator on the view function in the URLConf, instead of around the view definition, is novel. The ability to manipulate the arguments to the view in the URLConf file, returning objects instead of strings to the views is very powerful. Using decorators to abstract boilerplate is a really powerful pattern.

The explanation of views and making them more generic was good, but I already knew it from Practical Django Projects. This obviously lead into generic views. He also discussed using a Class as a view (and the downside of trying to reverse() it), but it was lacking a discussion of Class-based Generic Views which I thought was coming. This is going to be implemented by 1.2, so it might have been a good time to at least mention it.

Chapter 5: Forms

This chapter is not as interesting to me, because I don't plan to be doing a lot of work on front end forms. However, there is still a great explanation of the way that forms work on the backend (almost exactly like models). The harsh warning about verifying user input is good, because user input is indeed evil.

In example at the end of the chapter, Marty goes through how to create a form that can be saved and restored, no matter what the contents. It ends up being implemented in a decorator, which blew my mind. All you have is a view that handles a valid form, and a decorator gives you the ability to save and resume that same form, simply by posting an md5hash to it. Pretty Crazy. It also goes into Widgets and how to create your own to display forms. There is a bit of a lack of how to include Javascript and CSS inline in your forms, but talks about how to embed custom attributes. Also, there is no mention of creating forms from Models! I feel that this is probably one of the most common operations I do with forms. This might have been excluded because it is already well known and un-interesting.

Chapter 6: Templates

This chapter explains one of the simpler parts of Django. It is pretty neat that you can understand pretty much how the entire template system works in a chapter of about 20 pages. I knew a decent bit about it before, but this chapter certainly made my knowledge more concrete.

There is a really neat example at the end of the chapter about writing themes for a website. It goes through a really in depth usage of the Template system, including introspecting nodes. I've needed to do something like this for my own stuff, and this example is invaluable. It also talks about how to easily create tags and filters by yourself.

At this point I'm at page 163 of around 300. There is still a ton of great knowledge in this book, and I'm excited to read the rest of it.

Encouraging Web Interaction for University Students

So today I would like to tell a story that really shows why the internet is an amazing thing. This month's articles on A List Apart focused on web education for universities and I'd like to share a story about one way to empower students and show them the power of the internet. Teaching people to create for the internet is a great goal, but teaching people the power of the internet by example is something amazing as well. It is hard to motivate people to create things on the internet without the understanding of how that has value.

So this all started (like many things these days) on Twitter. I got a tweet from a professor at my old school, Jim Groom. It linked to an article from a student in a Graphic Novels class, who had a blog post about garfield. His post was about garfield dying, and had some really great analysis of the content. It was a genuinely interesting piece, and something that I thought that the rest of the world should see.

In a normal context, this work simply would have been given to a teacher, and I would never have seen it. Luckily I know people that are still at the school, who happened to link to it, because it happened to be online. The university actually has a thriving blogging system they are pioneering (which is linked from the front page!). Students think that these blogs they are publishing are silly, and there is little reason for it. Once things are online, they are searchable, and others can seek out the valuable content created by students.

This story doesn't end there though. I submitted the blog entry by the student to Hacker News and Reddit. I figured that people like me read these sites, and would appreciate it as well. I was correct. The article stayed on the front page of Hacker News for over a day, and gathered 51 votes (which is very high) and got 15 up votes on reddit.

This lead to a surge of traffic to the post! This information was being spread and a student was quickly learning the value of publishing things online. In a follow up post, Jim says that the post was viewed over 7,000 times in 3 days! Imagine being the student when he is told that something he wrote for a class was viewed by more people than the entire student body of the school you attend! How Impowering!

If more people had experiences like this, I think that a lot of people would find more value in learning how to create things on line. I can't imagine that the student isn't now extremely curious about how all this happened, and how he can make it happen again. It breeds curiosity and excitement over the internet, instead of wonder and fear. These are the kind of experiences that we should be fostering.

I agree that you should go volunteer and talk to students in classes that are already interested in the subject. However, it is much more important to show people the value of spreading information. Show the power of the internet through example (this one or others), and foster creativity and the ideals of public information. Things on the internet are forever, and that is scary if it's a picture of you drunk; but if it is knowledge that you created, then that is priceless.

Django Conventions Project Update

So about a month ago I started a project on my blog called the Django Conventions Project. It was an attempt to document and record conventions that are used across the community. Conventions are a great thing, with Python and Django relying on them a great deal. Things like private methods being underscored aren't enforced on a language level, but are more of a gentleman's agreement.

I think that conventions can indeed have a lot of value, but they are hard to discover without practice. I think that embodying this knowledge in documentation can be extremely valuable. It proves useful for people that are just starting to kind of establish themselves in a code base. It is also useful for more advanced people as a reference and to make sure they are following them. I know that I learned about a few new ones when I started the project.

I got around 20 comments, and people seemed really energized when I posted last time, so I think people are genuinely interested. In hindsight, I should have created a source repo with Sphinx at the beginning and started accepting patches. Brian Rosner is involved in Pinax, which has these conventions and standards as a stated goal as well. He created a django-reusable-apps-docs github project for these to live. So I went ahead and ported my HTML docs over to my fork of that project on Github.

Please feel free to branch the repo and submit patches/pull requests back to me. Also, feel free to join the django-hotclub mailing list which was created for discussion about reusable apps. The #django-hotclub or #pinax channel on Freenode is also a good place to find us and talk about reusable apps in real time.

Brian has a mirror of his repo updating every 10 minutes to http://appdocs.oebfare.com/.. I have a mirror of my github repo up on my site as well, updating hourly. The eventual plan for these docs is to make it into the Pinax or Django Official documentation. I think that they can probably go into the Pinax documentation once we clean them up a little bit, and I don't know if this is quite something that belongs in Django docs. So I invite everyone to come discuss what the conventions should be, contribute your own, and lets try and make some great reusable apps.

Using rsync with django

Just a quick warning/tip on using Django with rsync, for other people pulling their hair out later.

When you use rsync a good way to get a directory is using `rsync -aCq`, which means recursively, quietly, move a directory ignoring common files. The -a command means 'archive'; keep permissions and as much data about the files as possible. We use -C because it ignores .pyc and .svn files. However, in the list of included files is 'core', so that you don't move over core dumps.

Django however has a core directory inside of it, and using -C causes rsync to ignore that directory. So we ended up using the rsync command like so:

```
rsync -aCq --include=core
```

Hopefully this saves people some time trying to rsync Django in the future. I'd be curious what other rsync commands that people use for moving around django and/or other files.

Incredibly useful SSH flag

So at work we have a lot of different django environments, scattered across varies servers. All of this information is kept in a central resource. We have the pythonpath, settings file, and the remote server that the client is on. So every time that I want to go do work on a different site, I have to ssh into that server, set the PYTHONPATH and DJANGO_SETTINGS_MODULE environmental variable, and then do what I want to do. This is not a huge deal, but it is annoying when you're doing it 10-20 times a day.

So I went along and tried to figure out the best way to handle this situation. One of the popular ways to achieve this type of functionality is a simple shell script on the remote client, that sets the environment. You ssh into the client, change into the clients directory, and source it into bash to get into your environment. This didn't really solve my problem, because it's basically what I was already doing.

It would be really nice to be able to run an ssh command, to the remote server, and have it executed. SSH has this ability, simply by passing a command after your connect string.

```
$ ssh ericholscher.com uptime
19:11:28 up 69 days, 21:29,  1 user,  load average: 0.00, 0.00, 0.00
```

However, this simply executes and returns. I want to be able to set up my environment on the remote side. With the `-t` command to SSH, this allows you to do exactly that. It sets up a psuedo-tty on the remote side, which then lets you execute commands! So you can do something like this:

```
$ ssh ericholscher.com -t DJANGO_SETTINGS_MODULE=test.settings bash
eric@Odin:~$ env |grep DJ
DJANGO_SETTINGS_MODULE=test.settings

$ ssh ericholscher.com -t PYTHONPATH='$HOME/Python' django-admin.py dbshell
SQLite version 3.5.9
Enter ".help" for instructions
sqlite>

$ ssh ericholscher.com -t PYTHONPATH='$HOME/Python' django-admin.py shell
[..ipython startup truncated..]
In [1]: import django

In [2]: django.VERSION
Out[2]: (1, 1, 0, 'alpha', 0)
```

Note that the PYTHONPATH variable is quoted. This is because I am using the $HOME variable, if you don't quote it, it will be evaluated on the client side. If you quote it, it gets passed to the server to be set.

I have written a datastore backend that keeps all of my sites settings and pythonpaths (along with other information), and then I wrote a simple wrapper script around that. I will hopefully be releasing this code in the near future, but it allows me to do things along these lines.

```
$ ./assume.py production my_site shell
$ ./assume.py production test_project shell
$ ./assume.py staging my_site dbshell
$ ./assume.py local my_site syncdb
```

and end up in the correct place.

A really neat part of this is that when you do something like dbshell, you are sent to the remote site, and you can perform an action. However, when you leave that command, you are brought back to the local environment that you were already working in. This makes it really easy to be able to do simple one off actions on a remote site (like fixing a bug). This comes up a lot at work, eg: someone is reporting X on server Y, can you please check the database over there really quick.

This gives you a lot of value because you are connected to the database with the credentials in the settings. It really allows you to abstract the knowledge that is in your settings files and build commands around them. Any custom management commands that are defined on the remote server are also really easy to activate as well. This allows you to think "I need to reindex the search on client Z", you simply do something like `./assume.py Z reindex`, and then go on about your previously scheduled activity. You don't have to think what server the client is running on, what version of django, where the code lives, or anything like that.

I'd love to hear other ideas that people have for this kind of system. I think that a simple reusable app that keeps track of your different servers and commands would be really neat and useful. Perhaps integrating it into Fabric or something would be possible as well.

Automatically apply patches from Django's (or any) Trac

Lately I've been delving into Django development a bit more, and applying people's patches has been a bit of a hassle. You know you want to apply someones patch, but there are about five steps in between you and applying their patch to your source tree.

So I'd like to present trac_patch.py, which allows you to apply a patch from Django's trac automatically. It is posted on github, so I encourage everyone to fork it and modify it to fit your own workflow. This was done in about 2 hours, so it's still pretty rough. Also note, that this should work with a small modification on just about any trac install out there.

I threw a few features in that were useful for my development workflow. You can easily create a new git branch automatically with the name of the patch that you're applying. You can apply and revert a patch. It also has a mode where you can confirm the ticket you're looking at, and choose which of the patches on the ticket you wish to apply to your code.

You can use it by default if you're in your current top-level django directory (or where ever you want the patches applied). However, there is a `django_src` variable in the code that you can set and then it will work from anywhere.

I'll paste in the modules docstring below, so you see some examples of it in action.

```
Description::

Simple utility to grab and apply a Django trac ticket.
It could in theory be used for any trac installation.

Usage:: trac_patch.py [ticket_num]

  -h, --help     show this help message and exit
  -r, --reverse  Reverse the patch
  -g, --git      Make a git branch
  -a, --ask      Make a git branch

Examples::
    trac_patch.py [ticket_num] [-r]

    #Apply patch 6378
    trac_patch.py 6378

    #Reverse patch 6378
    trac_patch.py 6378 -r

    #Create a git branch and apply patch
    trac_patch.py 6378 -g

    #Confirm patch filename and ticket filename
    trac_patch.py 6378 -a
```

Google Summer of Code

It's that time of year again, and the Google Summer of Code is happening again. This year Django will be applying again, and there is currently a Wiki page on the Django wiki devoted to ideas and people who want to volunteer to help mentor. I think this is a great opportunity for students, mentors, and the projects involved. It is a really neat learning experience. Even if you can't participate, it's a good chance to put ideas up that some enterprising student might pick up and run with.

Last year had some really successful work done, with Django's Aggregation framework, F Expressions, and a revamped comments framework all coming out of last years efforts!

This is your time to really help give back to the community and help some neat, innovative work get done at the same time. You can also show a student the value and fun in open source work. The focus on SoC is in new feature development, so it's an exciting way to hopefully get your pony into Django. So go add some ideas to the wiki or volunteer to mentor if you have the time and knowledge. Most of all if you're a student, apply! This is a really great opportunity that Google provides you, you get paid, and you get to help make something amazing!

Twitter Spam

I keep hearing people talking about how twitter is going to be over run with spam now that it is becoming mainstream. I really don't understand this viewpoint, and will take time here to outline what they could be talking about, and what can be done.

This is in reply to http://www.twine.com/item/123c9051b-g8/can-twitter-survive-what-is-about-to-happen-to-it specifically, but these ideas have been mentioned over and over.

Short Version: **We need to stop worrying about spam on twitter, and start worrying about all the cool stuff we can make.**

Kinds of Spam

"Hypertweeting" A person you are following is tweeting too much? How is that spam? Simply unfollow them. This is one of the big ones I don't understand people complaining about. It's OPT IN to follow people; if you don't like what they say, unfollow them!

Hashtag Spam The current implementation of hashtag spam is indeed a problem, because it is a publish and not a follow model. So anyone can include a #hashtag and it will get picked up by a hashtag aggregator. This is the common problem of broadcast mediums. It can be solved filtering hashtags to only certain users, or some other kind of grouping concept. (A twitter account that retweets a hashtag only from the people it follows, for example). You could also do the filtering on the web end, showing only hashtags from user X and Y.

This seems like a problem that could be solved by the hashtag aggregators. Currently they are just dumb aggregators, and adding relevancy would probably be easy. This also screams out as an area where Bayesian Filtering would be useful, since you have a tag that is presumably about a topic.

@reply Spam This is legitimate. If a spammer gets on twitter and @replies your account, it will show up in your timeline. However, that spammer can only @reply 1 person a minute, and that kind of activity should be really easy for twitter to take care of. Alternatively, twitter could implement an option where you only receive @replies from people that you follow (like the settings to see their @replies to other people). This issue can also be solved in a client by separating @replies you get from people you follow, and those that you don't.

All in all, this is not a very worrisome method of spamming, and if it became used, it would statistically almost never happen to you.

Notification Overload Again, this is the same as the "Hypertweeting argument". I follow @slicehoststatus because it is just updates about my connectivity. They have a @slicehost account that is more customer service oriented that I don't care about. Services will have logical separation between their feeds or they won't be used.

Solutions

The post does mention some good solutions to the problem. I will address my thoughts on those here as well.

Number of Followers as a Filter. This seems likely to be gamed, and a trivial filter. This might be useful when combined with other metrics, such as how long a user as been on the site, how many people they are following (and have followed!). This is where the idea of metadata being important comes in.

Re-Tweeting Activity as a Filter. Perhaps, but this needs to be formalized. I would really like Twitter to formalize RT's so that I can filter them out, because I find very little value in them personally. Twitter already has functionality in it (liking tweets) that seems like a more logical choice to use.

Social Network Analysis as a Filter. I love the social graph, so I'm a fan of this one. This would be a very resource heavy way of validating anything, and the whole premise of the spam argument is that twitter is growing really fast. I don't think this is a viable option, at least not when applied to every message. There is a lot of interesting data to be gleaned from the social graph, however that is another post.

Metadata for Filtering. He makes a good point that metadata is what is needed. It will be really hard to do these calculations outside of twitter (especially as it grows, it will be hard internally). I really think that the spam problem is something that is a non starter, and twitter will work just fine without any more measure of spam protection. The metadata will be really interesting for a lot of other applications.

Conclusion

I have yet to see a real post that has made me think that twitter will have a spam problem. The opt-in subscription method is really genius, and makes spam almost impossible. The model of twitter will stay spam free (I will get content from people I follow). External services (search and aggregators) will suffer from spam problems, until they get better (spam) filtering.

I think the real problem of twitter is how to find interesting people to follow, and not how to remove spam. This is where the problem of spam and filtering really come into play. Starting with a network of people you know, and branching from there is how twitter will work. The social graph is really interesting in that realm.

A lot of the conversation above leads itself into other really interesting areas of data analysis. Stopping spam is easy, let's go data mining :)

Really easy SSH tunneling

SSH Tunneling has become an invaluable tool that I probably use more than I should. I love tunneling, and use it all the time. This will be a quick tutorial on how to use the SOCKS proxy ability of SSH to allow you to tunnel your HTTP traffic through a remote server.

This is useful when you're on a connection that has a silly filter on it (school or library). Since it's a SOCKS5 proxy, it is useful for tunneling other things as well (like IM). It is also useful when browsing on public wifi or anywhere that you can't trust the network connection you're on, since it encrypts all the data that is sent over it.

SSH

The command to tunnel in SSH is really simple. You simply do: `ssh -ND localhost:5555 example.com` to tunnel traffic through example.com. This is a nice one off, but I actually have the configuration in my ssh config. To do that, in your ~/.ssh/config, you need to put in the settings you want your proxy to have.

```
Host tunnel
    Hostname example.com
    DynamicForward localhost:5555
```

This allows me to simply do `ssh -N tunnel`, and it will setup a proxy. This is basically turning my local port 5555 into a proxy that goes through the remote host. It is encrypted from my network to the remote network, which is really nice. The `-N` flag is used so that it doesn't create a shell on the other end, and simply creates the proxy connection.

Firefox

In firefox, you need to go into your Preferences > Advanced > Network > Connection > Settings. This is where your proxy settings live. Go down the the SOCKS host, and set it to localhost, with the port you set up above, 5555 in this case. It should look something like this:

Figure 2.5: Configuration for Proxy

I use the Quickproxy extension to easily turn my proxy settings on and off. It puts a small button on your bottom status bar in Firefox, and clicking it turns your proxy on and off.

Now you simply flip the switch on your QuickProxy, and you are surfing through an encrypted connection. To check if it's working, I use http://whatismyip.com to check my remote IP. If it changes between the proxy being on and off, you know the proxy is working.

This is a really easy way to simply create a two click encrypted proxy. Hope this is helpful, and I'd be curious if people have other tips and tricks in this regard.

Pycon and Euro Djangocon

The ice is starting to thaw and I'm making my way out from under my first winter in Kansas. You know that spring is coming because the conference season is starting to bloom. I'm looking forward to a bunch of conferences that will be upcoming in the new few months. I'll be attending two of them, and hope to see lots of interesting people there!

Pycon

First up is Pycon, which is happening at the end of March. I am really excited to be going, seeing some of the great talks that are planned, and meeting some of the awesome people in the community.

Following the conference, there will also be a Django sprint that I will be attending. I know the 1.1 release candidate of Django is supposed to be released some time during the sprints, so that will be exciting!

Euro Djangocon

I submitted a talk that was accepted for Euro DjangoCon that I am really excited about. I will be talking about Testing in the Django realm, which I think is one of the areas where our community needs to improve. I'm looking forward to having discussions with people about how we can make testing (even more) amazing in Django. It will be a great opportunity to get to Prague, and enjoy the European Django Community that I have only met online.

The conference talks will be on May 4-6. There will also be two days of sprints after this one, which I assume will be used to brainstorm and get some neat things started for Django 1.2.

I'm really looking forward to these conferences, and I hope that everyone can get a chance to come out and participate in the discussion about Django and the community.

Testing AJAX Views in Django

A lot of the Django code we use at work has a special case for AJAX. It has been a kind of a pain to test, because the test client by default doesn't use AJAX. Luckily the is_ajax call in the Django HttpRequest object is a simple check of an HTTP Environmental variable.

An undocumented feature of the Django Test Client is that you can pass in custom HTTP ENV variables on requests. The definition of get for example is:

```
def get(self, path, data={}, follow=False, **extra):
```

Later on in the file, the request environment is then updated with the extra keyword args: `r.update(extra)`.

This lets us throw in arbitrary variables in our get and post requests in the test client. Like so:

```
r = self.client.post('/ratings/vote/', {'value': '1',},
                                HTTP_X_REQUESTED_WITH='XMLHttpRequest')
```

Note that the custom env is outside of the dictionary of get parameters. This will now return the /ratings/vote/ view with the output that is normally called on an AJAX request.

Django's Summer of Code students announced!

Today is the day that Google has announced the accepted projects for the Summer of Code. Django has 6 spots this year, with a bunch of exciting projects. I am lucky enough to be mentoring Kevin Kubasik with his project "Upgrade the Awesomness Quotient of the Django Test Utils and Regression Suite". I'm really excited for the opportunity to help improve Django by overseeing Windmill testing of the admin, and lots of other small testing improvements that will hopefully make it into trunk.

I would like to congratulate all the students that got accepted, and to everyone who didn't, you can still contribute! Also remember that contributing through the year will make the chance of you being accepted next year a lot higher! Here is the full list of accepted proposals:

Honza Král, "Model aware validation" Mentor: Joseph Kocherhans

Kevin Kubasik, "Upgrade the Awesomness Quotient of the Django Test Utils and Regression Suite" Mentor: Eric Holscher

Christopher Cahoon, "Improved HTTP and WSGI Support" Mentor: Malcolm Tredinnick

Zain Memon, "UI improvements for the admin interface" Mentor: Jacob Kaplan-Moss

Marc Albert Garcia Gonzalo, "Implementation of additional i18n features" Mentor: Jannis Leidel

Alex Gaynor, Multiple Database Support in Django Mentor: Russell Keith-Magee

Note that you can view the full list with abstracts on Google's site.

Looking forward to a summer full of great new features and amazing community involvement for Django and all the organizations participating. A big thank you to Google and Jannis Leidel for sponsoring and administering Summer of Code, respectively.

Adding Google Analytics to Sphinx Docs

This is just a reminder for myself later, or people looking on Google. Also note, that this method is useful for putting any Javascript content into your sphinx docs, but Analytics tracking is a common use case.

Step 1: Where to put my files?

Check your conf.py on your Sphinx docs. You need to make sure your `templates_path` variable is pointed to a directory that exists. It is relative to your current directory. I use `_templates`, which I believe is the default. This is where you can override Sphinx templates. They use Jinja2, which is a relative of Django templates, so it should be pretty simple if you're used to Django templates.

Step 2: Override the default template

In your `_templates` directory, add a file called layout.html. The Sphinx Docs are pretty good in this area, containing a full listing of all the template that you can override. The Sphinx Source layout.html is also really nice, so you can see what it is by default.

Analytics says that you should put your analytics code "Right before the `</body>` tag" on your site. This means at the bottom of the footer. Google should have given you a piece of Javascript code to paste in your site, copy that below. So in your new `layout.html`, put in the following code:

```
{% extends "!layout.html" %}

{% block footer %}
{{ super() }}
<script type="text/javascript">
var gaJsHost = (("https:" == document.location.protocol) ? "https://ssl." : "http://www.");
document.write(unescape("%3Cscript src='" + gaJsHost + "google-analytics.com/ga.js' type='text/javasc
</script>
<script type="text/javascript">
try {
var pageTracker = _gat._getTracker("YOUR_GOOGLE_CODE_HERE");
pageTracker._trackPageview();
} catch(err) {}</script>
{% endblock %}
```

Take note of the `{{ super() }}` call. This means that you are calling the inherited template's code, which pulls in the default Copyright notice into the footer. Then you can put in your custom code after that.

Let me know if there are any other neat Sphinx tricks and tips that you have. I'm in love with the software and learning more about it daily.

A playground for Django Template tags and filters

The Problem

Any sufficiently large Django project starts to have a wide variety of Template Tags and Filters. Even Django ships with a dizzying array of them that allow you to do all sorts of fun and interesting things. Ellington, our CMS at work, has a ton, and I've been thinking about ways to make tags and filters a bit more accessible to people who are using the CMS.

I'm thinking along the lines of people who are tech savvy, but who were just hit with a huge wall of tags and filters to look at. I want them to be able to really easily play with the functionality and see what it does.

The Solution

I created a proof of concept playground for tags and filters in the django admin. It is released as a simple third party app that I have up on Github. Here is a small 1:40 minute screencast that explains what I did:

Django Admin Playground from Eric Holscher on Vimeo.

It gives you a "Play" link next to each of the tags and filters in the admin. Once you click on that, if the docstring for the tag has a code example, it attempts to parse that out. This allows you to easily test out the examples that you should have in the docstrings for your tags.

It displays the docstring above the input areas and allows you to input context variables (naively) and render the template. It uses Jquery to do an ajax post and response that is displayed on the right side of what the output of the template would be.

A simple example with the Add Template Tag:

Figure 2.6: Add Template Tag

Caveats

This is very hacky and basic code. Totally just a proof of concept and might not work for you. I think that the ideas are worthwhile, and something that could be included in Django at some point.

Currently the context values are just being parsed at the : and split into a dictionary. If anyone knows a good way to turn a basic list like this (using YAML?) into Django objects, then I would be all ears. I thought about it a little bit but couldn't think of an elegant solution.

Also the parsing of the templatetag syntax out of the templates is incredibly simplistic. If I took some time and played around with ReST I'm sure I could figure out a better way to do that (Pulling out the "code blocks" somehow?). But a basic regex worked well enough to get the idea done.

Feedback welcome.

EuroDjangoCon Talk: Testing Django

Just got off the stage at EuroDjangocon, which was my first real talk in front of the Django Community. I hope that people enjoyed it, and that it was informational. Here are the slides to my talk in PDF Form, and on slideshare.

Testing Slides

I hope that people have questions or constructive feedback, and I'd love to hear what everyone thought. I will be writing a full writeup at the end of the conference, but I'll say that it has been amazing thus far!

Migrating Django Test Fixtures Using South

The Problem

Migrating test fixtures is one of the biggest pains of testing. If you create your tests too early, then change your schema, you have to go back and touch all your old test fixtures. This discourages people from writing tests until their app is relatively 'stable'. As we all know, this may never happen :) This solves half of the problem, the part where you have to manually change a bunch of fixtures to reflect changes in your schema or data.

Possible Solution

During the questions in my EuroDjangoCon talk, someone asked a question about this. I didn't have a good answer, but someone from the crowd raised their hand and suggested using South. Once your project has data migrations for it's real models (like any production site should), **it should be relatively easy to then load up your test fixtures, migrate your database, and dump them back out**.

I will be writing out a pretty basic tutorial on south, alongside of the example of how to migrate your test data using South. I think it's pretty fantastic. I hope that if you aren't already using south, this tutorial will show it's simplicity

and power, and if you are, I hope to show you another way to use it. If you already understand south and have data migrations, you can skim the Example section and just focus on the later part.

I am using my Django Test Utils's test_project at a certain revision for this demo, so you can look there and follow along if you want to see the actual files used. Look at the next commit to see the outcome of the project :)

Example

I went ahead and made a simple little example application to test this on. It will be the common migration scenario of adding a slug to a model. We will be using the Polls app that everyone knows and loves from the Django Tutorial.

I don't currently have south installed at this point in test utils, so if you're following along, you just have to download south and add it to the installed apps before you start.

Basic Setup So you have the basic polls models.

```
class Poll(models.Model):
    question = models.CharField(max_length=200)
    pub_date = models.DateTimeField('date published')

class Choice(models.Model):
    poll = models.ForeignKey(Poll)
    choice = models.CharField(max_length=200)
    votes = models.IntegerField()
```

We realize that we don't have a way to show these well on the site, because they don't have a slug. So we want to add a slug to the Poll model. First off you need to have the initial migration for your app, so that we can migrate it. We are going to use south, so we need to create our initial migration.

```
$ ./manage.py startmigration polls --initial
Creating migrations directory at '/Users/ericholscher/lib/django-test-utils/test_project/polls/migrat
Creating __init__.py in '/Users/ericholscher/lib/django-test-utils/test_project/polls/migrations'...
 + Added model 'polls.Poll'
 + Added model 'polls.Choice'
Created 0001_initial.py.
```

Adding your fields As you can see, south knows how to add a migrations directory to your app and fill it up with the correct migration name. Now lets go ahead and edit our model.

```
class Poll(models.Model):
    question = models.CharField(max_length=200)
    pub_date = models.DateTimeField('date published')
    slug = models.SlugField(null=True)
```

As you can see, we added a slug field. It has to be null=True because we will be creating a lot of them, and they must be able to be null before we can add the data to them. Now lets go ahead and create two migrations. We want to add one that creates the field, and then we want to create one that fills out the slug from the question field.

```
$ ./manage.py startmigration polls add_slug --auto
 + Added field 'polls.poll.slug'
Created 0002_add_slug.py.
```

Writing your Data Migration This creates the migration that we want that allows us to add the field. Now we go ahead and run `startmigration` again, just passing the app name. This creates a stub migration for us, with the model serialized on the bottom, which allows us to just write the code we care about.

```
$ ./manage.py startmigration polls populate_slug_data
Created 0003_populate_slug_data.py.
```

Note that it is a good practice to separate your migrations that effect your table structure and things that actually migrate data. Now we go in to the migration and add in the code that migrates the data. It will end up looking something along these lines.

```
from south.db import db
from django.db import models
from polls.models import *
from django.template.defaultfilters import slugify

class Migration:

    def forwards(self, orm):
        for poll in orm.Poll.objects.all():
            poll.slug = slugify(poll.question)
            poll.save()

    def backwards(self, orm):
        "Write your backwards migration here"
        for poll in orm.Poll.objects.all():
            poll.slug = ""
            poll.save()

... Chopped for clarity ...
```

As you can see, the migration is really simple! This uses a fake Django ORM (which is just the real one, loaded a different way.) Now you can go ahead and test out your fancy new migrations.

Running the migrations on your test data.

Now as you see, you have these fancy migrations that actually haven't touched your database yet. I'm going to walk through the entire process of creating your database from the `syncdb` stage to the outputting of your shiny new test fixtures.

Setting up your test database So the whole point of this exercise is to be able to migrate your test fixtures the same way you do your real database. This means that we simply load up a new version of our database with our test data, run our migrations, and serialize it back out, ready for our tests.

Go ahead and run `syncdb` on your project. This will do all the normal things you're used to, except that at the bottom of the output, you'll see a message about things not being synced because of south:

```
Synced:
 > django.contrib.auth
....

Not synced (use migrations):
 - polls
(use ./manage.py migrate to migrate these)
```

Now we need to go ahead and get the polls data in our database at the point where our fixtures exist. This means that we only want our initial data to be loaded. So we go ahead and tell south to migrate to our first migration.

```
$ ./manage.py migrate polls 0001
 - Soft matched migration 0001 to 0001_initial.
```

```
Running migrations for polls:
 - Migrating forwards to 0001_initial.
 > polls: 0001_initial
   = CREATE TABLE "polls_poll" ("id" integer NOT NULL PRIMARY KEY, "question" varchar(200) NOT NULL,
   = CREATE TABLE "polls_choice" ("id" integer NOT NULL PRIMARY KEY, "poll_id" integer NOT NULL, "cho
   = CREATE INDEX "polls_choice_poll_id" ON "polls_choice" ("poll_id"); []
 - Sending post_syncdb signal for polls: ['Poll']
 - Sending post_syncdb signal for polls: ['Choice']
```

Migrating your test data As you can see, this created out database table without the slug field. This is good, because our fixture data doesn't include the slug field. This is where things get a bit annoying. The loaddata command uses the models that are on disk to check if the data loads correctly. So you need to check out your code at the revision before the migrations were applied (in our case, we can simply comment out the slug line). Then you are able to go ahead and load your test data.

```
$ ./manage.py loaddata polls_testmaker
Installing json fixture 'polls_testmaker' from '/Users/ericholscher/lib/django-test-utils/test_projec
Installed 8 object(s) from 1 fixture(s)
```

Then you can put the slug back in (or check out the current version of your code). Now you have your data in your database in the old un-migrated form. Let's go ahead and migrate out test fixtures :)

```
./manage.py migrate polls
Running migrations for polls:
 - Migrating forwards to 0003_populate_slug_data.
 > polls: 0002_add_slug
   = ALTER TABLE "polls_poll" ADD COLUMN "slug" varchar(50) NULL; []
 > polls: 0003_populate_slug_data
 - Loading initial data for polls.
```

Now lets see if that worked. Let's go ahead and run dumpdata and see what all you have.

```
./manage.py dumpdata polls --indent=4
[
    {
        "pk": 1,
        "model": "polls.poll",
        "fields": {
            "pub_date": "2007-04-01 00:00:00",
            "question": "What's up?",
            "slug": "whats-up"
        }
    },
... snip rest of data ...
```

You now have your migrated data fixture! Hopefully everything worked for you, and that this works for larger examples other than this trivial example.

Conclusion

The little bit at the end where you have to revert back to the old version of your code to use loaddata is a bit of a hack. With a bit of tinkering, you should be able to use south's serialized representation of the model instead of the models on disk in order to load the data. Doing this will make this whole process more seamless.

If you would like to see the changes to the models and fixtures, and migrations that were created, you can check out the south demo branch of test utils.

I would also like to thank Andrew Godwin for creating south, of which none of this would be possible.

Thanks for reading, and I'd be curious to see what people think, and if there are some improvements that could be made.

Enable setup.py test in your Django apps

Setuptools comes with a way to run the tests on your application. This allows the user of your software to download it, and run `python setup.py test` and check to see if the tests in your application pass. This is really useful for distribution, because the user doesn't need to know or care how to run your tests (nose, django, unittest, py.test, or whatever else), and can simply see if they pass.

To do this, you simply define a `test_suite` variable in the `setup()` function of your setup.py. This argument is a callable that should return a test class. However, since Django has it's own test runner, we have to point this at a simple test runner that we construct, and allow that to run the tests. This is because we must set a couple of environmental things, like the settings module and PYTHONPATH.

I did this with my test_utils project, you can see the commit here, but basically I simply added this line to my setup.py:

```
test_suite = "test_project.runtests.runtests",
```

Then put this in the file `test_project/runtests`:

```
#This file mainly exists to allow python setup.py test to work.
import os, sys
os.environ['DJANGO_SETTINGS_MODULE'] = 'test_project.settings'
test_dir = os.path.dirname(__file__)
sys.path.insert(0, test_dir)

from django.test.utils import get_runner
from django.conf import settings

def runtests():
    test_runner = get_runner(settings)
    failures = test_runner([], verbosity=1, interactive=True)
    sys.exit(failures)

if __name__ == '__main__':
    runtests()
```

Note that there is a bit of path and settings hackery, this is to enable Django to run correctly, with the test_project in the PYTHONPATH. However, now if you want to run the tests, you simple do a `python setup.py test`. You can also do a `python runtests.py` to have the same outcome. You could also have the runtests function simply be a `call_command('test')`, but without the explicit sys.exit, setuptools complains that it hasn't been given a TestCase back.

This has a couple drawbacks in that it simply runs the entire test suite. You can pass in a Test Suite to setuptools, but that isn't how the tests are organized in most Django apps. However, if you want to run specific tests, you can still test things the regular way through `manage.py test`. Presumably there is a better way to do this, but it's good to at least have a hacky way until a better way emerges.

There is a lot of value in providing a standard interface to running the tests on your application though. This allows the distribution tools (pip and setuptools) to run the tests on your application if they'd like. In the perl universe, CPAN runs the tests on apps before installing them, and quitting if they fail. If more people started making setup.py test work, we could hopefully add this ability to Python's distribution tools, and make the world a happier place with better working software.

Pretty Django Error Pages

Continuing on with the simple tricks that make everyone's life a little bit better, I know a lot of people hate that Django's 500 pages don't get rendered as a RequestContext. This means that if you have context processors (like one that sets a MEDIA_URL), they don't get called. This was causing our 500 pages not only to make users sad because something broke, but knock them out of context becaue our entire design blew up.

Luckily, Django makes it incredibly simple to redefine your 500 handler in your URLConf. Most pythonistas know that `import *` is a bad thing, but it is standard in the Django community in your URLConf to do a `from django.conf.urls.defaults import *`. This has the effect of pulling in Django's default `handler500` function. So if you want to override Django's default, you simply set it up like so.

```
from django.conf.urls.defaults import *
handler500 = 'path.to.my.sweet.views.server_error'
```

Then you simply define a server_error view that renders the error page with a RequestContext.

```
from django.shortcuts import render_to_response
from django.template import RequestContext

def server_error(request, template_name='500.html'):
    """
    500 error handler.

    Templates: `500.html`
    Context: None
    """
    return render_to_response(template_name,
        context_instance = RequestContext(request)
    )
```

If you're feeling extra special, you can even change the template rendered. Note that you can also do this for the 404 handler by defining a `404handler` in your URLConf in the same fashion. Then you can get pretty error pages!

Hacker Book Club

At LPDN, our weekly programmer drinkup, we have been talking for a while about watching the SICP lectures from MIT as a fun thing to do. I then got to thinking about how it would be neat to involve more than just the few of us in Lawrence. Everything is more fun on a larger scale, and having compatriots makes you more likely to finish it. Somewhere along the lines of the Infinite Summer, I was thinking about having some kind of Hacker Book Club.

SICP may be a bit intense for the beginning of the club, but it's something that I think that everyone can learn and have fun with. Along with the typical book club style things, seeing the audience, I think we could make some really neat additions to the idea. A few have come to mind, but I'm sure there are lots more.

I would imagine having an IRC channel where we can all hang out and ask each other questions. I'm sure the #scheme channel will be a resource as well, but it could be neat to have a focused group of people focused on learning the same material. It feels a bit like an open source classroom, where disperate people can come together to learn and share in the open source mindset. Sharing is caring, not cheating.

Since SICP is available for free online, we could suck the book into a database of some sort and allow a commenting/discussion medium around the text as well. The Django Book is a good example of that. I think it would be interesting trying to tie in discussion on the site with IRC. Perhaps have some kind of encoding that allows you to reference pages in the book on IRC and have that inline the log transcripts. My friend Nathan also runs Readernaut, and already has a basic syntax established for tracking book progress on Twitter. We could throw this on IRC and twitter as well, where each user could register their account and keep progress.

I think it would be a really interesting way to combine a lot of the conversations that happens across the web in one place, with context, about a book. I have also wanted similar abilities in other things (Think your work IRC channel and code instead of Book pages). This could spawn a neat open source project, if we were so motivated.

Let me know what you think!

Debugging Django in Production Revisited

In a previous post I talked about a neat middleware to debug production environments in Django. It basically checked to see if you were a superuser, or if you were in settings.INTERNAL_IPS, and if so, then it displayed a technical 500 page for you (The yellow one you know and love). Anyway, at that point it was more of a simple idea, and not really used in production.

At work the other day I was working on a bug that was only showing up in production, and not on staging. I remember back to this middleware and thought it would be perfect. Anyway, at work we have a lot of non-technical people that are superusers (think my bosses boss). We also all have the same external IP's when at work, so none of the previous methods I had would work for this.

Thinking about it, and talking to my co-worker Ben Spaulding, we thought that Django has Groups built in, so why not use that? So I went ahead and re-jiggered the middleware to be based around groups.

```
from django.views.debug import technical_500_response
from django.contrib.auth.models import Group
from django.core.cache import cache
import sys

class UserBasedExceptionMiddleware(object):
    def process_exception(self, request, exception):
        users = cache.get('technical_error_users')
        if not users:
            skip = cache.get('no_technical_error_users')
            if skip:
                return None
            try:
                g = Group.objects.get(name='Technical Errors')
                users = g.user_set.all()
                cache.set('technical_error_users', users, 60)
            except Group.DoesNotExist:
                cache.set('no_technical_error_users', True, 60*60)
                return None
        if request.user in users and request.user.is_superuser:
            return technical_500_response(request, *sys.exc_info())
        return None
```

Since it is middleware, I went ahead and decided to use the cache framework to make sure that we weren't doing a DB query on every request. Also, I had to account for the case when the group hasn't been added yet, so when that happens, it caches the fact and doesn't check again for another hour. If the Technical Errors group exists, it caches the members that are in it for a minute. This means that a DB query only happens every minute, which is fine.

I'd be curious how other people might improve this, as it seems a little bit janky still. However, it works for us, and is incredibly useful when debugging. Instead of getting a link to a broken page, you go to the page and get a nice 500, telling you exactly what went wrong.

I can think of one basic improvement in just writing this post, which would be to import settings and to just return None if DEBUG was True, or if the CACHE_BACKEND was set to None. This would allow it to stay out of the way if there was no caching, or the Technical 500 was already going to be raised.

I do think that this middleware removes a lot of the reason to run a site under DEBUG=True, so hopefully it will result in less sites launching with DEBUG on.

Token Testing Talk Slides: Djangocon 2009

There are the slides to my Token Testing Talk from Djangocon. I'm hoping the videos will be posted soon, but I think that it went well. There were a lot of good questions, and I need to put some recap posts up, but for now here is a copy of the slides. PDF

If you have any questions or comments, feel free to leave them below.

Token Testing Slides

View more documents from ericholscher.

Easily Running the Django Test Suite

Update As of Django 1.2, Django ships with default test settings for sqlite. They require two databases to be defined, because of Multidb. More information at Django advent and in the docs

Alex Gaynor had a write up about Running the Django Test Suite, which is a quick overview of how to run the suite. The official docs also have a simple mention of how to run them. This post will be more step by step, walking you through the steps to run the tests for Django. This is a really important first step in writing patches against Django. It is easy, but something that a lot of people have a question about when they start.

Step 1: Grab the Django Source

To test Django, you need the code, so go ahead and grab the source.

```
svn co http://code.djangoproject.com/svn/django/trunk/ django_src
```

Step 2: Settings

In order to run the Django test suite, you need to have a settings file. Usually for testing, you run the Django test suite under SQLite. This is the easiest and fastest way to run the tests. If you writing code against something that touches parts of the ORM or Database code in general, running it against another database that you have at your disposal if generally a good idea as well.

To run the SQLite tests, you simply need a settings file with one line in it:

```
DATABASE_ENGINE = 'sqlite3'
```

Go ahead and put this command in the top-level of your django checkout (the one with the `tests` directory in it).

Step 3: Run the tests

Now you can run the tests. Your checkout should look something like this:

```
django_src/
    docs
    django
    examples
    setup.py
    tests
    settings.py
    ...
```

We need to make sure that Django is on your PYTHONPATH, this allows Python and thus Django to see the django module that we want it to test. You can set this inline, and then run the tests with the correct settings file. We can do that in a single command like so:

```
PYTHONPATH=`pwd` ./tests/runtests.py --settings=settings
```

The final commands, which you should be able to copy and paste into a shell to check out the code and run the tests is as follows:

```
svn co http://code.djangoproject.com/svn/django/trunk/ django_src
cd django_src
echo "DATABASE_ENGINE = 'sqlite3'" > settings.py
PYTHONPATH=`pwd` ./tests/runtests.py --settings=settings -v1
```

The `-v1` will set the verbosity to 1, which gives you the dots that everyone knows and loves.

Now that you have a django source tree with running (and hopefully passing) tests, you can apply a patch or go ahead and develop on this code and be able to test it easily!

What they didn't teach me in college

Updated at the bottom of the post.

Warning: This is a bit of a brain dump.

In the software industry there is a lot of back and forth about the value of a college degree. This post won't go into that too much, I just want to talk about the notable things that were left out of my Computer Science degree. Mostly things that are used in the day to day environment outside of a university, but aren't used extensively inside of them.

My degree was a more classical CS degree, which focused on algorithms and theory. However, there was a decent bit of actual "real world" knowledge that they tried to impart. After being at a real job for over a year, I think it is interesting to look back on what I wasn't taught.

Software testing

In college, the concept of testing was basically 'compare your work against this expected output file'. Sometimes that job was automated, other times it wasn't. There was absolutely not concept of an automated test suite. However, I think that this may be a limitation of the semester long class idea. A lot of the value from testing comes from things that are real (production, refactoring) or looking back at code that you wrote a long time ago. I have a lot more thoughts on this, and it deserves it's own post. However, it was certainly a glaring part of what I do now that I had no experience with out of school.

Version Control

In the classes that I took, we simply submitted the work to the teacher and that was that. We didn't check it into a repository, or even version the work we were doing locally. At the time the whole DVCS movement wasn't quite as big, so I can imagine a lot more people doing local versioning now. I think that the fact that viewing other students work is sometimes considered "cheating" (which is silly), makes it difficult to have a shared repository for all students.

A big problem with universities is that knowledge the old mindset that sharing knowledge is cheating. Luckily mine was a bit more enlightened, but I think having a shared repository of code would make this philosophy a bit too "real".

Web development

We had optional classes that offered PHP/MySQL based website making, but nothing in the curriculm about web development. It seems that in this day and age, so much of what we do is centered around the internet that ignoring it in the classroom is silly. That may be the fact that I now do web development, but I feel that someone coming out of a Computer Science degree not understanding the basics of Web Development is a bit silly.

Bug Tracking / Maintenance

Our code in university only had to be written once. There was no concept of going back and looking at old code and fixing it. It is one of those realities of everyday work that is totally ignored by universities. I think this one may be a bit hard for them to teach, but mostly because of systematic problems.

What to do about it? I think that a lot of the problems come from the **Single Semester Class Paradigm**. You do some kind of programming for a class, and then it disappears into the ether never to be seen again. A lot of the value and reality of coding is that you write code and then have to keep changing it and making it work.

Imagine if you were tasked with writing code your first year. This code was checked into the schools version control system in a branch of that class for that semester. Then in your following year you have a class that recalls that code, and you update it with some new technique you have learned. You are a bit dismayed at how badly you used to code, and how hard it is to understand.

Then your third year you go back to your code and have to write tests for it. The class that you had taken the year before is tasked with taking your code and adopting it for a new purpose. They file bug reports on your code, and your commits fix their bugs and contain tests. This allows you to **learn how to do maintence, interact with a bug tracker, and write tests.** For the younger students, it allows them to figure out how to write good bug reports, and interact with other coders.

Conclusion I think the really important part is that your code doesn't die. You write code in a class, and it is used by other people, or it is kept and brought back up later. This you as a student to reach the "aha" moments where you see how much you have learned in the past year, by how much your old code sucks. It provides a lot more knowledge of useful tools and real workflow. **Without too much effort, it makes the educational nature of college more valuable and more realistic.**

I would be interested to hear people's thoughts. If you got a degree, did they do anything similar to this? Are you using up tools and practices?

As a bonus, I think that a lot of the parts missing in universities are missing in good real world software shops. There are a lot of software houses that don't use version control, write tests, or use a bug tracker. This strikes me as crazy.

Update (11-11 13:20) A lot of people in the comments have said that computer science is more about math and algorithms, and that it shouldn't teach you these things. Most people who take this stand say that you shouldn't be teaching programming at all, and it should be a more math based education. I agree with that point of view, but that isn't how CS is taught these days.

CS students are doing a lot of programming, and performing tasks that could be made better with these use of tools. **I am simply arguing that if you're going to be teaching programming to CS students, you should also teach them the best practices and tools associated with that craft.** It would only take 1 or 2 classes out of a CS curriculum full of theory and math based classes.

Large Problems in Django, Mostly Solved: APIs

This is the third part of my Large Problems Series. The first two were Search and Database Migrations.

A lot of efforts have come and gone in the Django space, trying to provide a API's that do various things. Some have tried to give you automatic CRUD based off your models or by abstracting the admin, others have extended Django serializers to provide some kind of functionality, and there have been lots of other approaches.

I think that Piston hits a sweet spot for creating APIs. It has a lot of nice little features, and handles the general use case well. It is also abstract enough that it allows you to provide your own layer on top of it with ease.

Piston

Piston has three major philosophical concepts that are important; Resources, Handlers, and Emitters. A Resource is the "thing" that you are trying to represent in your API, the domain object. This could be a blog post, a user, or anything else. A Handler is how you do something with that resource. It is a lot like a view, where you get the request and it delegates to different functions based on what you want to do (create, update, read) with it. The Handler will return some kind of object, and the Emitter's job is to output this. It is where you choose the format (xml, json, yaml) and other information about how the data is returned.

The way these things are abstracted makes it really easy to create a REST API. In fact, the documentation has a full working example.

I would like to talk about some of the nicer features and abilities of Piston. **This is not a tutorial, but more a pointer, so that you know it exists and kicks ass.** The Piston Documentation is decent in regard to getting you going.

Useful Features

Authentication: OAuth, Basic Auth I have found OAuth something of a pain to implement when I tried to do it on my sites. Piston handles this for you, and does a good job of it! This gives you a really nice authentication scheme for your API users for free. If you need something simpler, HTTP Basic Auth is provided out of the box as well. The Authentication is also tied in automatically to the Django Authentication scheme, making this relatively hard problem of API's incredibly simple. This gives you both ranges of Authentication mechanisms, simple and advanced, without touching a line of code.

Automatically handles different serialization formats Out of the box you also get serializers for JSON, YAML, Python Pickle, XML, and Django's own model serialization format. By default, if you append ?format=X to a URL of a piston resource, it will automatically return the data in that format. Thinking about serialization formats is basically non-existent.

Guides you towards proper REST practices Piston by default and convention returns the correct status codes for events. It even has a convinent `rc` module that maps response codes to names, to make it super simple to know what you want to return. You have to try to not follow proper REST convention.

```
if not request.user == post.author:
    return rc.FORBIDDEN # returns HTTP 401
```

API isn't tied to models Tying your API to your models seems like a good idea at first. However, you quickly want to return objects from other models, results of methods, and other data that isn't related to your model. Piston to start out lets you define a model to tie it to, but this simply sets sane defaults for the handler methods. Once you override these methods, the fact that parts of the handler is tied to a model doesn't matter. You can keep providing the basic parts that you don't want to write, but extend where you need more advanced functionality.

Lots more, built in Piston has just a ton of really useful things that you need built in, and well configured. Among the things that I haven't mentioned, but that you will appreciate:

- Throttling (by view, user, or IP)
- Streaming Responses
- Form Validation (using Django's form library)
- Generated Documentation (allowing you to document the methods you have available)

Conclusion Piston is just incredibly well configured by default. You can write a couple of lines of code, and most of the features that you expect in an API are there for you. However, this doesn't mean that it is limiting you from doing complex things as well. All of the important bits are sanely configured, but easily pluggable. It is a really amazing piece of work when you dig down into it and realize that most things you want to change are simple.

Like Django and Python, **piston makes doing the correct thing simple and obvious.** If you end up fighting against the app, you're more than likely doing something wrong.

I know I didn't get all the way into piston, and it is amazingly well written. There are lots of little niceties hiding in dark corners, instead of demons. I would love to see this project get more attention, tutorials, and blog posts in the community. Are you using Piston? Is there something you love or hate? Let me know, or even better, blog about it!

The importance of striving for awesome.

When I was about to graduate from college, I was often asked what I would be doing with the rest of my life. This is a usual question that is asked of graduates and I have very rarely heard it answered to satisfaction. Upon being asked this for the 42nd time, I decided on my response.."Something Awesome"

I know this is a simple answer and decidedly nondescript. I think it is a powerful answer in the philosophy that it entails. A lot of people are described by their job, and that is what they live for. I really don't want that to be me. I want my job to describe who I am, an extension of the things that I care about.

"Something awesome" is an incredibly powerful, positive, and motivating prospect. Everyone wants to be doing something awesome. Why allow yourself not to? I think that is just unacceptable...

I want people to call me out when I'm sucking. When you set the bar awesome high then you really can evaluate where you are going. I think striving for the moon and getting to the clouds is a much better result than striving for the roof and waiting for an elevator.

There was an interesting tidbit that I read about the value of aiming high. Aiming high isn't the norm. Most people are busy trampling over each other to compete for mediocrity and the truly awesome stuff never gets done. Stop worrying about how you're going to do something, simply strive to be awesome and settle for nothing less.

Django Testing Code Coverage

As part of the summer of code 2009, Django test coverage has been developed. I mentored Kevin Kusabik, who developed the code. It is hopefully going to be merged in 1.2, but there are still a few issues to be worked out in the implementation. That said, it currently works, and provides a nice introspective view of your code. This post will tell you how to run coverage on your code base.

It should be noted that having code coverage is a good way to look into your code, but doesn't guarantee that there are no bugs. Ned Batchelder's Pycon talk last year is a good introduction to coverage. We are using his Coverage.py module in this example to produce the coverage output in Django.

I have taken the commits from the Summer of Code and put them in a Branch on github. You will want to clone this and put it on your PYTHONPATH as your django module. If you are already using the github mirror, simply add me as a remote and pull down the coverage branch.

```
git remote add ericholscher git://github.com/ericholscher/django
git fetch ericholscher
git co -b coverage ericholscher/coverage
```

Once you have the code, you simple run your tests in the normal manner. However, now have the added options of `--coverage` and `--report`. If you run the test command with just `--coverage`, it will generate a text based coverage report. If you also specify the `--report` option, it will output a HTML report in the current directory. The HTML report is where most of the value of coverage comes from, allowing you to see what lines were covered and missed. Here is an example HTML report, showing the Django source code's coverage.

One of the major problems with coverage is that it slows down running tests by a non-trivial amount. For every instruction executed, there must be a record made. With coverage.py 3.0, this extension is written in C for speed, however it still noticeably slows down test speed.

I hope that you give it a try and enjoy the results. I'll be spending some time over the next week or 2 cleaning up the code and trying to get it into shape for inclusion in Django.

You should stay for the sprints

At most open source conferences, a lot of attention is given to the talks. At the ones that I have been to (Djangocon and Pycon), the most fun that I have had, and the most I have learned is during the sprints. I want to talk about the value and importance of staying for the sprints at a conference.

First off, lets talk about why you are going to the conference. I am going to assume that you are a developer, interested in the technology, and passionate. So the main reason that you officially go to a conference is **to learn**. Open Source conference talks are amazingly tech heavy, and the knowledge transferred in the halls is vast. A speaker will give you a really great idea, insight into a problem, and other priceless knowledge.

Then you go to another talk. You are inundated with other amazingly new and interesting ideas. At the end of the day, your brain is saturated, and then the dinner and the nights happen. The unofficial reason to go to a conference is **networking**. Networking is such a crappy term though, I view it more as **drinking and making friends**. I now consider lots of ridiculously smart people friends from late night conference experiences.

So now the conference is coming to an end, you have learned a ton, and you have met a ton of great new people. It all seems so short, and you're sad that you have to go home. **WRONG**. This is precisely the time that you want to stay and enjoy things more! That's what Sprints are for.

Sprints allow you to solidify the friendships and knowledge that you learned during the conference. You get to spend 2(-4) days of working with these awesome people, on problems that you care about, **in person**. You can take all of the knowledge that you have gained, all the ideas that you have had, and put **pen to paper**.

The guy that gave the talk that inspired your idea, when you get stuck, is sitting across the table from you. The people that you took shots of whiskey with 3 nights ago are helping you debug something in the Django Admin. You are helping your new crazy friends conquer the concurrent turtle conundrum rife with GILs. You are absorbing the vibrant energy that emerges from rooms full of motivated, passionate, people **getting shit done**.

I really can't speak highly enough about the value of sprints. There is just so much goodness and uniqueness about them. I try my hardest to stay for them at any conference that I go to.

Do you have any awesome sprint stories? Something that I have forgotten to mention that just blows you away? I find that sprints aren't talked about very much, but I think they are one of my favorite parts of being a member of an open source community.

Announcing Kong: A server description and deployment testing tool

At work we have to manage a ton of Django based sites. Just for our World Company sites, we have over 50 different settings files, and this doesn't take into account the sites that we host for other clients. At this size it becomes basically

impossible to test each site in a browser when you push things to production. To solve this problem I have written a very basic server description tool. This allows you to describe sites (settings file, python path, url, etc.) and servers.

You can see a basic version running for my personal site. It is super barebones, but it should give you an idea of what exactly is possible.

The source is available on Github. A 0.1 release will be uploaded to Pypi soon, after a few of the blemishes have been worked out. I would like to thank Nathan Borror for the design parts that are pretty :)

What does it do?

Figure 2.7: Admin

On top of this base, I have written a way to run tests against these sites. You can categorize the sites by the type of site they are (We have Marketplace, ported Ellington, and old Ellington sites). This allows you to run tests against different types of sites. You may also have custom applications that run on only one or two certain domains. You can specify specific sites for tests to be run against as well.

The tests are written in Twill, which is a simple Python DSL for testing. Twill was chosen because it is really simple, and does functional testing well. The twill tests are actually rendered as Django templates, so you get the site that you are testing against in the context. A simple example that tests the front page of a site is as follows:

```
go {{ site.url }}
code 200
find "Latest News"
```

This simply loads the Site's front page, checks that the status code was 200, and checks that the string Latest News is on that page. The arguments to find are actually a regex, allowing for lots of power in checking for content.

This then gives you the ability to view all of the results for your tests in a web interface. Below is an example of the live view that I see when looking at our servers. We have only just started using Kong, but the tests it provides are really useful to make sure that functionality works after a deployment.

You can also see the history of a test on a site. Currently it shows the last 15 results, but paginating this page will be easy. It allows you to see if your test has been running well over time. Another nice thing is that it measures the Duration of the test, so that you can see if it is going slow or fast.

As you can see, the data display is really basic. It will be improved, but currently its basically the "simplest thing that could possibly work".

Using it yourself

When we deploy code changes, I generally run the Kong tests against our sites, making sure that things work. When we launch something new, I will write a kong test to exercise it across all sites. The tests usually take a minute to write, and save lots of time and heart ache, knowing all the sites work.

At the moment the tests can be kicked off by a django management command. The `check_sites` command will allow you to run all of the tests for a given Type or Test. Allowing you to run all of the Ellington tests across all sites, or just run one test across all sites.

```
django-admin.py check_sites --type ellington
django-admin.py check_sites --test test-front-page
```

We currently have this wired up to a cron job that runs every 10 minutes. If you set the `KONG_MAIL_MANAGERS` settings to True, it will send an email to the site managers on a test failure. At some point in the future, I will be

integrating Kong into Nagios, so that Nagios will handle the running and alerting of errors. That is eventually the way that it will be run.

There are a lot of ways that this can be improved, however in it's current state it works for me. I figured releasing it will allow anyone who needs something like this to be able to use it. There is no documentation or tests, which will be fixed soon! The web display can also be improved a ton, and that is a high priority as well.

Let me know if you have any *constructive* criticism, or questions. There are a couple other little nuggets hidden in the source, so poke through if you want. Otherwise I'll write up some proper docs soon, so that you can use it.

Finding Missing Indexes That Django Wants (Postgres)

On Monday at work, our sites started to slow to a crawl. We looked to diagnose the problem, and found that the database server had a load of 10, and was struggling to keep up with the morning rush of traffic. After EXPLAINing the slow queries from the slow query log, we noticed that a lot of sequence scans were happening. This shouldn't be happening because these queries should have indexes on them. We realized somewhere in the porting process that we had lost a bunch of indexes.

Check for missing indexes

So I went ahead and wrote a little script that basically diffs the current indexes and the ones proposed by Django. This allows you to see the indexes that you are missing. This will only work on Postgres, however if you parse the indexes for your DB it should work there.

You can simply copy that file into your management commands. Then you can run `django-admin.py check_indexes`, and it will output a tuple of table and name. If you pass in the `--show` option, it will actually output the CREATE statements that create the indexes. This allows you to create the indexes in your DB by piping it in.

```
django-admin.py check_indexes --show | django-admin.py dbshell
```

You want LIKE, fast queries?

In our search, Frank and James also discovered that when you have a UTF8 database (which you should), Postgres needs a special index to do LIKE queries against text fields. James filed a Django bug with details. However, if you are running a postgres database, it may be worthwhile to look for places that you might be making similar queries. For more information check out the postgres docs

We'll do it LIVE

One other postgres index optimization that James and Frank discovered was that Postgres gives you the ability to index on state of a field. So if you have tables that have any kind of status that is often queried, you can set a specific index on that.

```
create index "published_story" on "news_story" ("status") where "status" = 1;
```

I hope that my little script and these tips allow you to make your Postgres Database purr. I only just got schooled in Postgres recently. Frank has been doing this a long time and has some awesome postgres performance tips. I recommend reading through that if you really want to make your database run well.

Note: It has been pointed out that South uses a different naming scheme, so if you have indexes created with south, this may not work quite right.

Writing Code with Designers

When working on side projects, usually you wear all of the hats. Sysadmin, developer, designer, marketing, etc. You have to do all of them, and presumably you do one or two of them well, and the others well enough to get by. Working at the Journal World has been the first time that I have worked with real designers, and it has been a learning experience.

I think it would be interesting to see a talk or a panel at Djangocon this upcoming year about how to write code with designers. There are a lot of little things that allow them to work with your code better. A couple things that I can think of off the top of my head are:

- How to structure data for templates
- How to write templates that are easy to mark up, with the least amount of effort
- Writing good template tags and filters

I think that there is also a design process, where it really helps if you include the designer in the process of scheming functionality. I really find that designers have a better handle on asking the 'what does the user do here?' and 'what does this actually accomplish?' questions. Working with them on projects is a really interesting difference than working by yourself on things.

Do you have designers that you work with, and does it help your program design? Do you just have other programmers to bounce ideas of off? Does it work in a similar way?

I find that this workflow is really beneficial to all involved. It is the first time I have had the luxury of it, and was wondering if other people have this process as well.

Large Problems in Django, Mostly Solved: Search

It's been a little over a year since I started doing Django development full-time, for one of them real jobs. Around that time, there were a few large problems in the community that hadn't been solved yet. They were kind of blemishes when you would talk to people about Django, and I'm happy that most of them have been solved.

This will be a series of posts that talk about the different big problems that have been solved, and how they have been addressed in the community.

Search

Search was probably the biggest annoyance for Django. It is something that every site needs, and something that just wasn't really happening in the Django community. 2008's Summer Of Code ended with djangosearch as a half-finished shell, that needed to be better architected; but it did show a good push in the realm of search.

Out of those ashes, comes an awesome solution to the search problem in Django. Haystack is something I am more familiar with (we use it in production at work), and is the brain child of the ever modest, house rocking Daniel Lindsley. **It provides a number of Django patterns applied to search**, which makes it easier to internalize.

Note: Another approach to search is available, which patches django's ORM. This uses the existing full-text search in your database.

Useful Haystack Patterns

Registration The registration pattern of the admin allows you to unobtrusively make models searchable (including code you don't have access to). This allows you to register Django Comments as searchable for example, without forking the code base. This will look similar to the admin:

```
from haystack import site
site.register(Note, NoteIndex)
```

Search Querysets Haystack provides an interface familiar to the Django ORM Queryset API. This gives you most of the commonly used functions from the ORM, but allowing you to use them on searches!

```
unfriendly_results = SearchQuerySet().exclude(content='hello').filter(content='world')
unfriendly_results.order_by('-pub_date')[:5]
```

It also gives you Search specific methods such as boost and facet.

Class Based Views In 1.2, hopefully generic views will be class based. Haystack has an implementation of these as well. Like any other kind of class, it provides the easy ability to override functionality through subclassing.

This (simplified) example from the source shows how easy it is to provide extra context to a search view.

```
class FacetedSearchView(SearchView):
    def extra_context(self):
        extra = super(FacetedSearchView, self).extra_context()
        extra['facets'] = self.results.facet_counts()
        return extra
```

Pluggable Backends Haystack currently supports three different search backends: Whoosh, Xapian, and Solr. With a publicly documented backend API, you can enjoy all of the power of the search engine of your choice, by providing a backend for it!

```
HAYSTACK_SEARCH_ENGINE = 'solr'
```

Documentation A shining light in the Django world is the documentation. It is often talked about as being the biggest factor for how people learn Django and love it is the documentation. Haystack is another package with fantastic documentation. Here are a couple of little gems that really show the quality and thought that has been put into them:

- 7 step tutorial
- Debugging Haystack
- Best Practices
- Reference

The docs cover ways to improve your search and make it awesome, as well as just helping you get the software set up and running. The information is invaluable, and will help you make the search on your site really great!

Are you using Haystack? If I'm preaching to the choir and you already use Haystack, there is a growing list of users that are using haystack. Daniel would love for you to contact him, and get yourself added to the list.

Did I miss anything? Let me know what you love (or hate) about Haystack. I think it reuses a lot of the good patterns in Django, allowing people to take knowledge they already have, and apply it to a new problem domain easily. Is there anything that you don't like, or something that you love that I missed? Let me know in the comments!

Correct way to handle default model fields.

With Kong, I have been trying to figure out a way to provide overridden model defaults. At work, our pythonpath's default to `/home/code`, however your setup is probably different. It would be useful if there was a simple way to let you override the defaults for your Kong installation.

```
pythonpath = models.CharField(max_length=255, default="/home/code")
```

I received a pull request that put a `KONG_PYTHONPATH_DEFAULT` setting, which would be read in as the default. However, it seems like this doesn't scale particularly well, and would be annoying if you have 5-10 fields to make defaults for.

So I thought up a couple of different approaches to this problem, and am curious if people have input, or a better way to solve this.

One Big Setting

This would allow for a setting, but it would only be one setting for all of the kong defaults. I'm thinking about a dict, where the key is the model_field name, or just the field name. The key would obviously be the default (or a callable that returns the default).

```
KONG_DEFAULTS = {
    'servername': 'ljworld.com',
    'pythonpath': '/home/code',
}
```

This would keep things in the settings, but not cause a huge amount of settings bloat.

Specify a place to hold your defaults

Specify a setting along the lines of `KONG_DEFAULT_PATH`, which would be a module on the pythonpath. I would import this module and then try and pull the name of defaults from there. I would provide a sane default in Kong for this, that would be an example of how to do it.

```
KONG_DEFAULT_PATH = 'kong.defaults'

#kong/defaults.py
pythonpath = '/home/code'
servername = 'ljworld.com'
```

So you could set `KONG_DEFALUT_PATH`, and then redefine the values there. This is basically defining a convention for setting the default values on a model.

However, this is basically the same problem/solution as some kind of application specific settings. I really think this would be valuable, allowing reusable apps to specify default settings, and then letting users override them

Storing it in the database

I could create a super simple model that would hold defaults for my fields. This would allow the user to set the defaults in the database, and then I could pass a callable to the default, which would check for the existence of the model in the DB. This doesn't seem like a very good option, but would at least allow for configurable changes without touching the code.

I'm probably DOIN' IT WRONG

It seems like a subset of a larger problem, which is that it isn't easy to define application specific information. That is an ongoing Django problem, without a good solution. I am imagining a third party application that might make this process easier. Does anyone have a good solution to this?

Class Based Template Tags

The problem

In Django, template tags currently are separated between a Node class and a "parsing function". The parsing function takes the tag, represented as a string, parses the input, and passes the correct arguments to a Node class. The Node class then does whatever rendering it does, or updating of the context, and then renders itself in a form suitable for the template.

This is mainly by convention that there is a separation here between the parsing and the Node. As I see it, there is no particular reason that the Tag can't be responsible for the parsing and rendering itself. A lot of the time I find the parsing function and the Node separated by hundreds of lines in a file, making it hard to understand.

The proposed solution

We can combine the parsing and rendering of a node in a similar way in something I call Class Based Template Tags. This allows the template tag to be able to parse and render itself.

I have an example in my playground over at github. They are based around a lot of the ideas in django-template-utils. Specifically, this example will be recreating the get_latest_objects tag from that package.

```
class ClassBasedTag(template.Node):
    """
    Tag that combined parsing and rendering

    Subclasses should define ``render_content()`` and ``parse_content()``.
    """

    def __call__(self, parser, token):
        self.token = token
        self.parser = parser
        return self

    def render(self, context):
        self.context = context
        self.parsed = self.parse_content(self.parser, self.token)
        return self.render_content(context)

    def parse_content(self, parser, token):
        """
        This is called to parse the incoming context.

        It's return value will be set to self.parsed
        """
        raise NotImplementedError

    def render_content(self, context):
        """
        This is called to return a node to the template.
```

```
        It should return set things in the context or return
        whatever representation is appropriate for the template.
        """
        raise NotImplementedError
```

As you can see, this tag combined the concepts of Parsing and Rendering a tag into the same place. The `parse_content` and `render_content` are equivalent to the current Django way of doing a parsing function, and Node class render function. Currently the render function depends on self.parsed being there, and not being passed in, this is to keep the function arguments the same as previous render functions. The code isn't meant to be production quality, more of a proof of concept.

A couple of gains are made from combining things together. First of all is the fact that the code is right next to each other, as mentioned earlier. However, it also allows you to subclass these classes, and provide functionality that makes people's lives easier. Having the rendering and parsing in the same class also allows for some trickery with passing around data, like mentioned, which may be a good or a bad thing.

Let's go ahead and show an example of an implementation of this type of tag.

```
class GetContentTag(ClassBasedTag):

    def parse_content(self, parser, token):
        bits = token.contents.split()
        return (bits[1], 1, bits[3])

    def render_content(self, context):
        model, pk, varname = self.parsed
        self.pk = template.Variable(pk)
        self.varname = varname
        self.model = get_model(*model.split('.'))
        context[self.varname] = self.model._default_manager.get(pk=self.pk.resolve(context))

register.tag('get_latest_content', GetContentTag())
```

This tag is used in the following manner:

```
{% get_latest_content news.story as latest_story %}
```

As you can see, I think it makes it nice and concise to be able to have the parsing and the rendering of a tag right there in the same place.

This code is a very simplified use case for the idea. It is basically the simplest possible thing that could work. I will expand on the ways that this idea gives us a lot of power and flexibility over our Template Tags in the future, but I think this idea stands well on it's own.

Making Template Tag Parsing Easier

In my previous post about template tags, I discussed the two steps required for template tags. Today I will be focusing on Parsing of template tags, and how they may be improved in the framework of Class Based Template Tags from yesterday. I have talked about problems with template parsing in the past as well. This post will offer 2 different approaches to making parsing better.

I would like to thank Cody and Chris who were involved in a slightly drunken conversation that led to these tags. Chris actually wrote the other neat parsing implementation that I will talk about today. Cody wrote the underpinnings of that implementation as well.

Note: Both of these approaches are more Proof of Concepts, and the code probably shows. Please don't knock implementations, and just think about the ideas housed within.

Parsing from above - A DSL approach

I'm going to go ahead and start talking about an approach to parsing template tags that was pointed out in yesterday's comments. It takes surlex which is made for easily parsing URL's, and applies it to the concept of parsing template tags.

In the tag_utils package, I looked at the tests, because they make great documentation. Here is an example of a tag definition.

```
p = ParsedNode('test', '<arg1:int> <arg2:string> <kw:kwarg>', test_expected)
register.tag('test', p)
```

This defines a tag called test, which parses an int, string, and kwarg from a surlex expression. The third argument is a function that is executed on the arguments on rendering.

This allows you in your test_expected function, to act on the arguments that are defined inside of the surlex expression. A trivial example of the test_expected function is:

```
def test_expected(context, arg1, arg2, kw=None):
    print "Got %s and %s" % (arg1, arg2)
```

So if you called the tag `{% test 1 racoon %}`, it would print out `Got 1 and racoon`.

This is an interesting way to provide a sort of DSL on top of the current mess that is parsing of template tags. I really like how it reuses Surlex, which was made for parsing URLs. However, parsing template tags is a similar task, and it works well here too!

I could imagine this easily being bolted on to the approach from yesterday, which might allow for easier subclassing and reuse of the parsing functions.

Parsing based on keywords

An approach that I have talked about in the past is basically a subset of the above idea. It allows you to define kwarg type arguments for your tags, and have them magically parsed out for you. An example of this is my own SelfParsingTag. The following lines allow you to specify what arguments your tag will accept.

```
def __init__(self, required_tags=[]):
    if not required_tags:
        self.required_tags = self._get_tags()
    else:
        self.required_tags = required_tags

def _get_tags(self):
    return []
```

So you can either define the `_get_tags` function, or pass the allowed tags into the call when you make the tag. The following 2 bits of code are equivalent.

```
class GetContentTag(SelfParsingNode):
    def _get_tags(self):
        return ['as', 'for', 'limit']
register.tag('get_latest_content', GetContentTag())

#Is the same as the following:

class GetContentTag(SelfParsingNode):
    pass
register.tag('get_latest_content', GetContentTag(['as', 'for', 'limit']))
```

Once the Tag knows what it arguments it will be accepting, it parses them.

```
def parse_content(self, parser, token):
    parsed = parse_ttag(token, self.required_tags)
    for tag, val in parsed.items():
        setattr(self, '_' + tag, val)
    return parsed
```

This effectively sets a private varible on the tag to the value of the arg. So for example, if the tag was called `{% sweet_tag for news.story as my_stories limit 10 %}`, then `self._for` would equal `news.story`, and so on. It also returns the parsed values as a dictionary. There are a lot of improvements that could be made to `parse_ttag`, but it works as a basic implementation.

This approach allows us to implement a tag really easily. If you want a (silly) tag that just updated the context with whatever value you input, you could make a simple tag. It would be used `{% my_tag with "awesome text" as context_var %}`

```
class SimpleContextTag(SelfParsingTag):
    def _get_tags(self):
        return ['with', 'as']

    def render_content(self, tags, context):
        for tag in self.required_tags:
            context.update({tag['as']: tags['with']})

register.tag('my_tag', SimpleContextTag())
```

To implement the get_latest_object code from yesterday, we can skip all of the parsing steps.

```
class GetContentTag(SelfParsingTag):
    def _get_tags(self):
        return ['as', 'for', 'limit']

    def render_content(self, context):
        self.model = get_model(*self._for.split('.'))
        if self.model is None:
            raise template.TemplateSyntaxError("Generic content tag got invalid model: %s" % model)
        query_set = self.model._default_manager.all()
        context[self._as] = list(query_set[:self._limit])

register.tag('get_latest_object', GetContentTag())
```

Which is better?

To be truthful, I like the Surlex approach better than my own. It seems to have a lot of the benefits of mine, but with added flexibility. However, that does come with the implementation being a bit more complex. It brings some really neat ideas forward about how template tags might be handled differently. It allows for optional arguments, does basic type checking (based on it's regex nature), and ensures that the order of the arguments is the same.

I could imagine some kind of dispatch based template tag scheme that has a list of URLs, basically like the URLConf and view structure. I think that this problem has a lot more depth to it, and hopefully by pointing out a couple of different ways of solving it, and looking at it, we can improve the situation.

Adding testing to pip

Python packaging has been in a bit of a state of disarray for as long as I've been using it. Pip has come along to make installing python packages easier. It has a lot of features that are useful, but they have been talked about in many other

blog posts.

Today I want to talk about adding testing to pip. If you are familiar with the Perl community, then you probably know about CPAN. It is basically Pypi for Perl. They have a command, called cpan, which allows you to install packages in a similar way to pip.

One of the steps that a package goes through before being installed on your system is that the tests are run. This allows you to know if the package that you have installed is actually going to work on your system. It may be broken on your platform, or you may be missing a library that it thought you had. Currently, pip has no way to test packages when they are being installed. I went looking for a way to make that happen.

It should be noted that pip is based on setuptools. Setuptools is what parses and understands most of the logic inside of your setup() function in the setup.py for your project (which you have, right?). Setuptools has an option called `test_suite`, which allows you to run setup.py test on your package, and have it run the unit tests. This is done by calling whatever python function is defined in `test_suite`.

I added the ability for pip to run `setup.py test` on a package that it is installing. It is executed by running `pip install --test <package>`. The implementation is on a ticket on bitbucket, and in a repository on github.

If you want to check it out, go ahead and clone my repo and check out the test_command branch. Then you can simply run

```
python pip.py install --test wsgiref
```

for an example package. If a package doesn't have a `test_suite`, then it simply doesn't run anything.

Note that if the tests fail, it doesn't impact the installation of the package. The python community's tests aren't quite good enough,and almost any Django package you try this on will not have any tests. I wrote about how to add testing to your django package, but the process is long and involved. I'm working to improve the situation for Django and hopefully having the ability to run tests in the package management tool will spur people to add testing ability to their setup scripts!

Nose makes this really easy, by simply adding `test_suite = 'nose.collector'` to your setup.py, nose will run your tests correctly. This is the level of support that I am hoping to implement for Django.

On a side node, I talked to Ian Bicking about this, and he suggested writing the test command as a separate command, so you would be able to do `pip test wsgiref`, if it was installed. This has some other problems, which I will talk about after I have implemented this functionality.

I would love to hear feedback, or if anyone has ideas for improving testing in the python and django communities. I have lots of ideas, and I will be writing more of them up over the following weeks.

Large Problems in Django, Mostly Solved: Database Migrations

Continuing in the series of big problems that are mostly solved, we have database migrations. A couple days ago I talked about Search.

Database Migrations

Database Migrations are an interesting piece of the Django community. Rails has the functionality built in, but Django currently relies on third party apps for this functionality. One of the core philosophies about not including apps in the Django core is that ideas percolate better in the fast release environment outside of the core. When something goes into core, it is automatically seen as blessed, and will certainly become the defacto answer to a problem. Leaving things outside allows multiple different implementations to develop (as they did), and for one to become the standard (which it has). Along the way it has picked up ideas from others, and now provides a good answer to migrations.

South South has emerged as the obvious choice for database migrations in the Django community. We use it in production at work at the Journal World, and it has served us well.

I have talked about south in the past, using it to migrate test fixtures. This serves as a basic tutorial and introduction into south as well.

Main Features

Automatic Migrations Most of the migrations that I write, I don't write a single line of code. South has the ability to how you model looked at the end of your last migration, and then extrapolate what has changed (in most simple and modestly complex cases). There are obviously times that it falls down, but for simple addition, deletion, and modification of fields it has worked almost flawlessly for me. With a simple command, it will do all your work for you.

```
django-admin.py migrate app_name --auto
```

It has problems with Generic Foreign Keys and a couple of other more complex models. However, I would say that it absolutely nails the 80% case that most migrations fall in to.

Fake ORM ("ORM Freezing") This is a feature that South has grown from it's Migratory roots. I think it is one of the best conceptual features for migrations. It allows you to use a Fake ORM (the real ORM, applied to the aforementioned fake models), to do data transformation in your migrations. This example from the tutorial shows the value:

```
def forwards(self, orm):
    for adopter in orm.Adopter.objects.all():
        try:
            adopter.first_name, adopter.last_name = adopter.name.split(" ", 1)
        except ValueError:
            adopter.first_name, adopter.last_name = adopter.name, ""
        adopter.save()
```

Database Independent This sounds like an obvious feature, but a lot of the approaches for migrations were only viable on one database. The support for SQLite is still lacking, but that is because of fundamental limitations in the way SQLite works. Most people using SQLite can just wipe their database and start over, if not, you should probably be using another database.

It knows when you've been naughty South keeps track of all the migrations that you have run, and it intelligently informs you if you have missed a migration. It also supports inter-dependencies on migrations. This allows you to be safe in your knowledge that your migrations will be run properly, and that state is maintained. This sounds like a hand-wavey feature, but when you're migrating your data, knowing when things aren't quite right is a nice feeling!

South also keeps track of the migrations that are on disk, and won't let you migrate if they are different than previous runs. This makes sure that you aren't running against a different version of the code; allowing you to be sure that the migrations being run are correct.

Conclusion Overall, south solves a lot of the problems about migrations in a good way. There have been other solutions to the migration problem, and I think that south has taken most of the good ideas and combined them in one place. It has some drawbacks still, but overall it is the best of breed in Django for Database Migrations. If you are looking for a migration tool for Django, this is your best bet.

There aren't a lot of flashy features in the migration realm I feel. Mostly you just want something that keeps your data safe, and allows you to write migrations as simply and foolproof as possible. South lets you do that, so I consider it a win.

I view migrations somewhere along the lines of testing. It is one of those things that once you have, you don't see how you ever lived without it. Being able to immediately see the state of your database, what migrations haven't been run, and what all needs to happen is incredibly useful. Having a safety net of repeatable migrations also ensures that your databases are all the same, across many installations and machines. The value of database migrations are many, and South brings them to you in a nice package.

Correct way to handle mobile browsers

At work, a lot of our sites have sweet mobile versions. The problem is how to educate people of their existence. Currently we just have little ads that show up on the site that promote the mobile site, which seems a subpar solution. So I was tasked with doing providing a way to redirect to the mobile sites. Luckily, as a lot of the time with Django, most of my work was done for me.

Minidetector is a Django reusable app that allows you to know if a request is being viewed on a mobile device. It provides a middleware and a view decorator that sets a request.mobile variable to True if the request is coming from a mobile device. It's method of figuring out if a device is mobile is simple; It first checks for a special Opera Mini header, then for WAP support, then finally checks the User Agent against a list of known mobile strings.

So at work I have implemented a simple way to promote the mobile sites through redirecting, allowing for a couple of different use cases. This has lead to a problem that a lot of internet sites face, and I haven't found a good solution to the problem: **how do I redirect users to a mobile site?**

Obviously, **you should keep the request path**, so that when you go to SITE/blog/2009, you get redirected to m.SITE/blog/2009. A lot of sites actually chop off the request path, bringing you to the mobile home page!

The use case

The use case I am thinking about is a user that is using twitter, and they click on a lot of links to a site, through a mobile browser. They should be gentley introduced to the existance of the mobile site, and have the ability to always have mobile links go to the mobile site. However, they should also have the ability to say 'never show me the mobile site' as well.

Three approaches

No Redirects I see two basic approaches to the problem. The first is that we don't automatically redirect anyone to our mobile sites. We are able to detect if they are identifying as a mobile browser, so we can show them a message about our mobile site, and let them choose.

An option could be made to allow a user to say "Always redirect me" if they enjoy usage of the mobile site. This seems to allow the user to get expected behavior, but allow them to choose to use the mobile site on their mobile device if they want. However, you run into the problem of users ignoring the message about the mobile site, or just not caring enough to click it.

Redirect once (opt in) Redirect once is the plan where you redirect the user once, and then set a cookie to never redirect them again. This allows the mobile user to get a glimpse of your mobile site the first time they visit, and can then choose to visit in the future.

You can also allow them to set a cookie to automatically redirect all of their mobile requests in the future. This allows the user to get a glimpse of the mobile site, and see if they want to use it. Then based on this experience, they can choose to visit it by default if they want.

Always redirect (opt out) The third option is to always redirect mobile browsers to the mobile site, with the ability to go back to the main site. You would have a setting that the user could set to never be redirected. This is more of a 'all mobile users will use our mobile site, unless they choose not to'. I don't know if the mobile web is quite there yet (for example, we don't have a mobile version for every page), and it might lead to user confusion.

What do you think? I think that redirecting the user on their first visit on a mobile browser is a good idea. This introduces them to the mobile site, and by setting a cookie on that redirect, you can be sure that they won't be redirected again. Then you can have an opt in cookie, that basically says redirect me every time. This makes it do what people expect most of the time, while still allowing the choice to always be redirected.

Have you implemented mobile redirecting before on a site? How have you solves this problem? Am I missing some obvious solution that handles all these cases gracefully?

> **Warning:** Everything past here is from college. I was living in a bit of a different world back then, so buyer beware.

2.3.6 2008

America...*sigh*

A picture is worth a thousand (horribly said) words :)

via Five things i saw in america which freaked out a canadian

Hackers and Painters

Just finished reading Hackers and Painters by Paul Graham. It was an amazing book about the past, present, and future of computers. Lots of stuff about programming, but also fulfilling for people that don't know much about computers as well looking outside in. It explains a lot and is an amazing read. Paul Graham is an amazing Essayist and it shines through in this book. It contains 15 unrelated essays, and is highly recommended.

Weekend

Lost the weekend and this week in web land. A friend came to visit, everyone came back from break, and classes are starting. I will begin posting regularly again. I read 3 books in the last 2 weeks, which I posted about earlier. On Intelligence is amazing and has set my mind racing, expect some good posts coming up in the next week or two based off of reflection on that.

Iowa

Tomorrow is the Democratic caucus in Iowa. I'm really hoping that Obama wins, or basically anyone but Clinton. I believe that Edwards and Obama have the best chance of actually returning this country to it's basic morals and values, and I certainly plan to vote for whichever of them wins this primary. I also believe that if Clinton wins, then it will be the first time in history that an independant could possibly win the White House. (Obama/Edwards, or Obama/Paul perhaps?). Here's hoping that faith and morality wins out in this crazy world of ours, and a chance to put America back in it's rightful place as a law-abiding and moral country.

My dad was wikipedia

For most of my life my dad has been my wikipedia. Long before it existed, anything I had a question about, I could ask him and be assured to either be given an insightful answer, or a logically thought out answer that amazingly always seemed to be correct. I think this is the basis for the constant curiosity that still exists in myself.

When you can ask why about everything and be given an answer, it only leads to more why's. An analogy to the academics dilemma. The more you know, the more you know you don't know. I had a never-ending stream of why's and a never-ending stream of answers. Looking back I find this to be one of the greatest influences in my life thus far.

The killer feature of wikipedia will be when it can understand my questions and provide insight into them. Searching and link-following are great and all, but the inherent context matching of intelligence that can be provided by a real live human is unmatched by computer.

OCR with context

OCR should use context, when it sees the word 'everythxxg', it should know that the 'xx' is 'in'. This is how the human brain works, and is how the computer should work too. If Google can suggest spelling suggestions to my misspelled words, there is no reason that this technology couldn't be applied to OCR. It would make it much more powerful and useful.

Facebook Scrapage

I'm thinking about how to implement facebook's social graph in my Events calendar application. It would be a big boon for my site if when people signed up, they could automatically have their facebook friends imported as their friends on my site. However, I don't like the idea of having to have the person give me their login information to do this. (This is what OAuth is for!).

However, facebook lets me see all of the people at Mary Washington on it, and it lets me look at all of their friends. I'm thinking that I might be able to scrape facebook automatically when someone signs up, using my account, to atleast import basic information about their friends. This isn't as easy or useful as a full API that facebook could easily provide, but it would be a start, and a cool app to write!

Facebook Update

Funny, Scoble just got banned from facebook for doing exactly what I was talking about doing. Damned social information silos..

Earthquakes in politics

An interesting opinion piece that I read at work in the NY Times today. Talks about how Obama and Huckabee both embody vastly different philosophies of government than the previous established order. It gives me hope to hear them discribed that way. Hopefully the existing governmental structures won't sink their hopes and dreams for this once-great country of ours. Two Earthquakes

Related, a blog I read linked to a quote from Huckabee about education that I found inspirational, showing his very interesting and worthy outlook.

From Iowa winner Mike Huckabee: "Education is only a true education if we're developing both the left and right brain of the student . . . . Take a room of 5-year-olds and give them a piece of paper and crayon and every one of them draws a picture. When he's 15 that kid won't draw the picture or sing the song. Somehow the education system beat out of him or her the creativity that was innate in that student."

OpenID FTW

This is why OpenID is such a good idea.

URL based Identifiers

It talks about how we should use URLs instead of E-mail as identifiers for ourselves. When someone gets ahold of your e-mail, they can spam the hell out of you. E-mail inherintly has no way built into it to identify who the send is, so you have to accept ALL e-mail, and then use other mechanisms to sort out the stuff you want. With URLs, if you get my URL you get nothing. You know who I am, but that doesn't inherintly give you any way to contact me. If you are my friend and you can prove to me that you own a URL that I know you own, I can then allow you to contact me. This inverts the basic premise of E-mail, allows for authenticatin, eliminates spam, and is just a good idea. I'll put my URL on my business card, and then I'll decide what information you get about me based on other critera.

This is a very powerful idea, and has the potential to change the paradigm of web interaction. That's why I'm learning about openID and such!

Cool Music Video

Never heard of a band called Battle before. They have a really neat video that I got pointed to from a really cool advertising blog

Code on Launchpad

The code for the website is now located on launchpad. I am using their bazaar version control system that is kick ass. It's a Distributed VCS which means that you can run it completely locally, without a server. I use the server of course, but it allows you to do work on your code without internet access and other neat things.

Also testing an update to the Django basic blog that should allow blog posts to show up without restarting the server (big bug!)

Books to read

Just posting some books that I want to read this upcoming semester.

Structure and interpretation of computer program (free html version): The quintessential book on computer programming. It's a classic and I feel like I should read it just to say that I have and to understand the basic theory of computing presented easily. The book opens with an amazing dedication that I would like to share.

This book is dedicated, in respect and admiration, to the spirit that lives in the computer.

''I think that it's extraordinarily important that we in computer science keep fun in computing. When it started out, it was an awful lot of fun. Of course, the paying customers got shafted every now and then, and after a while we began to take their complaints seriously. We began to feel as if we really were responsible for the successful, error-free perfect use of these machines. I don't think we are. I think we're responsible for stretching them, setting them off in new directions, and keeping fun in the house. I hope the field of computer science never loses its sense of fun. Above all, I hope we don't become missionaries. Don't feel as if you're Bible salesmen. The world has too many of those already. What you know about computing other people will learn. Don't feel as if the key to successful computing is only in your hands. What's in your hands, I think and hope, is intelligence: the ability to see the machine as more than when you were first led up to it, that you can make it more." -Alan J. Perlis (April 1, 1922-February 7, 1990)

Inspirational on many-a-level.

On Intelligence: This is an AI book that I heard about through a blog. Sounds really interesting, and is applicable to my Events website. I am trying to do some neat stuff with tagging/hierarchical structure of data, so understanding the best ways to store data (intelligence) on a computer would be good.

Hackers & Painters: This is a book by Paul Graham that I have read most of the essays from. But it looks amazing and I want to read it all in printed form. It talks about how Programmers are really artists, which is a theory I subscribe to, and hopefully that book will give me the vocabulary and authority to have that conversation with people and not have them look at me funny :)

Lucky for me the awesome UMW library has all three books. Yey!

NCUR 22

I got accepted into NCUR. The National Conference on Undergraduate Research. Here is my abstract. I get to have at least the abstract published, and maybe the entire paper that I write in support of my project, still not sure how it works. UMW is paying for me to go, which is amazing. It should be an awesome opportunity to meet some people in my field doing interesting things. The presentation there is the weekend after my Honors Project presentation that I have to do to graduate with honors. The timing works out well :)

Bill Clinton

Saw Bill Clinton speak on campus today. It was awesome! He is such a great public speaker. It's amazing to hear a politician say logical things, backed up with numbers, and actually agree with their general ideas. What a concept. He hasn't made me want to vote for Hilary Clinton over Obama (certainly the point), but if she does win over Obama, I will feel genuinely better about voting for her in the national election. Bill is an amazing speaker, and was funny and serious at the correct times. Very well done. It lasted over an hour, and the turnout was amazing. There was a line halfway down our entire campus! Very cool.

Security Vulnerabilities on the Internet

I was reading an article on LWN about security vulnerabilities on newly shipped machines. The qualm is that the same place that the updates for vulnerabilities come from is the same place where you are going to get infected. They are asking if there isn't possibly a better way to do it. I think there is:

Don't let the user use network facing services until the system is patched. When the user first gets the machine, don't let ftp/ssh/etc. connect and give them a warning that they have to update their systems before they can have access to the internet. This will keep them protected until their machine has a chance to update, with the update mechanism the only way for them to be infected.

Graduate

Was talking to an old friend today. My friend Shane who graduate from Georgia Tech with a degree in Electrical Engineering. He's currently in Cali, with a cool job somewhat related to his major. I was talking to him about the current situation (graduation approaches),and he has an amazingly apt drawing.

RocketShip

Cheers.

Job hunt

Starting the good ol' job hunt. Trying to figure out what I'm going to be doing with myself for the next couple years of my life. Big ol' decision that it is.

I'm seriously considering continuing on with my current job at CACI. It is an awesome job with awesome people. I work with good kids, and my bosses and everything else is really good. The main downside is my moral dilemma of working for the government. I'm technically working for a defense contractor, working on a portal for on-base use.

So I'm not doing anything except allowing the government to communicate with themselves better; which with my view of the inefficiencies of government, could be seen as a good goal.

I'm looking at places out in Cali, Kansas (the makes of Django!), and Australia very strongly. I want to be in a neat place, with good culture and good waves :) Too much of my life has been lived without surfing, so I need to get back to it. It's one of my true passions in life, and cannot be suppressed much longer. I also want to be in a college town, because that breeds culture and cool people. Plus I do someday want to be a teacher or at least an adjunct at a school, which requires an advanced degree. So being near a good college would allow me to go to said college easier ;)

I've been throwing lots of my old projects back up on my website, in working order. They are all Perl programs, and bring me back to my roots. It's amazing that I wrote a couple of them in high school! I feel like my college years have not been as useful in side projects, but that can be easily explained. For one, I've gotten a degree in CS! Lots of my time has been devoted to learning, and school projects that are academic in nature but important none-the-less. A couple of my updates have in fact been school projects, or independent studies, which is good. I see the progression in my skill as a coder and general "computer scientist" to be a natural and linear progression. All of the baby steps I took earlier in my career have enabled me the through understanding of the mechanisms now at my disposal.

Been trying to find time to actually update my resume. Tinkering with my old code to get it working again is good fun, resume writing is about the polar opposite. It's coming along though, and is a work in progress. It's interesting writing a resume for myself because it differs from most other people I know in fundamental ways. As my friend Jeff said: "You have marketable skills, lucky". Technical resumes are awesome in that regard. Lots of languages with lots of years of experience is a good things.

However, my main problem that I've been struggling with is how to convey my passion for the craft in my resume. I feel that it is intrinsic, and people will be able to see it. The whole getting my CompTIA Linux+ certification in high school, and doing the lyrics program to scratch an itch are good indicators. However, I can't help but feel that there are more design-oriented and literary aspects that I could use.

Upon reflection (aka writing it down..neat, knew there was a reason I did that) I presume that these feelings are more of a lack of control and understanding. My limited skills in the design and literary fields breed insecurities in my understanding of their use. I assume there are ways that I could represent myself differently if I had those skills, but I guess you can't have everything right? The always present academics dilemma: The more you learn, the more you learn you don't know...

Cheers and wish me luck, should be an interesting couple of months.

Awesome 3d

This is an amazing video of some dynamic 3d work a guy did for his PhD Thesis. Johnny Lee is a PhD student at CMU, and he modified a display to use the wiimote and a special headset to give real 3D effects. It actually changes the picture on the screen based on your proximity and angle to the screen. Amazing.

<embed src="http://www.youtube.com/v/Jd3-eiid-Uw&rel=1&border=0" type="application/x-shockwave-flash" wmode="transparent"width="425" height="355">

via zdnet

Perfect Abstraction

Here in computer science land, the quest is for the perfect abstraction. That's what our job is anyway, Software Engineer my ass, more like lead abstraction implementer. This quest for the perfect abstraction is never-ending, and certainly cannot be attained by humans. We aren't capable of creating bug-free software, so our abstractions will be inherently leaky. If it wasn't leaky, then it wouldn't be abstracted.

Today I was doing some brainstorming in class and came to the conclusion that paper is the perfect abstraction for our minds. The ideas, words, or pictures that you put on a piece of paper have meaning. However, the way that you put them down only has meaning to you. Others might be able to grasp why it is laid out the way it is, but true free form

brainstorming I feel is inherently meaningful, but the ideas cannot be translated easily. It has some abstract meaning to you, but the meaning is in the relation of the objects on the paper and in your head. Presumably if the ideas could be laid out more succinctly with words they would have been. So in the terms of abstraction perfection, I think that it is one of the few examples we have of a perfect abstraction.

Of course, this means that you assume the brain, if only we could do that in CompSci. It would make things a lot easier :) Perfect cognitive relations are out of reach for a little while longer...

Website Interface Design

I plan to design the events site through the lense of the user. The UI philosophy is thought about in that way. We don't ask how to design a page about adding an event to the calendar. We ask what the user wants to do when putting something on the calendar. What are the use cases of the calendar, why is the user there. This ties in with what makes our cal better than other cals. Trying to make the UI amazing.

Why I love the CLI

Simple example. I walked into my room today to a picture screensaver which is awesome. Aparently it uses the Pictures folder on the Desktop, of which only my latest pictures are in. I want it to use all of them...

rm Pictures ln -s /store/pics Pictures

With tab completion.

All my pictures are now there :)

Time to use that education

Okay people. Here's a proposal for you. Let's change this school of ours. For my Senior project at UMW I'm creating an "Events" calendar for the school and fredericksburg community, if you're interested in that, read below.

=========QUOTE=========

This website has a couple of different goals that I would like to tell you about.

The first and biggest goal of this website is for people to use it. I want people to enjoy using this website, and find it useful. Without that none of the other goals could possibly be accomplished.

The second goal is to provide a central repository for events that are happening in this area. When someone asks 'What is going on tonight?', I want my site to be the first thing that comes to mind. If someone asks you, it should be obvious to say to check this website. That function in our community is currently not being served, and I hope that this site becomes that very tool.

The third goal of this website is to connect the Mary Washington and Fredericksburg communities. Currently, there is very little interaction, at least on an institutional level between our great school and great local communities. I wish to foster this relationship, and allow Mary Washington students to be immersed in the great culture that surrounds them here in this great town.

=======END BS======

Sound good? Okay, that's where you come in...

We're going to college, and I hope for the right reason. We love what we do! I know that's true for me, and for others of you as well. We all have our own special talents and skills, and I have a lot of faith in my friends. I want to extend this offering to all of you to hopefully be a part of something awesome. I know this sounds cheesy, but have faith. I will make a badass website, and you can help me.

I need help in some of the following areas, but that is nothing compared to what could be accomplished. Give me your ideas, feedback, and other things. The site isn't public quite yet, but if you ask nicely I'll throw you a link. It should be launching in 3-4 weeks, at least for a some-what private beta. Your help would be greatly appreciated.

Marketing: People need to know about the site...Else nobody will use it. The more people that use the site, the more useful it becomes. It is already useful in it's vanilla state (as an events aggregater for the local community), but with people using it, adding events, and all the other planned interactions then it will become so much better...I can provide generic events for big venues, but people are required for the local knowledge of the Dixie Jims playing the Hot Dog Opera downtown, or Junk Science playing the loft last Thursday.

Design: I'm a technical person, I can't design my way out of a box. I need critiques and other stuff from people who know their shit. From color schemes to spacing of text on a page, I know some people live for that stuff. Any help is greatly appreciated. The current look is fredericksburg themed, and that is here to stay, but any cool photographs or idea for integrating the city more into the design is muchly appreciated.

Writing: Lots of you fools are great writers (yea liberal arts!). The best way to get people to remember cool things (this site) is a story. You can tell them, and people want to hear them. I want to talk about some cool ways of expressing all the shit I said above better, and in more memorable ways.

Awesomeness in general: IDEAS!!!

The point of this post is multiple. One is that once this is public, I have to do it or I look like a fool. Two is to get people talking about it. Have a conversation about it, come up with ideas. Yell at me or buy me a beer, any thought on this problem is progress.

Thanks for your time...

Obama & Va

I'm really excited that Obama "won" Super Tuesday. He got more states and more votes, and has been declared the winner, even though Hilary is only like 5 votes behind. I'm glad that Virginia is actually important this primary season, and I think this is the first time I will ever vote. I have never had a good reason to vote before, but Obama is such an inspiration. As is almost cliche these days, he is actually inspiring me to get out and vote! That is no small undertaking, and I believe a large part of how he is doing so well. Young people aren't apathetic, all previous candidates have just been God Awful.

I also love this onion piece about the candidates and their stances on Iraq, so funny.

Hillary Clinton: "I would never have voted for the war had we known it would become unpopular."

In closing, here's to kicking out Bush! :) The original homeland security

Work this week

At work this week I've been tasked with using PL/SQL (Oracles version of SQL scripting) to create traverse a tree structure stored in a database. The data is stored in a simple tree, with each node having an id, and a parent_id. When the parent_id is null, then that means it is a root node. This structure will be used to display navigational links for pages in an automatic fashion.

I found a couple useful websites that have helped me do this, so I'll link them here for future reference.

Traversing Trees in SQL

SQL Tree and Graph Algorithms

Sweet ads

I've always been a fan of those ads where one things leads to another to another without intervention. I'm sure there's a name for it, but you know what i'm talking about. Just found this neat one online from a dutch website, which is simply awesome!

Watch it and see

It reminds me of the honda commerical they did a couple years ago, using all the parts of the car. Simply stunning..

Another neat ad

This is a really cool Ford ad. I'm glad companies are starting to understand that commericals are content too. If you make them worth watching, they will get spread and your message will be heard a lot more places. Somebody has been listening to Seth Godin :)

All majors are the same

Had an interesting conversation with my roomate Mike last night. It helped me clarify something I have always understood, but never found a good way to say. This is probably going to be another failed attempt, but here goes.

All majors in college are generally the same. You are learning the same ideas in a different context. Mike is writing 'something', using changing channels on a television to represent changing of ideas. He is playing with all of the points of view, from first to third, and then on to 'fourth' and fifth as well. In talking about the style that he is writing I began to look at it through my computer science viewpoint on things. Viewing the writing in levels of abstractions, arranged in a hierarchy, starting from the first person point of view on the bottom and working up.

This got me thinking about the top level of both of our majors. It is generally assumed that English and Computer Science majors are about as far apart on the spectrum of brain workings as you can get, but not the way I see it. My take on writing is that it is one level above computers in the abstract hierarchy of everything. A programming language and a spoken language perform the same operation, but the programming language is limited by hardware. The English language is the programming language of our brain, and it offers the utmost flexibility. You can do anything with words and ideas (thanks metaphor). Computers are one way of limiting that excellent flexibility.

Good design embraces constraints, and the technical constraints provided by a computer allow me a frame of reference. I embrace the computer because of its technical nature. It is the closest thing that humans are gotten to constructing an artificial brain. AI being a huge area of study in computers. So computer science is simply learning to think like a computer. The programming language that you use to program a computer is also how you think about a computer. An analogy in spoken language: there are ideas in German that can't be expressed in English. The constraint of language doesn't allow one to conceptualize some part of that idea.

The brain is the hardware that English majors write for. The computer is my hardware. We are using the same ideas just applied through a different lens.

Crazy times

Only have 2 weeks left until presenting at NCUR. I give my honors project presentation to the UMW Compsci faculty on the Wednesday before the conference. Lots of work to do, but enjoying it. Getting to really dig into Django and learn it and understand it's modularity is awesome. I'm super busy and it feels like real life is starting...Should be a fun endless summer.

PS Text editors are also cool in a GUI mode over an X connection. You only update one tiny little place on the screen at a time, and doesn't reload much of anything. Works much better than firefox, which is quite the opposite.

Predictive text FTW

I am just starting to use Gnome Do, and it's amazing. You hit (windows key) + space, and it pops up a little window where you give it commands. It tries to figure out what you mean when you type a certain combination of words, and remembers what you usually do on those combos, and does that in the future.

Eventually this could replace my entire need for a command line, in a much more intuitive and user friendly sense. Getting instant feedback, and seeing where you can stop typing into the command gives you a "free hint" as to the minimum amount of typing required to perform an action.

Presumably all actions can't be condensed down into a simple combination of 3-4 keystrokes (the level where you see a noticable time savings), so only the most commonly used actions will be useful for this (the whole learning from you thing). I believe that if somehow you incorporated context into the choice when you are figuring out what to call from what combination of letters, you could solve that problem....The hard part, defining the context.

I think that for example, since I'm writing a blog post, and I type (after calling Gnome do) 'sp', then I want to check the SPelling of whatever I'm typing. However, if I'm inside my mail application (or G-mail if you want to get fancy), 'sp' means SPam, either that this is spam, or call the spam dialog (or whatever I usually do when I type spam(while viewing a message, or from the Start screen)). This was a trivial example, but it lead to a very easy analogy. I feel that this concept could be extended almost indefinitely, with the only limitation being on how we define context, and how smart the algorithm is.

If the algorithm gets good enough, you get to stop typing the shortcut.

Edit: I feel this is the same way that spam filtering currently works. Look at all of the incoming mail, and look for patterns in it. If you dont' define an "action" and a "recipient", then the input becomes one item, dependent on the context. I think you could adapt a Bayesian algorithm to do this very thing. (Completely ignorant on how Bayesian filters actually work, could be way off point).

I also believe the major downfall of this idea is the concept of "modes". Modes are generally a bad thing, and having all of this context will confuse some people. If they don't understand the different contexts, and doing the same thing has different results sometimes, they will consider it "broken". I think the visual feedback provided by the program will be crucial in how this is interpreted. Having the feedback will allow the person to know what the program is going to do. Also, some visual representation of the current context (like a favicon in the corner) will allow them to understand what context their in, and why the program is doing what it is doing.

My Second Poem Ever

Watched Dead Poets Society about 3 times in the last 2 weeks, and was inspired in the ways of rhyme and rhythm. Here's a poem, loosely based on one of my favorite quotes.."The more you know, the more you know you don't know"..

Untitled.

When the Light out is stronger than the light within.
Don't be afraid to let it begin.
The torch of knowledge, torch of soul.
The torch neverending. Light; a toll.

Browser Login Discovery

There are some really cool ideas floating around the interwebs these days, dealing with discovery of authentication. A lot of the talk is about integrating OpenID into the browser, but I don't think it needs to be limited to that. People are working on good ways to auto-discover what the login end-points are on some pages. So when I go to ericholscher.com,

there will be a specific URL to go to that will list the places where you can login, and what they support. For example: /authEnds.xml would say that /account/login/ is the endpoint of login on my site.

I feel that this could be builtin to the browser. That wouldn't be too hard, but I think we can do it with already existing technologies. All of the major browsers currently support saving of passwords for logins on a site. This means that they know what page, and what domain they are on, for the correct login information to be displayed. Why can't we make a Firefox extension that does:

Land on a page. Search saved passwords for that domain. If a saved password exists, display a small dropdown box to ask if we would like to login (if we aren't already). Submit saved login information to the saved login form destination. PROFIT!

Power through conversation

I feel that I am most able to convey my ideas and gain new ones through conversation. Viewing a blog as a conversation is interesting, but I have yet to gain the same value through a blog as a good conversation in real life. I feel like I ask good questions, and have a skill in the ability to conduct conversation well. I feel that this is a good skill, and not something that needs to be changed. I guess the new skill that needs to be learned is how to make online 'conversations' more like the real life ones described above.

Graduation

Three days until graduation. I'm getting really excited. Getting all of my stuff in order to go out to Kansas, and enjoying the rest of my summer. I'll start posting more frequently soon hopefully, since I'll have lots of free time, and need to be getting into programming mode for my job. Super super super excited about life right now!!!

Lawrence Day 1

FRICKIN SWEET. What a cool town. Couple random things made my day today.

First was the guy walking around downtown just giving everyone peace signs. Eyeing them down until they looked at him, and people gave him peace signs back! It was so funny and random, and just made my day. What a great community of people.

The second was the random drum circle in downtown. There were about 3 or 4 people sitting around playing Djembe's in the downtown (I almost said Djangos haha). Not asking for money or anything, but just chillin and providing good music for the people. Very cool.

The third was the guy from the video game store out on the street selling $1 N64 and SNES games. I didn't buy any because I owned most of the ones he was selling, but damn that's awesome.

Had a great day just driving around the town checkin stuff out. They have 3 disc golf courses, which is awesome. Went out to Clinton Lake and the place is gorgeous. There seems to be a swimming area, but it was in the pay section of the park and I didn't have any cash.

My house is amazing. I get the biggest room and my window goes out onto the porch roof so I get my own private huge little balcony area. The house is beautiful, Victorian and old. Wood floors in the bedrooms, and lots of cool decorations. I setup a few of my treasured belongings around the house and it already feels like home.

Went out last night with one roommate and their friends and got along great, had a great time. Then went with Emily (my aussie friend) and Jake to see Friday the 13th at midnight, on Friday the 13th at the local really cool theater. Great first night!

Here's to more good times. Cheers!

Goodbye East Coast part 1

Hey world, how goes? This is part 1 of 3 in my whirlwind trip around the East coast. This is the trip to Boston, Part 2 is the trip around Virginia, and Part 3 is Maryland. Here goes nothing!

I graduated! and I'm moving to (fucking) Kansas! Just took an awesome trip around the East Coast, since I won't have a chance to do it for another little while.

The first trip was to Boston and was awesomely coincidental. The day after graduation there was a Radiohead show at the Nissan Pavilion. I decided to give myself a graduation present and head to that. My friend Emily was going, and her friend from New Hampshire was coming down to go. So my roommate josh and I went with them to the show. (It's annoying when your friends have no online presence, can't link their names to anything except facebook, and I refuse to do that).

The show was AMAZING, it was pouring down rain and we had lawn seats. The experience was epic, the crowd huddled together in the rain for the love of the music. On the ride back from the show we were talking about life, and it turned out that Elliot went to school just south of Boston, and happened to be looking for someone to share gas money with on his ride back. He was leaving that Thursday, and I found myself a ride to Boston. I went online and realized that Barcamp Boston was happening the same weekend. SCORE! Now I have something to do, and a free ride there. All I needed was a place to stay.

My roommate Staiti and my good friend Johnny Mac are both Massholes, so I asked them if they had any recommendations for accommodation. J mac set me up with his friend Drew who lives in Cambridge and goes to Emerson college. I got his number and my whirlwind trip to Boston was off.

We left Thursday afternoon (avoiding rush hour in NoVA and DC) and got into Boston around 4am Friday morning. The dorms at Elliot's college were empty, so I got to crash in my own bed in my own room, double score. I hopped on the commuter rail into Boston the next morning, and called Drew. I got into Central Station in Cambridge and met Drew and his roommate at Whole Foods. I walked in the door and they had Orangina for sale for cheap! Super double ultra Triple Score! Orangina is like crack to me, and it's expensive and hard to find in Va.

So we go back to their awesome apartment (old brick and beautiful), and it turns out one of their roommates is in the Middle East for a couple weeks, and I get my own bedroom, with a TV, stereo, queen sized bed, etc. SET UP. So I crashed there for 4 days, taking a train home on Monday at 9pm. The trip was awesome and I got to know those kids really well. (It's so nice having friends with friends that you know you can get along with, laid back people FTW!)

I went to Barcamp on both Saturday and Sunday and it was amazing. I've never been to anything like it, and it was an amazing experience. I met lots of like minded, incredibly intelligent people. A very novel feeling to be able to relate to people on a technical and social level. Lots of good connections made for hopefully starting a company in the future, and knowing amazing people. Namely Jonathan and Simon. One starting a startup, and the other already having a pretty successful one. Jay was also another interesting person, and he gave a really neat presentation about the intersection of everything. I could go on all day about all the neat people that I met, but those were the ones I got to talk to the most.

It was interesting being in a place where most people were representing someone. I was representing myself, but also was able to add legitimacy to myself saying that I was going to be working for the World Company (makes of Django). It was interesting how conversations changed tone a little bit with that ounce of reputation. It was a very real experiment in the conversion between student and professional (and if it keeps going like that, it won't be a very hard transition). I look like a student and could play that card, and also have informed discussion about technical topics with a slightly more authoritative stance (hardly, but I noticed a slight change).

I really loved the community as well. If the people at the conference were any indication of my career choice, I think I picked the right career. I've known that for a while, but some outside validation is always nice as well. I've considered myself good at what I do (and knowledgeable in conversation about most things nerdy), but have never really had it put to the test. Getting job offers from two leading Python companies in doing what I want to do was some good outside validation, but events like this prove to be more so. The "thrown to the lions" approach, and very telling about how much you know.

I fell in love with Boston and could see myself living there someday. Seems like the San Fran of the East. The train ride back was at night and I slept most of the time, and only cost $80, probably cheaper than driving. The Amtrak had power outlets, but no Wifi. It was nice to have room to work on the laptop, with power, and having no internet actually lets you get more work done.

JOB!!

Eek! The job starts Monday! That is like, 36 hours from now. I'm really excited and slightly nervous. The excited feeling comes from the place I'll be working. For posterities sake, here is the series of interactions that landed me the job:

Senior Project

Start using Django and Python to learn the Python language. Fall in love with Django pretty close to the beginning of the project.

Thanksgiving

I read most of the Django documentation PDF in the Virgin Islands over break, and really fall in love.

April

Start applying for jobs, and decide to work with Perl or Python. Decide working with Django would be awesome! The people who invented it are hiring, why not go straight to the source...

Their job posting

Unofficially, the job description is "build cool shit." Our goals are nothing short of being the coolest and most innovative web teams in the world. We're the people newsrooms come to when they need to implement special features and new sites, but we do a whole lot more: Full Posting

Sounds awesome!

My e-mail to them Title: Junior Developer with extreme ass-kicking skills

Body: I've been a huge fan of LAMP style development for a good 4 years. I started using Perl and have switched to using Django and Python for my latest project (very impressed). I attached my resume, and my current site is http://www.fredvents.com It still isn't launched, but getting closer everyday. I've heard a lot about Lawrence from my friend I met from there while studying aboard in Australia. Wakarusa is a big draw, as well as the amazing music scene.

My resume is attached with more examples of my work and other formalities.

Cheers, Eric Holscher

And the rest is history waiting to be made! I start Monday!!!

Things I say all the time

I just got around to updating my profiles and online stuffs (Graduating, Moving, and getting a new job will do that!). I just updated my "about me" section, and threw in some things that I say way too often (My friends can vouch for this). Anyway, I think they say a lot about me (and are said a lot by me)...

Awkward is a state of mind.

I don't believe in Boredom.

Do what you love; Love what you do.

Appreciate the present!

Cheers!

Bear Head

My house just got a bear head. It is about 2.5 feet fall, and real. It came from a museum. My house rules.

That is all.

Change of RSS address

Hey all, anyone who is getting this content on an RSS reader, if you could please update your links to point at http://feeds.feedburner.com/EricsThoughts . I'm starting to use feedburner, and i can change that to where ever i'll be blogging, so that's the last RSS url you'll need for me. :) Thanks!

Living well

Loving Lawrence still. I haven't had to fill up my gas tank since I've been in town. My car probably hasn't moved in about 3 days. I have established a pretty good schedule, and I have been living really well.

I've been riding my bike to work all week, and it's been awesome. It's only about a mile and it's a really pretty ride. I can either go through downtown, or through the old neighborhoods in Lawrence. Win-win I say. Tuesday and Friday afternoon's we play pickup soccer in South Park, and that's been going really well. I forgot how much I love playing soccer (played for 12+ years), and the only reason I stopped was because I hurt my knees. I'm hoping I don't re-injure myself, because that will mess up all my healthy activity. Now I just need to start actually working out and lifting some weights.

There are lots of awesome cheap * nights in town because of the fact it's a college town. Here are a couple of the bigger one's that I've happened upon so far. Monday's: $1 bowling games, $.30 wings at the bowling alley. $1.75 pints at the Free State Brewery (Best Damn Brewery in Kansas!) Tuesday: 2 for 1 Ice Cream at the Ice Cream place downtown! I'm sure there's more, but I can't think of them right now. But those make the beginning of the week awesome!

Plus my job is awesome. Working with great people doing interesting stuffs. :)

Cheers!!

DjangoCon September 6-7, at Google!

Heard from Robert Lofthouse on Twitter. The Djangocon 2008 conference will be held at Google Campus (Googleplex) in Mountain View!! September 6 and 7th. That's only two months away, so hopefully this gets pulled together well.

The Official Post announcing DjangoCon. The DjangoCon website should be up on friday.

The DjangoCon Website

Automating tests in Django

Updated

At work lately we've been writing a bunch of tests for all of the work we've been doing. This is generally a good thing (tm). I was getting tired of manually having to write all of the code to test the views inside of my app. So I decide to write a little app that helps me automate the writing of tests.

I wrote a piece of middleware (that should obviously only be used during development!) that shadows the current activity in django into a log file. This log file should then be ready to copy and paste into a doctest for easy testing of your views. This is a little hard to explain, but the code should be pretty self explanitory.

I created a 'google code project <http://code.google.com/p/django-testmaker/>'_for it so that people can go ahead and hack on it and make it better. It is pretty rudimentary at current, but it gets the job done.

I think a big win from this approach is that your testing data is much more "real", since it's a copy of your session with a real browser. I know writing django tests I sometimes use contrived data because it is a pain to enter it all. This should help improve on that situation.

Here is a video of it in action, this should allow it to make more sense.

Django TestMaker from Eric Holscher on Vimeo.

Writeup

Figured it would be good to writeup the screencast.

Step 1: Get django-testmaker `svn checkout http://django-testmaker.googlecode.com/svn/trunk/ django-testmaker-read-only`

Make sure the testmaker module is in your PYTHONPATH.

Step 2: Add `'testmaker.middleware.testmaker.TestMakerMiddleware',`

to your MIDDLEWARE_CLASSES in your settings file.

Step 3: Run the test server with the middleware installed. `./manage.py runserver`

Browse around your site.

Step 4: run `tail -f /tmp/testmaker.log`

to see your output.

Step 5: Take the output from testmaker.log and put it into a file in PROJECT/tests.py. Make sure that your tests.py contains:

```
"""
>>> from django.core.management import call_command
>>> call_command('loaddata', 'PATH/TO/PROJECT/fixtures/PROJECT.json', verbosity=0)
>>> from django.test import Client
>>> c = Client()
YOUR TESTS GOES HERE
"""
```

at the top of your tests.py file.

Step 6: Run the command `./manage.py dumpdata > PROEJCT/fixtures/PROJECT.json`

You can have dumpdata just dump the data for a single project if you provide PROJECT as an argument to it. Be warned though, that the tests might break because of it using data from other apps. (Like my example would break because the mine project uses data from my blog app.)

Step 7: `./manage.py test PROJECT`

Step 8: PROFIT!!

Update

I added a management command to the project to simply this process a ton. I'll be making another screencast and blog post (and maybe even some REAL DOCS!) tonight, so stay tuned for that.

Testmaker .002 (Even easier automated testing in Django)

Okay, Well I have been hacking away at django-testmaker for the last couple of days based on some ideas from the community. It has gotten a lot better, so here is another blog post showing what's new.

First let me just say how awesome the Django community is, and I am really beginning to understand (and appreciate!) the open source ethos. I would never have gotten this code this good in 3 days without releasing it to the public. Thanks everyone for looking over it and giving feedback!

New stuffs

Testmaker got a management command! Now you don't have to worry about messing with your middlware and having to take out the testmaker stuff when in production. You simply add testmaker to your INSTALLED_APPS and away you go.

Here is another video of it in action:

Django Testmaker v2

How to make it work

Step 1: Add testmaker to your INSTALLED_APPS settings.

Step 2: In the directory above the APP that you want to test, run `./manage.py testmaker -a APP`

This should tell you where it is logging to, and where the fixture files are going. It should only make fixtures once, so if you change something in your database, you need to go ahead and delete the old fixtures file and it will re-create a new one. It also has 3rd grade file-naming heuristics built in. So if a tests directory exists in your project, it will log to APP-testmaker.py, if there is no tests.py in your APP directory, it will put itself in that file. If you have an existing tests.py file, it will make a tests-testmaker.py file. You will then need to take the contents of this file and integrate it into your normal tests. In a future version if it encounters a tests.py, it may make a tests directory and put both files inside of that, but I don't know if that is a good idea.

Step 3: Once you have reconciled the above changes (only necessary if you previously had a tests.py file in your APP directory) `./manage.py test APP`

Then you have awesome base-line tests for your app.

In my release post I had some comments about the usefulness of these tests. I think that it is a very useful thing if you have an existing body of code with no tests. This will give you a non-trivial base to then at least have tests for your code.

Testing all of your views will also presumably alert you to errors introduced in your models and URLConf files as well. Having dedicated tests to testing models is still better and a good idea, but this code will at least give you a good starting point.

Being able to automate a base-line level of tests for an app will hopefully make people more inclined to include these basic tests in their apps, and everyone knows tests are better than no tests.

Known issues

There are also a few problems that I've had with the output. It appears Satchmo is hijacking the logging module on output? If anyone knows a good way to fix this, please let me know.

Also, the POSTing stuff hasn't been well tested, so there might be a few bugs in that, it is pretty rudimentary.

Beatles Lecture

This is a video from Gardner Campbell, one of the best professors ever, English Professor at the University of Mary Washington.

It is a talk about the Beatles...Elequent, engaging, and breathtaking, an amazing talk.

Enjoy it

Jim Henson before Sesame Street

I'm currently reading Malcolm Gladwell's Tipping Point, and it is an amazing book. One thing that he mentions is that the Muppets were actually used by Jim Henson before Sesame Street to do advertising! I never knew this, and find it fascinating. There are a bunch on youtube that some posted. Great stuff!

DjangoCon 2008

It is being kinda announced about DjangoCon 2008! It is going to be in the Bay Area and sometime around the release of Django 1.0 in September. I heard about it a couple days ago from Jacob at the office (because I work at Mediaphormedia, birthplace of Django). I'm really excited about it, and I'm thinking about heading out for it. I have never been to the Bay Area. We'll see how it pans out when it gets announced, but I'm almost definitely going! YAY!!

Update: An official announcement should be made Monday or Tuesday, including dates and location.

Check my post here for updates: DjangoCon: September 6-7, GooglePlex

Setting up Django and mod_wsgi

I was just convinced to setup mod_wsgi on my server instead of mod_python, and I'm going to write up how I did it. All of the documentation I found on the internet was really hard to follow, so I'm going to distill it here the best that I can.

This is assuming Ubuntu 8.04 Server Edition.

Update: Take note, this is installing mod_wsgi 1.3. The latest version of the package is 2.3. If you want to get the latest version from apt, you should use the Debian 2.3 package

Step 1: `apt-get install libapache2-mod-wsgi`

This should automatically install mod_wsgi into your apache instance and install it.

Step 2: Create an apache directory on your filesystem, presumably inside of your Django project. I keep my code in ~/Python/Project, so I did:

```
mkdir ~/Python/PROJECT/apache
vim ~/Python/PROJECT/apache/django.wsgi
```

Then in that file you need to copy this code:

```
import os, sys
sys.path.append('/home/eric/Python/PROJECT')
os.environ['DJANGO_SETTINGS_MODULE'] = 'PROJECT.settings'
import django.core.handlers.wsgi
application = django.core.handlers.wsgi.WSGIHandler()
```

This creates an interface between Django and WSGI, as far as I can tell. If you start getting errors about not seeing your project or modules, try adjusting and/or adding some things to your sys.path.

Step 3:

Inside your /etc/apache2/ directoy, you will find the directory sites-available/. This is where you are going to put your configuration for your server. Presumably it will have a file called default in it that you will edit. So:

In /etc/apache2/sites-available/default:

```
<VirtualHost *:80>
ServerAdmin eric@ericholscher.com
ServerName  ericholscher.com
ServerAlias www.ericholscher.com
DocumentRoot /var/www/
LogLevel warn
WSGIDaemonProcess ericholscher processes=2 maximum-requests=500 threads=1
WSGIProcessGroup ericholscher
WSGIScriptAlias / /home/eric/Python/PROJECT/apache/django.wsgi
Alias /media /var/www/media/
</VirtualHost>
```

The last 3 lines of WSGI stuff if what you want to pay attention to. You are pointing WSGIScriptAlias to the file we created in Step 2. The other two WSGI prompts aren't necessary unless you are running multiple sites on your server. The Alias is so that the /media URLs on your site continue to work, it should point to where ever you have your media files stored.

Hopefully this will get you started along the way to setting up mod_wsgi on Apache with Django. If not, feel free to leave comments or email me

EDIT: Someone in the comments pointed out this website on the mod_wsgi wiki is also helpful: Integration with Django

There appears to be a page on the Django WIki as well if you need more pointers.

Using Mock objects in Django for testing the current date

Today I ran into a fun problem when writing template tags at work. (I'll write another post later on the fun-ness that is testing of template tags :) In ellington we have some templatetags that test for the current time of day. ifmorning, ifnight and so on. These template tags are using datetime.datetime.now() to check to see if the time is within a certain range. This is impossible to test in a standard way without doing some hacking on the datetime.datetime object.

The solution is actually pretty easy. Let me warn, although this is the correct solution in this case, **monkeypatching is generally BAD**. You don't want to just be playing around with python or django's stdlib and breaking things for other people. With that warning, let me show you how I went about doing this.

This code is called in this fashion:

```
import unittest
class LoadDateutil(TemplateTestCase):
    def test_load(self):
        olddatetime = datetime.datetime
```

```
        datetime.datetime = make_datetime(5)
        self.assertEqual(self.render(u'{% load dateutil %}{% ifnight %}Hi{% endifnight %}'), u'')
        datetime.datetime = olddatetime
```

Now let me explain what all is going on here. TemplateTestCase is an internal base class for doing templatetag tests. This will probably be released (by me or Matt Croydon) sometime soonish.

The first thing you want to make sure you do is leave everything how you found it. So before we go about editing the `datetime.datetime` object, we save it into `olddatetime`, and once we are done with the test, we return `datetime.datetime` back to its original value. In the middle of the test, we are calling `datetime.datetime = make_datetime(5)` which is returning a datetime.datetime object that has it's now() method overwritten. The argument to `make_datetime` is the hour of the day you want to represent.

Let's take a look at how make_datetime is working:

```
import datetime
def make_datetime(hour):
    class MockDatetime(datetime.datetime):
        @classmethod
        def now(cls):
            return datetime.datetime(2007, 1, 1, hour)
    return MockDatetime
```

This code is creating the `MockDatetime` class, and then defining the now() method. The `@classmethod` decorator must be used because the now() method is a class method. Then the now() method simply returns a `datetime.datetime` object with the correct hour in it.

This is pretty simple, and is the correct and best way to do testing of this nature. I hope this is helpful to someone out there :) Also note that these methods are not django specific, and can be used in anything with python.

As a caveat, make sure that the templatetag code you are running this test against is importing the datetime module, and not datetime.datetime, because in that instance this code will not work (because the code we're overwriting will be re-imported in the templatetag (as far as I can tell))

Thanks to Malcolm for helping me with this.

Screencast: Debugging with the Django Error Page

This is part 1 of a week long series of screencasts

Hey Everyone, I'm here to make a minor announcement. In the upcoming 7 days, I'm going to be releasing 5-7 screencasts on Django, mostly focused on debugging, and hopefully trying to throw in a couple of other useful ones that people might be interested it. This is my own way of helping people get ready for 1.0 and hopefully sharing some good tips on ways to use Django well. Please add you own comments and tips to the end of the posts! Please subscribe to my feed to get all of the screencasts (It'll be worth it!)

Also a big thanks to Simon Willison who's excellent debugging blog post got me started and interested in this stuff.

Screencast 1

The first in the series is going to be how to use the Django error page to it's fullest. It is a very useful piece of work (Thanks Wilson).

Setup *Please install Django Command Extensions and The Werkzeug Debugger before you go on.

```
svn checkout http://django-command-extensions.googlecode.com/svn/trunk/ django-command-extensions
easy_install Werkzeug
```

Make sure that they are installed (on your PYTHONPATH) before continuing on.

Video The full video can be downloaded here (18MB H.264 .mov)

Debugging with the Django error page from Eric Holscher on Vimeo.

Writeup First off I fire up the debug server. Showing people how to use `assert False`. You use `assert False` when you just want to bring up a debug page to look at your context. It will bring up an `AssertionError` at the point where you put this code. `assert False, foo` brings up the error, showing whatever is in `foo` on the top of the error page.

The debug page is really useful because it contains a lot of useful information. It shows your entire ENVironment, including your GET, POST, COOKIES, PYTHONPATH, and lots of other good data. This information is really useful for debugging forms and session errors especially.

I then showed how to post your error to dpaste with one click in the error page. This is really handy for sharing your errors and tracebacks with people for them to help you debug your code.

"Part 2" of the screencast is about using the Werkzeug debugger for **more debugging power**. The command to run the Werkzeug debugger (with django-command-extensions) is:

```
./manage.py runserver_plus 67.207.139.9:8000 --settings settings_debug
```

with settings_debug.py being in the same file as your settings (and the current directory) containing:

```
from settings import *
DEBUG = True
INTERNAL_IPS = ['YOUR_IP']
```

whatismyip.com is a really handy utility for getting your external IP address. Then once you restart your debugging server, your error page should be the Werkzeug error page!

The Werkzeug error page is similiar to the django one, it has a traceback and all that good stuff. The killer feature however is that you can open up a python console at any of the places in your backtrace!

The `dump()` command inside the Werkzeug console is really handy. It will output a prettyprinted version of whatever you pass in. Allowing you to readily see what internal variables your object that you're debugging has.

Remember this is powerful! NEVER use this in production! People will have access to all your data!! It is a very powerful debugging tool, which is a double-edged sword.

PS: I hope you enjoyed me floundering in that later part. I thought it showed the value in the debugger so I left it in :)

Stay tuned daily at my feed for posts everyday up until Django 1.0! Cheers.

Screencast 2: Logging in Django, for fun and profit

This is the second screencast of a week long series.

So that I don't spam all of the Django Community Feed (Bad RSS handling has done that more than once, Sorry!) I'm only going to be posting this post and the last post summarizing all of the screencasts on the aggregator. So if you're trying to keep up with all of the screencasts that will be coming out this week, either stay tuned to the site or subscribe to my feed.

Screencast 2: Logging in Django

Setup This screencast is going to be about how to use the python logging module in Django. It's in the Python standard library, so there is nothing extra to install to use the simple python logging module. The screencast also makes use of the excellent Django-Logging, which should be downloaded and installed beforehand.

```
svn checkout http://django-logging.googlecode.com/svn/trunk/ django-logging
```

This allows for you to follow along on the second part of the screencast. I also do a little bit of work inside my django-testmaker app, so you can go grab that if you want to follow along as well. If not no big deal, the screencast is more about general logging anyway.

Video and Download The video is available to download in higher res or for streaming.

Download (21MB H.264 .mov)

Screencast 2: Using Logging in Django from Eric Holscher on Vimeo.

Writeup

Part 1: python logging module We start out with some really simple logging methods. The first is a simple `print` statement inside of your views. This outputs the command to the terminal in your development server. The next way to do it is with a simple logging command:

```
import logging
logging.error('your error goes here!')
```

During the screencast I say that logging.error goes to Standard Out, when in actuality it goes to Standard Error. In this case they're the same thing...Then I go through how logging is done within the testmaker app. This is the basic setup for python's logging module that I used:

```
logging.basicConfig(level=logging.INFO,
            format='%(message)s',
            filename= "/file/to/log/to",
            filemode='w'
            )
```

More documentation about this and the python logging module is available here It includes a lot of really good information about the logging module, like lots of message formatting options. I show how you can tail a log file from testmaker and talk about the neat advanced features of the logging module like logging across a network. As you can see the logging module is **very powerful!**

Part 2: django-logging In the second part of the screencast we show how to use the Django Logging middleware. This is what is going to go into your settings_debug.py:

```
from settings import *
DEBUG = True
INTERNAL_IPS = ['YOUR.IP.HERE']
MIDDLEWARE_CLASSES += ('djangologging.middleware.LoggingMiddleware',)
LOGGING_LOG_SQL = True
```

This then has all of your logging output appended to the bottom of the page you're currently on. This gives you a similiar capability of using the error page to debug, except you don't have to have an error. You can debug while the pages are still working.

The next and last really neat feature is showing the SQL queries that are being executed to render a page. This is incredibly useful for fine-tuning your django sites. With the ORM, there are tiny little tweaks that you can sometimes make to decrease (or increase!) the numbers of queries you execute pretty dramatically. Having this enabled during

development is a good way to catch those mistakes, and really understand what is going on under the hood of your Django apps.

Remember, when you wrap all of your logging stuff in the logging module, you can do some really neat things after the fact. I was able to use all of the django-logging stuff while still logging output to a file and whatever else happened to be going on with the logger at the time. This in incredibly powerful, and allows you to almost have django-style 'pluggable logging backends' that do different things.

Again in this one I end a little silly: Thanks for your time and have a good day (since I'l hopefully see you again tomorrow :))

Using pdb, the Python Debugger (Django Debugging Series, Part 3)

I had a couple of comments about my last post saying that I should be sending all of the screencasts to the aggregator because this is content and isn't spam. So I'm going to do that. Thanks for all the feedback everyone! Hope you're enjoying the series.

Screencast 3

Setup No setup today. pdb is included with python, so everything that you need is available at a python install near you.

Video and Download You can download the video here (17MB mov)

Screencast 3: Using pdb from Eric Holscher on Vimeo.

Writeup I started the show by talking about a little bash alias that I made to be able to run the testserver from anywhere. Here is that code, edit accordingly:

```
alias rs='/usr/bin/python ~/EH/manage.py runserver 67.207.139.9:8000
--settings settings_debug'
```

In order to get into the debugger, you need to call it inside of any of your python code.

```
import pdb
pdb.set_trace()
```

Then I go in to talk about the basic Pdb commands:

- l (list): Shows the current code around the line that your on. The line that is about to be executed has a -> before it.
- n (next): Executes the current line and moves to the next in the current file.
- c (continue): Finishes the debugging session. If there are more breakpoints (or if your set_trace() code gets called again before the request finishes) then you will get back to the debugger, otherwise the requests will complete back to the browser.
- s (step): Goes down into the next level of execution (presumably a different file). You can follow your code through Django's internals this way. This is really good for finding bugs and getting a better understanding about how Django works.
- w (where): Shows you a backtrace of the calls that have gotten you to the current point in the code execution. This is really handy for the following 2 commands.
- u (up): Allows you to go up one level in the backtrace.

- d (down): Allows you to go down one level in the backtrace. These two commands allow you to see where you came from, and what variables were called where. This lets you see how the state ended up the current way that it did, which is great for figuring out how to fix it. :)
- locals(): This isn't a debugger function, but it is really handy for seeing what is in the current scope that you can muck around with. locals().keys() is really nice too just to see the variables that are there, because request tends to pollute the locals() command.

I had about double the content that is in this screencast to talk about pdb. It is incredibly powerful and there are lots of other neat things you can do with it. This screencast was running a decent length already, so I decided to split it into 2 parts. This is more of the "Intro to pdb" part, and tomorrow, I will be presenting a little bit more advanced/different use case for the debugger.

Stay tuned and have a good labor day weekend if you're in America. Cheers!

Easily packaging and distributing Django apps with setuptools and easy_install

First off let me say that I know that not everyone likes setuptools and that is fine. distutils works well and is included with python. However, I believe that Python needs to get some parity with what Perl has with CPAN. Pypi is Python's alternative, so tools that integrate with it are good.

Step 1: In the directory above the source directory of your project, create a setup.py file. In that file put something like this (editing for your project):

```
import ez_setup
ez_setup.use_setuptools()
from setuptools import setup, find_packages
setup(
    name = "django-testmaker",
    version = "0.1",
    packages = find_packages(),
    author = "Eric Holscher",
    author_email = "eric@ericholscher.com",
    description = "A package to help automate creation of testing in Django",
    url = "http://code.google.com/p/django-testmaker/",
    include_package_data = True
)
```

This is all that you need to do to really get your file in a state to upload. name is the name that your package will go under on Pypi, so choose something unique and descriptive. version is the version of your app, it understands most common versioning schemes, more info here. include_package_data makes sure that ez_setup.py gets included in your package. find_packages() is a setuptools thing that includes everything in the current directory that it thinks is a python module, you can also simply put the name of your app there.

```
wget http://peak.telecommunity.com/dist/ez_setup.py
```

If you're on a platform that doesn't have wget, simply download that file into the directory containing setup.py and your app. This will make sure that the person who downloads your app has setuptools installed. If they don't, it will automatically be installed.

Now run

```
$ python setup.py register
We need to know who you are, so please choose either:
 1. use your existing login,
 2. register as a new user,
 3. have the server generate a new password for you (and email it to you), or
 4. quit
Your selection [default 1]:  2
```

```
Username: whatever
Password:
 Confirm:
   EMail: your@email.com
You will receive an email shortly.
Follow the instructions in it to complete registration.
```

Obviously, if you already have an account, simply say 1 and login. Finish the login procedure in your e-mail and then login on the command line.

```
python setup.py register
Your selection [default 1]:  1
Username: whatever
Password:
Server response (200): OK
I can store your PyPI login so future submissions will be faster.
(the login will be stored in /home/eric/.pypirc)
Save your login (y/N)?y
```

Now you will be logged into Pypi and will be able to upload your files. Then you want to upload your package to Pypi.

```
python setup.py sdist upload
```

This should output a bunch of code with your app being packaged. The bottom of it should contain a 200 and say that it was successfully uploaded.

```
running upload
Submitting dist/django-testmaker-0.1.tar.gz to http://pypi.python.org/pypi
Server response (200): OK
```

Congrats! Your code is now in Pypi! It can be seen and downloaded by the world, and easily installed on people's machines. If you want to tell people how to install your app, it's as easy as..

```
[sudo] easy_install django-testmaker
```

For some reason if easy_install doesn't work, you can still resort to installing things the old way. You can grab the code out of svn, or take the package from Pypi, unzip it, and move the project directory somewhere on your PYTHONPATH.

Let me know if this doesn't work, of if I've missed something obvious. This is my first work done packaging a django app. I really think that it would help a lot if all django (and python in general) apps were easily installable and listed at Pypi. Having the directory really helps with people locating apps. Hopefully a culture of apps getting put up on Pypi will make this reality a little bit closer.

Setuptools can do a lot of neat things. Look at their site for more information on advanced packaging (like dependencies and other things).

Screencast: Django Command Extensions

This is a screencast on the Django Command Extensions project. It is one of my favorite third party apps, and it gets installed in every Django environment I work in. It provides a plethora of useful manage.py commands, and a couple other little goodies as well.

Setup

Before you get started using these things, there are a couple of packages you need to install. The first is Graphviz which is a really nice toolkit for graph visualization. The other is Werkzeug which is a little python web framework with an amazing debugger that we'll be using. There can easily be installed:

```
apt-get install graphviz
easy_install Werkzeug
```

Video and Download

A high-res version of the video is available here (mov 21MB)

Django Command Extensions from Eric Holscher on Vimeo.

Writeup

The website for the django extensions has a pretty good list of all of the commands that are available. Below I will just write about the way to use some of them that isn't well documented or a bit different or unclear.

As a note, for things that output something to the screen, you can redirect that output to a file really easily. For example: `./manage.py dumpscript blog > blog.py` redirects the output to blog.py. The command extensions site has a list of output formats for export_emails here. Which is really useful.

The command for graphviz is `/manage.py graph_models auth blog |dot -Tpng -o test.png`

The output of graphviz is awesome. There are ways that you can hook this up to a url in your URLConf so that it will be regenerated whenever someone requests it (for data that changes often). That is a really nice feature for data that is changing a lot, where someone is watching over your work (school etc.).

Just to note, `runserver_plus` requires that you be running `DEBUG = True`, and that your IP Adress is in `INTERNAL_IPS`, look at my (screencast)[http://ericholscher.com/blog/2008/aug/28/screencast-debugging-django-error-page/] here for a full explanation of this, and more!

Things that weren't covered in the screencast include `sqldiff` which is still halfway working, and provides a diff against what your model and the current database look like. `create_app` and `create_command` are things that just flesh out the directory structure for a new app or management command. `create_superuser` creates a new supersuer for you. generate_secret_key gives you a new secret key. `passwd` allows you to easily change a users password. `reset_db` resets your current database.

Thanks for watching, and stay tuned for more screencasts (and other content too ;))

Getting started with Pinax

NOTE: This content is a little out of date. Some of the layout might be wrong, but the ideas are still relavent.

I went ahead today and figured that I would try out Pinax, seeing as it's been getting a lot of good press in the Django community lately. The talk from James Tauber at Djangocon was really good, and I certainly recommend checking it out. This is going to be a basic introduction to pinax.

First thing you need to do is grab the code from the pinax repository.

```
svn checkout http://svn.pinaxproject.com/pinax/trunk/ pinax
```

Once you have the code checked out you are already half of the way there. The only current external dependency is on PIL, the Python Imaging Library. Django itself has this dependency for the ImageField in forms. More than likely you already have this installed, so it shouldn't be a problem. If not, that is the only dependency of Pinax.

The directory structure of Pinax is pretty simple, copying from the README:

```
pinax/          contains a django project and templates
external_apps/  contains external re-usable apps brought in via svn:externals
local_apps/     contains re-usable apps that aren't yet externalized
core_apps/      contains non re-usable apps specific to pinax site
external_libs/  contains external libraries
```

The manage.py script inside of pinax links up all of the apps correctly for you. So all you need to do to get started is go into `pinax/` and run `./manage.py syncdb`. Then go ahead and run `./manage.py runserver` and you should see the pinax welcome page! That is pretty awesomely simple to get up and running!

At this point, you now basically have an entire social application working on your box. That is pretty damn impressive. The `pinax` directory is the project directory in this setup. Then, all you have to do is swap out the templates, and you have your own site with the exact same functionality.

The main use case for me with Pinax is to take the groundwork that they're layed and throw a little bit of custom code on top. That is the goal of the project. They are trying to give you a really solid base that provides all of the generic functionality that 99% of websites need. From this base it is then incredibly easy and fast to get "yet another X site" going, wherein you then add the magic that makes your site unique.

A couple days ago I actually took my first Django site I'd ever done and converted it over to Pinax. It took all of about two hours, with the awesome help of James Tauber and Brian Rosner in the #pinax IRC channel on Freenode. They were helpful and I bemoaned the lack of documentation, so that's why I'm writing this up :) The site is now about 100x more powerful, and it's really cool the power of Pinax there.

I want to talk about my philosophy behind the usage of Pinax. The way that I've been thinking about it is basically it gives you the groundwork with some nice default templates for the apps. The way that you go about skinning the app is with the `base.html` and `site_base.html` in the `pinax/templates/` directory. `base.html` allows you to change the basic layout of the site. This is where I changed out my CSS and Javscript code. Basically you don't want to be changing any of the block tags here, just HTML. I ripped everything out of the `<head>` except for `<title>` and `{% block extra_head %}`. In the `<body>` I basically ripped everything out, and put in my previous template, then adjusted the block tags to make them appropriate.

One of the big gotcha's is the way that template blocks are named. The pinax app templates are all coded to specific block names (as they have to be), but if you're trying to use existing templates then you might need to update your blocks if you want to be able to have the backend stuff "just work". Here is a listing of the main template block names and what they are. Remember, these simply need to be present in your `base.html`, and they will be given content in `site_base.html`.

{%block logo_link_image %} This is where your image goes (in the header) {% block login %} Is where the login stuff goes (leave this empty if you want to use their auth) {% block tabs %} Menu or tabs, since it's only used in base and site_base, this can really be anything. {% block body %} This is where your main content goes. {% block footer %} This is where you put the footer contents

A couple more gotchas:

- The pinax manage.py does a decent amount of editing of your PYTHONPATH, so if you want to deploy it then you need to understand how this works. Check out this post by Greg Newman for help with deployment!
- The media in pinax is served out of `pinax/site_media` and in URLs are site_media, so you need to put all of your css and javascript in there to get it working on the dev server. When you deploy this can go back to where it was previously (assuming previous install).
- At the bottom of the pinax settings file, you see it does an import of localsettings. You can define your own settings than override the pinax ones in a localsettings.py file anywhere on the PYTHONPATH. This keeps you pretty safe from updating pinax and having it wipe your settings in the default settings file.
- Remember this is still a work in progress, so the code will be updated (and probably break backwards compatibility) pretty frequently. Keep an eye on the BackwardsIncompatibleChanges

That's enough for today. That should get you up and running with Pinax. I will be doing a screencast on this stuff sometime this weekend, so look out for that. It should make this a lot more obvious.

Using pdb to debug management commands and unit tests (Debugging Django Series,Part 4)

Screencast 4

Today's screencast is about pdb again. This time we are going to be debugging management commands, and unit tests for django. This is a little bit more powerful than the previous screencast which just introduced the basic debugging commands.

Setup This screencast uses a couple of really handy pieces of software. iPython is a wonderful piece of software for all python developers. It gives you handy things like tab completion, syntax highlighting, and all the modern amenities that we're used to in our editor from the python shell. Django's manage.py shell even uses ipython if it detects it, that's an endorsement if i've ever heard one.

ipdb is a simple wrapper around pdb that allows you to use ipython when you are doing your debugging. This is really handy as well. To get the code for these projects, go to their websites linked above or use the following code:

```
bzr branch lp:ipython
easy_install ipdb
```

Download and Video You can download the video here (20MB mov)

Debugging management commands and unit tests from Eric Holscher on Vimeo.

Writeup We start the screencast by breaking the testmaker management command that I've written. We call it like this:

```
python -i ~/EH/manage.py testmaker 67.207.139.9:8000
```

The import thing to note is the `-i`, which tells python to drop into the python (>>>) shell after the command is run. I then show how to use pdb postmordem command to go back into the crashed management command. This is called like so:

```
import pdb
pdb.pm()
```

and this allows you to actually go back into the previous command! Even if pdb isn't currently loaded at the time. This is a really neat feature of the debugger, and incredibly useful for diagnosing breakage that is hard to reproduce. You can go back up into the application and see the actual state of the variables at that time.

Next I introduce ipython which is a really nice python distribution. It has a really nice debugger, called idpb, which gives you all the ipython commands inside the debugger.

Next we go on to run testmaker with valid input after showing how to do a simple fix to check if the input was correct. We call ipython with an app passed in:

```
ipython ~/EH/manage.py testmaker 67.207.139.9:8000 -- -a mine
```

Note that – is meant to tell bash that the input is done for ipython, and the rest will actually go to manage.py and into your python code. This is good to know for trying to pass things into management commands in ipython on the command line. This code will generate tests and fixtures for the application inside of the mine/ directory. Once we browser around a little on the test server, we have generated a unit test based on what we have done.

Assuming you have ipdb installed from Pypi, you can include ipdb inside of your unit test (or any python file being executed) and get the ipython debugger instead of vanilla pdb:

```
import ipdb
ipdb.set_trace()
```

Although I don't expound on it inside of the screencast, getting inside of tests is probably one of the more useful things you can do with the debugger. Trying to debug tests is really difficult, and sometimes they return really strange errors that are hard to get a handle on. Sometimes the line numbers are also off, and debugging doctests are notoriously hard to debug. You can debug doctests just as easily, with the following code (using ipdb if preferred):

```
>>> import pdb
>>> pdb.set_trace()
```

I figured out that the error in the unit test was actually due to a stale fixture left over from a previous run of the testmaker app. It didn't have enough data to return a paginated list, so has_next was false instead of true like when we ran it against the live database.

In related news, searching for ipdb on google made me stumble onto the The Internet Pinball Machine Database, which I didn't know existed previously. Yess!

Big list of Django tips (and some python tips too)

We were talking about things that we wish we had known before while developing for Django the other day in IRC. I proclaimed that we should write them down somewhere. So I'm writing a post to get this effort started. Please feel free to leave comments with your own tips and tricks, and I'll compile them in some kind of good fashion. These are mostly just pointers, and not full-blown writeups, just more of a big list of stuff you should think about. I think these tips will really help out new people when they're trying to get the hang of Django.

App level

Local library installation When you don't have root access on a machine, and you want to use easy_install, you can install files into a designated directory. I use ~/lib to hold my python modules, so when I do easy_install I simply use the `-d` option like so:

```
easy_install -d ~/lib nose
```

Use iPython IPython is a python distribution that gives you lots of handy features like tab completion, syntax highlighting, better debugging, and lots of other nice features. Poke around, I find more and more stuff I like every time.

Use django command extensions This is one project I use every time I do some new code. I wrote up a whole post and screencast about how good they are. They have gotten even better since then, highly recommended.

Performance tips from the man himself Jacob (co-BDFL) wrote up some performance tips back in '05 that are still relevant today. Some big architectural stuff, but they still make sense.

Use mod_wsgi mod_wsgi is currently the best way that I know of to run Django. I did a simple write up a while back that has proved popular.

Running out of memory? Web faction has a good blog entry about how to keep memory usage down. Might be useful even if you're not running on their hardware.

Use pdb Pdb is a great debugger. Simon has a great post on it, and I took a lot of his ideas and expanded them to do my debugging django screencast series. Using pdb

Read b-list.org archives My tips here are short and sweet, James has a wealth of amazingly informative Django information stowed away in his blog archives. Do yourself a favor and peek through it and be enlightened.

Don't be afraid of Reusable apps Watch James' presentation on Reusable apps at Djangocon. Learn it, and use it. A lot of the functionality that you want to do has already been done for you. Check out Pinax which has a ton of nice reusable apps. django-basic-apps also has a ton of really nice reusable apps that use best practices. I use the blog here and it's a great way to learn how to use django well. Learn by other people's awesome examples!

Watch the Djangocon videos The videos from Djangocon give you some great insights into Django.

Search and replace Search and replace across an entire directory. This is useful for changing template vars or doing basic refactoring (good editors should do this for you too)

```
perl -pi -w -e 's/foo/bar/g' *.html
```

Check out virtualenv virtualenv is an awesome python tool that allows you to create mini-sandboxes of python. You can contain an entire django install (and supposedly you can get mod_wsgi and some other stuff inside). I haven't played with it too much, but it sounds really nice to keep a contained python environment, and allows you to run different versions of libraries, django, and anything else you can think of.

Use Django snippets Django snippets is a great place to post your tips, or get other peoples code examples. It's a big cookbook of helpful and neat things about django. It doesn't have search, so use google's site:djangosnippets.org syntax to find what you need.

Use your environment! I find that my .bash_profile file is a huge help for all Django stuff I do. Here is an example or mine, I'm really curious about other people's awesome aliases and other settings foo.

```
export PYTHONPATH=$HOME/Python:$HOME/Python/Modules
export PATH=$HOME/bin:$PATH
export DJANGO_SETTINGS_MODULE="settings"
export HISTFILESIZE=10000000
set -o vi
alias rs='/usr/bin/python ~/EH/manage.py runserver 67.207.139.9:8000 --settings settings_debug'
alias mp='/usr/bin/python ~/EH/manage.py'
alias sp='/usr/bin/python ~/EH/manage.py shell_plus'
alias bkup='/usr/bin/python ~/EH/manage.py dumpdata'
alias destroy-pyc='find . -name \*.pyc -delete'
alias mod='cd ~/Python/Modules'
alias dj='cd ~/Python/Modules/django-trunk'
alias a2='sudo /etc/init.d/apache2 restart'
alias tm='/usr/bin/python ~/EH/manage.py testmaker 67.207.139.9:8000 --settings settings_debug'
alias p='python'
alias x='exit'
alias tst='./manage.py test'
```

Models

Use managers for commonly accessed queries Writing managers is really simple, and they provide a better user interface to your code. This code snippet simply adds a latest() method to the default objects manager

```
class ForecastDayManager(Manager):
    def __init__(self, *args, **kwargs):
        super(ForecastDayManager, self).__init__(*args, **kwargs)
    def latest(self):
        return self.get_query_set().order_by('forecast_date')[0]
```

It can be called `ForecastDay.objects.latest()`. This is a trivial example, but there is a lot of power that lies in this functionality.

Meta is your friend You can define the default ordering of your model, so when it returns things in a queryset you don't need to do an order_by() clause (like above). Possible settings. If you set get_latest_by, the above code is already written for you.

No really, Love meta. Ever wonder where all of that lovely metadata that you have set goes? It all gets stored in your objects _meta variable. Note the underscore, this is private and might change at some future point. However, a lot of it is stable and it gives you some really nice things that you can get access to. _default_manager is a another really nice one on query sets, this returns objects (or whatever the default manager is). It's really handy for writing re-usable code.

Settings

Relative imports When you are using a setting file multiple times, it is nice to be able to define relative variables for your things.

```
import os
DIRNAME = os.path.dirname(__file__)
DATABASE_NAME = "%s/dev.db" % DIRNAME
MEDIA_URL = os.path.join(DIRNAME, 'media')
TEMPLATE_DIRS = (
DIRNAME + "templates",
)
```

more

Local settings If you have local changes to your settings file, that you don't want to share, or that are specific to your box, there is an easy way to accomplish that. Put this at the bottom of your settings.py file:

```
try:
     from local_settings import *
except ImportError:
     pass
```

This allows you to define a local_settings.py in that same directory (or on your pythonpath if you so feel). This can then override (or add on to) the settings previously defined in the file.

Use a settings debug file. This kind of inverts the logic above, but runserver allows you to pass it a settings command. So you can run runserver with the command `./manage.py runserver --settings settings_debug` and I keep a settings_debug.py file sitting around that looks like this:

```
DEBUG = True
INTERNAL_IPS = ['24.xxx.xxx.xx']
MIDDLEWARE_CLASSES += ('debug_toolbar.middleware.DebugToolbarMiddleware',)
INSTALLED_APPS += ('debug_toolbar',)
```

This allows me to keep my normal (production) settings file from ever having DEBUG set to True. That way there's no way to run with it in production. The other things are just good easy way to maintain some stuff that is useful for debugging/testing, but you don't want to include in your normal production server.

Views

Wrapping generic views It's really easy to use generic views in Django. Sometimes you want to change a little functionality or what they return, so you think you have to write a whole new function. Malcolm goes into how to extend them, to save you some time.

Use RequestContext By default, when you render a template, you aren't given the request object. It's nice to have and really simple to make django give it to you.

```
from django.template import RequestContext
def index(request):
    return render_to_response('weather/index.html', {},
                  context_instance=RequestContext(request))
```

Templates

Use the {% url %} tag. Using the url tag allows you to make your templates portable and is a good way to implement DRY. Whenever the links in your view changes, your templates automatically update, and they always have the correct links.

Use Template Utils django-template-utils contain some really nice generic template tags and other goodies that make your life easier. From getting the latest X number of objects from a model, getting a random object from a module, or getting the last updated one; they provide you with a really nice generic way of extending template nodes and doing generic content tags really easy.

Use MEDIA_URL Django now comes with a Context Processor that gives you MEDIA_URL in your templates. Use this so that you can apply DRY to all of your external media Urls, like you did with the {% url %} tag for internal things.

Use a 3-level template hierarchy This is referenced in the 'Django docs <http://docs.djangoproject.com/en/dev/topics/templates/#id1>'_(about a page down). But it works really well to do a base.html, app-base.html, and then templates on top of that. This gives you a really nice way to contain site-wide, app-wide, and view-wide functionality inside their own little spaces.

Using template inheritance to extend itself This is a really neat trick when dealing with multiple template directories. It allows you to take most of a chunk of one template, and overwrite just a small part of it. They explain it better than I do.

Testing

Using the tests/ directory inside of your application you can define a tests.py that will hold tests. You can also define a tests/ directory that can hold tests. Inside the tests directory **init**.py you need to import all of your unit tests. Inside **init**.py you need:

```
from basic import *
from views import *
```

etc. Assuming your tests are named basic.py and views.py.

Watch Files This tip is useful for doing TDD. You can go ahead and watch the output of your test file and see when something changes based on the edits you're making to your files.

```
watch "python tests.py"
```

Nose tests Use nose tests. They have some neat auto-discovery tools and lots more. nose-django allows this to work with Django fixtures (note it may not work well yet). This would be nice if someone wrote a test runner in django for nose.

Mock objects Using mock objects to test is really handy. There are a couple of good mock testing libraries for python, and i show a simple way to do it here This allows you to try your code when it's interacting with things that are somewhat random (like times of day, random numbers, etc.)

Use testmaker I wrote an app that writes view tests for you. A little self promotion, but go ahead and check it out.

Want to do something a little different? You can define your own test runner and set it in the settings. Then you can tweak the way that django runs your tests for you. This is a lighter weight approach than using nose or something to run your tests, and is integrated with django, which makes it more portable.

Use testserver Django comes with the testserver command that allows you to load a fixture into the development server and run against that. This is really useful. It also leaves the database around after it's done, so you can inspect it. This can be really handy in debugging fixtures and tests.

A blog post a day keeps the doctor away

November blog posting month has a special moment in my Django history. It was this time last year that I really got serious into Django. With the help of James Bennett's and Marty Alchin's blog post a month streak, I got an incredibly valuable insight into Django. It showed me a lot of the power and other great things about Django (especially the community).

In that spirit, I figured that I would jump on the blog post a month bandwagon (that looks like it's going to be big in Djangoland this year). All of my posts won't be about Django, most of them will probably be about programming, or technical in nature. However, I promise nothing along these lines. Writing 31 posts back to back is quite a lot of work. So I might blank out somewhere halfway through for a day and resort to posting my philosophical musing that used to be most of this blog. You are thusly warned ;)

I mentioned that some other people in the community are doing post-a-day for the month. So expect to get your fill of awesome Django and Tech related content. Brian Rosner, Eric Florenzano, Justin Lilly, James Tauber, and Greg Newman will be trying this gargantuan task along with me!

Some of the things that you can look forward to in this month of blog posts from my side:

- At least 2 code releases. One is a new addition and release to testmaker, the other is another neat new testing tool.
- Pinax related material. They have their first release, and most of us posters are in that community.
- Screencasts. These are time consuming, but I have plans for at least a couple news ones over the month.
- A new design. The finishing touches are being put on this site as we speak. So there will be a new version launched sometime early this month.
- A newish approach to testing code that I have been thinking about for a while. Hopefully it isn't new and the Google-fu isn't strong with this one.
- Some talk about template tags and how to make them better
- Multiple viewpoints expressed on topics around the different people doing this.
- Lots more!

I hope that you all stayed tuned and even join in on these posts throughout the month. In talking about doing it, one of the big wins of doing it at the same time was to enable the separate view points on the same topic to be discussed. The point of blogs is to have a discussion, and I think it will be neat to see what kind of discourse we have throughout this month.

It looks like James Tauber has already posted his first functional combinatorial mind bender over on his blog already. Let the games begin!

Now on with regularly scheduled programming.

And because I feel bad that there isn't really any content in this post, here is a little tip:

Doing template testing is a pain sometimes because it suppresses your errors. You can load your templates on the command line to test them, and it will show you the errors if they are having any.

```
>>> from django.template.loader import get_template
>>> get_template('mine/index.html')
<django.template.Template object at 0xa833d0>
>>> get_template('mine/error.html')
Traceback (most recent call last):
 [Chopped a bunch off]
  File "/usr/lib/python2.5/site-packages/django/template/__init__.py", line 362, in find_filter
    raise TemplateSyntaxError("Invalid filter: '%s'" % filter_name)
django.template.TemplateSyntaxError: Invalid filter: 'wtf'
```

This is incredibly useful, and great for debugging templates without worrying about caching and other things like that. Also note that this isn't anything special used just for testing. This is actually the way to load templates inside of your code. Check out the docs for more. `select_template` is another really useful thing to know about, so check it out!

Announcing Django Crawler and django-test-utils

Today I'm going to be releasing a new project, called django-test-utils. It's rather empty at the moment, but it does have one cool feature. That is my Django Crawler. I have some big plans for this little guy, but for the moment it has enough functionality to make it pretty useful.

Usage

The crawler at the moment has 4 options implemented on it. I will outline them below and show example of the output. It is implemented as a management command, named crawlurls. You simply add `test_utils` to your `INSTALLED_APPS` and you are good to go. So to run it you simply do `./manage.py crawlurls`. It crawls your site using the Django Test Client (so no network traffic is required!) This allows the crawler to have intimate knowledge of your Django Code. This allows it to have features that other crawlers can't have.

Core features

The Crawler at the beginning loops through all of your URLConfs. It then loads up all of the regular expressions from these URLConfs to examine later. Once the crawler is done crawling your site, it will tell you what URLConf entries are not being hit.

-v –verbosity [0,1,2] Same as most django apps. Set it to 2 to get a lot of output. 1 is the default, which will only output errors.

-t –time The `-t` option, as the help says: Pass -t to time your requests. This outputs the time it takes to run each request on your site. This option also tells the crawler to output the top 10 URLs that took the most time at the end of it's run. Here is an example output from running it on my site with `-t -v 2`:

```
Getting /blog/2007/oct/17/umw-blog-ring/ ({}) from (/blog/2007/oct/17/umw-blog-ring/)
Time Elapsed: 0.256254911423
Getting /blog/2007/dec/20/logo-lovers/ ({}) from (/blog/2007/dec/20/logo-lovers/)
Time Elapsed: 0.06906914711
Getting /blog/2007/dec/18/getting-real/ ({}) from (/blog/2007/dec/18/getting-real/)
Time Elapsed: 0.211296081543
Getting /blog/ ({u'page': u'5'}) from (/blog/?page=4)
Time Elapsed: 0.165636062622
NOT MATCHED: account/email/
NOT MATCHED: account/register/
NOT MATCHED: admin/doc/bookmarklets/
NOT MATCHED: admin/doc/tags/
NOT MATCHED: admin/(.*)
NOT MATCHED: admin/doc/views/
NOT MATCHED: account/signin/complete/
NOT MATCHED: account/password/
NOT MATCHED: resume/
/blog/2008/feb/9/another-neat-ad/ took 0.743204
/blog/2007/dec/20/browser-tabs/#comments took 0.637164
/blog/2008/nov/1/blog-post-day-keeps-doctor-away/ took 0.522269
```

-p –pdb This option allows you to drop into pdb on an error in your site. This lets you look around the response, context, and other things to see what happened to cause the error. I don't know how useful this will be, but it seems like a neat feature to be able to have. I stole this idea from nose tests.

-s –safe This option alerts you when you have escaped HTML fragments in your templates. This is useful for tracking down places where you aren't applying safe correctly, and other HTML related failures. This isn't implemented well, and might be buggy because I didn't have any broken pages on my site to test on :)

-r –response This tells the crawler to store the response object for each site. This used to be the default behavior, but doing this bloats up memory. There isn't anything useful implemented on top of this feature, but with this feature you get a dictionary of request URLs with responses as their values. You can then go through and do whatever you want (including examine the Templates rendered and Contexts.

Considerations

At the moment, this crawler doesn't have a lot of end-user functionality. However, you can go in and edit the script at the end of the crawl to do almost anything. You are left with a dictionary of URLs crawled, and the time it took, and response (if you use the `-r` option).

Future improvements

There are a lot of future improvements that I have planned. I want to enable the test client to login as a user, passed in from the command line. This should be pretty simple, I just haven't implemented it yet.

Another thing that I want to do but isn't implemented is fixtures. I want to be able to output a copy of the data returned from the crawler run. This will allow for future runs of the crawler to diff against previous runs, creating a kind of regression test.

A third thing I want to implement is an option to only evaluate each URLConf entry X times. Where you could say "only hit /blog/[year]/[month]/ 10 times". This goes on the assumption that you are looking for errors in your views or templates, and you only need to hit each URL a couple of times. This also shouldn't be hard, but isn't implemented yet.

The big pony that I want to make is to use multiprocessing on the crawler. The crawler doesn't hit a network, so it is CPU-bound. However, running with CPUs with multiple cores, multiprocessing will speed this up. A problem with it is that some of the timing stuff and pdb things won't be as useful.

I would love to hear some people's feedback and thoughts on this. I think that this could be made into a really awesome tool. At the moment it works well for smaller sites, but it would be nice to be able to test only certain URLs in an app. There are lots of neat things I have planned, but I like following the release early, release often mantra.

Practical Django Testing Examples: Views

This is the fourth in a series of Django testing posts. Check out the others in my Testing series if you want to read more. Today is the start of a sub-series, which is practical examples. This series will be going through each of the different kinds of tests in Django, and showing how to do them. I will also try to point out what you want to be doing to make sure you're getting good code coverage and following best practices.

Instead of pick some contrived models and views, I figured I would do something a little bit more useful. I'm going to take one of my favorite open source Django apps, and write some tests for it! Everyone loves getting patches, and patches that are tests are a god send. So I figured that I might as well do a tutorial and give back to the community at the same time.

So I'm going to be writing some tests for Nathan Borror's Basic Blog. It also happens to be the code that powers my blog here, with some slight modifications. So this is a win-win-win for everyone involved, just as it should be.

Nathan's app has some basic view testing already done on it. He was gracious enough to allow me to publicly talk about his tests. He claims to be a designer, and not a good coder, but I know he's great at both. So we're going to talk about his view testing today, and then go ahead and make some Model and Template Tag tests later.

Basic philosophy

Usually when I go about testing a Django application, there are 3 major parts that I test. Models, Views, and Template Tags. Templates are hard to test, and are generally more about aesthetics than code, so I tend not to think about actually testing Templates. This should cover most of the parts of your application that are standard. Of course, if your project has utils, forms, feeds, and other things like that, you can and should probably test those as well!

Views

So lets go ahead and take a look to see what the tests used to look like. He has already updated the project with my new tests, so you can check them out, and break them at your leisure.

```
"""
>>> from django.test import Client
>>> from basic.blog.models import Post, Category
>>> import datetime
>>> from django.core.urlresolvers import reverse
>>> client = Client()

>>> response = client.get(reverse('blog_index'))
>>> response.status_code
200
>>> response = client.get(reverse('blog_category_list'))
>>> response.status_code
200
>>> category = Category(title='Django', slug='django')
>>> category.save()
>>> response = client.get(category.get_absolute_url())
>>> response.status_code
200

>>> post = Post(title='My post', slug='my-post', body='Lorem ipsum dolor sit amet', status=2, publis
>>> post.save()
>>> post.categories.add(category)

>>> response = client.get(post.get_absolute_url())
>>> response.status_code
200
"""
```

Notice how he is using reverse() when referring to his URLs, this makes tests a lot more portable, because if you change your URL Scheme, the tests will still work. A good thing to note is that a lot of best practices that apply to coding apply to testing too! Then the tests go on to create a Category, save it, and then test it's view and get_absolute_url() method. This is a really clever way of testing a view and a model function (get_absolute_url) at the same time.

Next a post is created, and saved, then a category it added to it, the one created above. That is all that these tests do, but it covers a really good subsection of the code. It's always good to test if you can save your objects because a lot of bugs are found in that operation. So for the length of the code it is remarkably well done.

This is a pretty simple test suite at the moment. But the fact that he has tests is better than 90% of other open source projects! I'm sure if we asked Nathan, he would tell us that even this simple test suite helps a ton. Most of the bugs people make break in very loud and obvious ways. Which just goes to emphasize my point that everything should have tests, even if they're simplistic.

So how are we going to improve this testing of views? First of all, note that these tests are hardly touching models, and not testing any template tags; this will be addressed later. In regard to views, these tests aren't checking the context of the responses, they are simply checking status code. This isn't really testing the functionality of the view, just testing

if it doesn't break. There are also some views that aren't being touched, like search, pagination, and the date archive views. We aren't going to test pagination because we don't have enough data to paginate. This brings me to a meta subject, slight tangent time.

Do I test Django stuff?

So we have some Generic views in our application, should we test them? I don't think that there is a correct answer to this question, but I have an opinion. I think that you should test generic views, but only in the ways that they can break based on how you define them. This doesn't look much different than normal tests that you should be writing anyway.

For the date-based generic view for example, you are passing in a QuerySet and DateField in the URLConf; and the parts of the date you're using in the actual URLs. So what is the easiest way to test that all of these things are being set correctly? Find the most specific example and test for it. So you would test the context and response code of blog_detail page, because it has to use the query set, the date field, and the full path for the URLs. Assuming that your code isn't broken in some horrible way, that means that all the other parts of the date URLs should work.

Let's write some tests

So we need to add some stuff to the tests. We need to get some data into the tests, in order to use the date-based archives, and search stuff. So we're going to take the stuff that was previously at the bottom of the test, and move it up to the top. Also need to add 2 posts and categories, so that we know that our filtering functionality is working.

```
>>> category = Category(title='Django', slug='django')
>>> category.save()
>>> category2 = Category(title='Rails', slug='rails')
>>> category2.save()
>>> post = Post(title='DJ Ango', slug='dj-ang', body='Yo DJ! Turn that music up!', status=2, publish=
>>> post.save()
>>> post2 = Post(title='Where my grails at?', slug='where', body='I Can haz Holy plez?', status=2, pu
>>> post2.save()
>>> post.categories.add(category)
>>> post2.categories.add(category2)
```

Pretty obvious what this test is doing. If these tests were going to be much more complicated than this, it would make a lot of sense to write a fixture to store the data. However I'm trying to test the saving functionality (which is technically a model thing), so it's good to make the objects inline.

So now we have our data, and we need to do something with it. Let's go ahead and run the test suite to make sure that we haven't done anything stupid. It's a tenet of Test Driven Development to test after every change, and one that I picked up from that philosophy. It's really handy. I don't do it on a really granular level like it suggests, but I try to do it after any moderately important change.

Getting into context

So we have the tests that were there before, and they're fine. They perform a great function, so we should keep them around, we just need to add some stuff to them. This is one of the reasons I really don't like doctests. Using unit tests you can just throw an `import pdb; pdb.set_trace()` in your code and it will drop you into a prompt, and you can easily use this to write new tests. Doctests however hijack the STDOUT during the tests, so when I drop into pdb with a `>>> import pdb; pdb.set_trace()` in the test, i can't see the output, so it's hard for me to get testing information.

Note: You can also do this by changing your settings file database (because otherwise these objects would be created in your real DB), running syncdb, running `s/>>> //` on your test, adding a setup_test_environment() import and call to the test, and running `python -i testfile`, if you want. But do you really want to do that?

Let's go poking around inside of response.context, which is a dictionary of contexts for the response. We only care about [-1], because that is where our context will be (except for generic views, annoying right?). So go down to the first view, `blog_index`, and put

```
>>> response = client.get(reverse('blog_index'))
>>> response.context[-1]['object_list']
[test]
```

In your tests. We know [test] won't match, but we just want to know what the real output is. When you go ahead and run the tests your should find some output like this:

```
Expected:
    [test]
Got:
    [<Post: DJ Ango>, <Post: Where my grails at?>]
```

So go ahead and put in the correct information in where [test] was. This is a really annoying way of testing, and I'm going to repeat that this is why doc tests are evil, but we're already this far, so let's push on. Writing tests this way requires the tester to be vigilant, because you're trusting that the code is outputting the correct value. This is kind of nice actually, because it forces you to mentally make sure that your tests are correct, and if you're code isn't outputting what you expect, then you've already found bugs, just by writing the tests ;) But if you're testing code that's complex, this method breaks down, because you don't know if the output is correct!

If you look in the context, you'll see lots of other things that we could test for as well. Some that Django (oh so nicely) gave us, and other stuff that is user defined. Things like pagination, results per page, and some other stuff that we really don't care about. The object_list on the page is really what we're after, so we can move on. Run your tests to be sure, and lets move on.

Updating current tests

Now that we have our hackjob way of getting data out of the tests, we can move on to writing more tests. Go down to the next view test of `blog_category_list`, and pull the old object_list trick. You should get the following back out once you run your tests:

```
Expected:
     [test]
 Got:
     [<Category: Django>, <Category: Rails>]
```

This looks correct, so lets go ahead and put that in the test. As you can see, for this simple stuff, it isn't really a huge deal doing testing this way. The test suite runs in about 3 seconds on my machine, so it's not a huge hurdle.

Let's go ahead and do it for the category and post detail pages. When I don't remember or don't know what variables we'll be looking for in the context, I usually just put `>>> request.context[-1]` to output all of it, and see what it is that I want. For the `category.get_absolute_url()` we need `object_list` again. For the `post.get_absolute_url()` we just want `object`.

```
>>> response = client.get(category.get_absolute_url())
>>> response.context[-1]['object_list']
[<Post: DJ Ango>]
>>> response.status_code
200

>>> response = client.get(post.get_absolute_url())
>>> response.context[-1]['object']
<Post: DJ Ango>
>>> response.status_code
```

We can consider those views tested now.

Creating new tests

So now we've improved on the tests that were already there. Let's go ahead and write some new ones for search and the date-based views. Starting with search, because it will be interesting. Search requires some GET requests with the test client, which should be fun.

```
>>> response = client.get(reverse('blog_search'), {'q': 'DJ'})
>>> response.context[-1]['object_list']
[<Post: DJ Ango>]
>>> response.status_code
200
>>> response = client.get(reverse('blog_search'), {'q': 'Holy'})
>>> response.context[-1]['object_list']
[<Post: Where my grails at?>]
>>> response.status_code
200
>>> response = client.get(reverse('blog_search'), {'q': ''})
>>> response.context[-1]['message']
'Search term was too vague. Please try again.'
```

As you can see, we're testing to make sure that search works. We're also testing the edge case of a blank search, and making sure this does what we want. A blank search could return everything, nothing, or an error. The correct output is an error, so we go ahead and check for that. Notice that you pass GET parameters in the test client as a dictionary after the URL, and passing them as `?q=test` on the URL wouldn't work. Russ is working on fixing that, and by the time you read this, it might not be true.

Next, on to testing the generic date views. You should be in the hang of it by now.

```
>>> response = client.get(reverse('blog_detail', args=[2008, 'apr', 2, 'where']))
>>> response.context[-1]['object']
<Post: Where my grails at?>
>>> response.status_code
200
```

Notice here that we're using the args on reverse, and not using get parameters. We're passing those arguments as positional into the view. You can also use kwargs={'year': '2008'} if you want to be more explicit. As talked about above, I feel that this is enough of testing for the generic views.

Wow! That was a long post. I'm glad I decided to split the testing up into separate posts! I hope this has been enlightening for everyone, and I'm sure that I'm doing it wrong in some places. I would love some feedback, and to hear how you work around and solve some of the problems above. Also your thoughts on this kind of stuff.

Nathan has graciously included my new tests in his project, if you want to see them live, or check them out.

The importance of not deleting blog posts (read: ideas)

A lot of the time I start a blog post as a sentence. It is something that strikes me and I don't really know what I think about it. It is a moment of thought that needs to be revisited, but cannot yet be expounded upon.

A while ago I went through all of my un-published blog posts in an effort to delete them. I read the titles and thought that there was very little that I had to think or say about the topics. (The beauty of django not letting you delete things en masse is that I had to go into the post to delete it). Once I went into the post to try and delete it, I found it nearly impossible. I realized that I had been using my unfinished blog posts as a kind of note pad of ideas, or a place where I just brain dumped when I felt the need.

The other crazy thing is that some of these posts had the ideas more thought out that I previously thought. I had written a lot in that space, and promptly forgotten about it. There is nothing quite like remembering writing something that you forgot you knew.

The next question was what to do with all of these half finished ideas, thoughts, and half-quack brained theories. What better than to publish them? The value of an idea is very little. It gains a lot of value when you are forced to write it out. It gains even more insight when you appreciate that it will be read by others. There is a progression that these texts go through that allows them to be better than they were before.

The simple act of writing an ideas clarifies it. The simple act of publishing an idea betters it. The feedback that is gained from the public exposure is more or less external to the idea. The idea will get better once there is feedback, but getting to the point where there is feedback is the key. Once you have an idea 'polished' enough for public consumption, it has gone almost beyond being an idea. You have hopefully thought it out to a logical conclusion, because you are going through, and putting yourself in the heads of others.

This isn't how all posts work, or even most. But if you look at things this way, it is a real driver to putting things out there and not keeping them in. A little extrinsic motivation if you will. The fact that your idea is getting spread to others to do with what they please, means that you want your idea to be in the correct form for that to happen, and that you want it to take those people to the right place.

The other awesome gain from posting things is indeed the feedback you get. A lot of people release software and it gets used in ways that nobody ever intended it to be used. Having as many eye balls looking at something is the best way to achieve something awesome.

All of the above applies to code as well. Also, don't take this post as an example of the above, it could be written better, but i wanted to get the idea out there. It's better than it was yesterday ;)

Encouraging Testing in Django

I was having a conversation with Jacob tonight about testing in Django. He has shot down testmaker for being too specific for Django core, which I almost agree with, given my grandiose plans for it before the month is out. I'm quite okay with it staying a third party app for a little while longer.

However, that got us on the topic of testing, and I think it's interesting enough to post here to get some feedback and to tell people what's up. First we talked about trying to stub out some tests for people in the startapp command in Django. Like Rails does, except there is a really hard question about what to provide in that file. Should we provide a simple test that passes and makes people feel good about testing? Do we be evil and provide a test that fails, with `assert False, 'Write some tests yo!'`?

So the idea then progressed into perhaps having a command that can be called later in the process to stub out your tests files with real data. Perhaps stubbing a test for each view in your URLconf and each method in your Models or something like that.

We also talked about the possibility of adding a fifth part to the Django Intro Tutorial about testing. I think that this is a really great idea, and would help further the testing culture inside of the Django community. I volunteered to write the first draft of that document, so expect that to be posted to this blog sometime next week.

So I'm just kind of curious what people think is a good way to get testing integrated into the Django community better. I am trying to write some tools that will help people write tests, which would help them have tests :). But I think that there is a lot more that can be done to get people thinking about testing their applications.

Should we be encouraging people to be testing from when they start a new application in Django? If so, what should we put in the tests.py file when they create an application? Should we just stub out an empty tests.py file to remind them that they should be writing tests? Should we be pushing best practices from the beginning in that form, or giving people a builtin option to perhaps then stub it out later?

I think that the Django community is lacking in the testing realm these days, and I'm curious what we can be doing to get more people **excited** about testing. It's a great tool, and something that everyone should be doing. So I'd love to hear feedback or ideas about what people think can and should be done with regards to testing.

Should reusable apps have templates?

There is a debate among the Django community about whether people should include templates in their reusable apps.

The arguments for including them are generally that it is nice to simply install the app and have things just work. This is a really nice feature to not have to dig through the code looking for template names and context variables. Then creating your own templates for code you didn't write. I am usually of this persuasion, because it's really annoying to have a big up front cost to begin using an app.

I think that an app that comes with no templates is going to be used less. Some people might think that it a good thing, but I think it is better for an app to be used as much as possible. I know that I personally have tried to check out an app, and given up because there are no templates. It is a pretty sizable mental hurdle (and a decent bit of work) to get some template to even test the functionality. Sure, you can look at the code, but that isn't nearly as quick and easy as simply plugging the app in a testbed and seeing what happens.

The argument against it are that there is no possible way to know how your users are going to use your templates. They will file bugs against the default templates because they don't work the way they want them to. I understand this point of view as well. There is no possible way to ship a set of default templates that will work for everyone, every time. So then the question is, what do you do?

I think that there is a nice middle ground that we can achieve. Reusable apps should ship with templates, but they shouldn't be in a place where they are run by default. I say that they should ship in the docs/ folder. This way they are seen as a reference implementation. A user doesn't automatically get them to run, and if they move them out of the docs folder, then they know it is a reference implementation instead of something that is meant to work for them every time.

I think that this solved the problem by letting people evaluate apps with templates that show it's basic functionality. But it puts a slight barrier between seeing them as templates that should work for me. They should be seen as a reference, and as so they can go in the docs/ directory. Of course, if you want to ship your templates in the templates/ dir inside the app, that is cool. But I think that apps that have views, should have templates to go along side them, even if they're not 'installed' by default.

Yes, this is a half assed post. I was going to write a longer one, but there is a free Robert Randolph show in Lawrence tonight. This is scheduled to post at 11pm, so if I don't have time to post, you will get this awesome half assed one ;)

Debugging Django in Production Environments

Nick had a nice post about setting DEBUG based on the hostname of the server that you're site is running on. This allows you to set DEBUG to True for your staging site, and False for your production site.

I do something along those lines, but a little bit differently. I can't take credit for this idea, because it came from this snippet. It is a really neat trick, that I have expanded on a little bit.

```
from django.views.debug import technical_500_response
import sys
from django.conf import settings

class UserBasedExceptionMiddleware(object):
    def process_exception(self, request, exception):
        if request.user.is_superuser or request.META.get('REMOTE_ADDR') in settings.INTERNAL_IPS:
            return technical_500_response(request, *sys.exc_info())
```

Now simply save this in a file somewhere. Add it to your MIDDLEWARE_CLASSES, and you are good to go. For example, mine looks like:

```
'tools.middleware.superuser.UserBasedExceptionMiddleware',
```

This is a pretty simple middleware that is crazy useful. When you throw this inside of your site, it will give you a normal Django error page if you're a superuser, or if your IP is in INTERNAL_IPS.

This makes it really nice, because you can get an error message on your production servers, where your normal users get your normal pretty 500 pages. This makes debugging things that are showing up in production, but won't be reproduced on the staging server possible. Caching behavior is a big one that I know isn't tested when you are using DEBUG = True. This lets you keep DEBUG = False on, but gives you some nice error pages.

Hope this tip is useful.

A start to the uber community

Well I spent all day Saturday, and all night. Into Sunday morning hacking on some code. Probably the most productive 24 hours of my life. I have a couple of announcements, but in the spirit of post-a-day, I'll spread them out over a couple days :).

The big one, is that I basically wrote an entire Django application last night. You can check it out over here. It is basically an aggregator for the Django Community Feed. I seeded the data with the people who were on Django's Community Feed, and then I grabbed as much of their social networking information as possible.

All of this code will be shown at some point over the next couple of days. However, I don't have the energy to write it up quite yet. However, please play around with the data, there's some really neat possibility in there for something cool...

At the moment there is..

- A river of data for every user, all of their services
- A river of data for every user, for each service
- A river of data for each service for all users combined
- A river of data for all services for all users combined
- Atom feeds for all of the above (it says RSS but I lied)
- A permalink to every item a user has
- A profile page, listing each users username at each network that I could find.

Feeds should be updating hourly, but I haven't tested them. This model is currently pulling data, and will always have to for some data. However, I plan to be open for pushing as much of this data as possible. Most people have tumblelogs on their sites, so it would be pretty easy for them to push all their data in, with more metadata than I can apply from outside.

I'd love to hear some comments, feedback, and ideas for future directions. With a solid OpenID server and distributed identity, I think that we could really make something special.

Note: I also added the ability to combine (only) 2 tags in the user and everyone views. See http://ericholscher.com/django/river/friendfeed+twitter/ and http://ericholscher.com/django/profile/Eric%20Holscher/friendfeed+twitter/

PPS: If your info isn't showing up in your page. Add some rel="me" links to your profiles on your blog's homepage. Person profiles get updated daily.

Busy Busy

So at work since I started for Mediaphormedia, currently World Online, and the birthplace of Django, I have been tasked with porting Ellington. Ellington is the CMS that we create and sell, and is what Django originally was. Django was pulled out of Ellington and Open Sourced.

So when I got here, my first job was tasked with porting Ellington. We have a version running on Revision 1290 and .91. Aka, really damn old, pre-magic removal old. We were tasked with porting this to Django 1.0 (which is around

revision 9300). That is a whole 8000 source code revisions, and 600% more revisions than the base. That is a whole lot of code.

So today at work, we are launching our 3 main sites; Ljworld, KU Sports, and Lawrence.com. These will be running all on Django 1.0, which is a monumental task.

That is the reason today's post is short. Was bug fixing all day, and I have to be up at 5am to go in and put out the fires that will surely happen once users start hitting the site.

I will have an updated post afterward about the experience.

Introduction to Python/Django testing: Basic Doctests

This is the first in a series of blog posts and screencasts that will walk you through how to test your Django application. These posts will focus more on how to get things done in Django, but note that a lot of the content is applicable to pure python as well. A lot of best practices are codified into Django's testing framework, so that we don't have to worry about them! I will try to point them out as we are using them through, because they are good things to know.

The currently planned structure for this series is below. Please comment if there is something that you think is missing, or something that I shouldn't do. This is subject to change, a lot, as well, so your feedback will help direct it. Also note that most or all of this content is available in the Django and Python documentation, and I will try and point there and not re-invent the wheel. I hope that these posts will take a more practical look, and try to point out some pit falls and other things that are useful.

Outline

- Basic Doc tests
- Basic Unit tests
- Real examples of both types
- Comparison of Unit tests vs. Doc test
- Fixtures
- Using Mock objects
- Third party testing tools
- Writing your own test runner
- Getting code coverage for your tests

Where to start I'm assuming that you already have a project that you're working on that you would like to test. There are two different ways of putting tests inside of your django project. You can add a tests.py file and put your tests inside of there. You can also define a tests/ directory and put your tests in files inside of that. For these tutorials it is assumed that the second is the way things are done. It makes it a lot easier when you can break your tests out into logical files.

Doctests These can go in two places inside your django project. You can put them in your models.py file, in the Docstring for your modules. This is a good way to show usage of your models, and to provide basic testing. The official docs have some great examples of this.

The other place your Doctests can go is inside your tests directory. A doctests file is usually pretty simple. A doctest is just a large string, so there isn't much else to put besides a string. Usually you want to use the triple quote, multi-line string delimiter to define them. That way your " and 's inside of your doctests don't break.

```
"""
This is my worthless test.
>>> print "wee"
wee
>>> print False
False
"""
```

You can go ahead and put that in a file in your `tests/` directory, I named it `doctst.py`. I didn't name it doctest, because of the python module with the same name. It's generally good to avoid possible name overlaps. My application that I'm writing tests for is `mine`, because it's the code for my website. Make sure that directory has an `__init__.py` as well, to signify that it is a python module.

Now here is the tricky part; go ahead and try and run your test suite. In your project directory run `./manage.py test APPNAME`. It will show you that you have passed 0 tests. 0 tests? We just defined one.

You need to go into your `__init__.py` file and put some stuff in there.

```
import doctst
__test__ = {
    'Doctest': doctst
}
```

You are importing the doc test module and then adding it to the `__test__` dictionary. You have to do this because of the way that python handles looking for doc tests. It looks for a `__test__` dictionary inside of your module, and if that exists it looks through it, executing all docstrings as doctests. For more information look at the Python docs.

Now you should be able to go ahead and run the tests and see the magical `Ran 1 test in 0.003s OK` that all testers live for. This is little bit of overhead really threw me off when I was trying to break my tests.py out into the tests/ directory. Notice that the doc test runner sees all of your tests as one single test. This is one annoying thing that the doctests do.

So now we have a test suite that is worthless, but you know how to use doc tests. If you didn't notice, the doctest format is simply the output of your default python shell, so when you are testing your code on the command line and it works, you can simple copy and paste it into your tests. This makes writing doc tests almost trivial. Note however, that they are somewhat fragile, and shouldn't be used for everything. In the next segment, we will talk about unit tests. Then we will compare the two and see what the use cases are for each.

Python gems of my own

Note: I'm launching a redesign today to address the styling issues. Please bear with me

A great example of how this month of blog posting is spawning great content on the interwebs. Other Eric posted a gems of python post, in which he pointed out some of the neat functions that he uses. The python stdlib has a ridiculous amount of really really useful things inside of it, and it's hard to know what even exists there. I love posts like that, that point to some neat little utility functions and tricks that make things really nice. In that spirit, here is my own list of Python Gems

1. urlparse

urlparse is a really handy piece of functionality if you are trying to deal with URLs on the web. As per usual, an example shows it best.

```
>>> from urlparse import urlparse
>>> urlparse('http://www.ericholscher.com/example/dir/')
('http', 'www.ericholscher.com', '/example/dir/', '', '', '')
>>> urlparse('http://www.ericholscher.com/example/dir?query=r0x0r#awesome-part')
```

```
('http', 'www.ericholscher.com', '/example/dir', '', 'query=r0x0r', 'awesome-part')
>>> parsed = urlparse('http://www.ericholscher.com/example/dir?query=r0x0r#awesome-part')
>>> parsed.path
'/example/dir'
>>> parsed.scheme
'http'
```

I was looking for a good way to check for relative versus absolute urls, and this made it really easy. Also incredibly easy to check for the type of link (http, https, mailto, ftp). You can access the data via the named patterns or as a list.

2. inspect

The entire inspect module is incredibly useful. I was looking through the Django source, which is where I stumbled upon it. It is used inside the simple tag code for Django templates. `getargspec` is really handy, but there is a lot of really interesting stuff in that file. I don't have anything that I can show quite yet, but I'm going to use the inspect stuff in an upcoming snippet. Here is a simple use case.

```
>>> import inspect
>>> def test(a, b=True, *args, **kwargs):
...     pass
...
>>> inspect.getargspec(test)
(['a', 'b'], 'args', 'kwargs', (True,))
>>> import django
>>> inspect.getmodule(test)
<module '__main__' (built-in)>
>>> inspect.getfile(django)
'/Users/ericholscher/Sites/EH/django/__init__.pyc'
>>> inspect.getsourcefile(django)
'/Users/ericholscher/Sites/EH/django/__init__.py'
>>> inspect.ismodule(django)
True
>>> inspect.isbuiltin(django)
False
```

As you can see, there is lots of really nice stuff in there.

3. Generator expressions

These are very similar to list comprehensions, except they are evaluated lazily. They are described "as a high performance, memory efficient generalization of list comprehensions and generators". The only difference in syntax is that you use a () instead of a [] around the comprehension.

```
>>> iter = (x for x in range(1,5))
>>> reg = [x for x in range(1,10)]
>>> iter
<generator object at 0x29e3c8>
>>> reg
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> iter.next()
1
>>> iter.next()
2
>>> for x in iter:
...     print x
...
```

```
3
4
```

The main use case that I have seen for this is if your list comprehension is going to take a lot of memory. If you aren't sure you're going to need all of it, or don't want to store it all in memory, then you can use the iterator. It will then get generated on demand when you need it. If you want to create your own generator, you simply use the yield keyword instead of return. Python makes this really easy!

4. 128

Note: Some commenters pointed out that this is also re.DEBUG. James Tauber has a nice post explaining it more in depth.

I don't know if this is documented in the Official python documentation, but it is an incredibly useful regex debugging tool. You can pass in 128 to your re.compile() function and get the parse tree back out! Really neat, check it out:

```
>>> import re
>>> re.compile('(\w+): (<.*?>)', 128)
subpattern 1
  max_repeat 1 65535
    in
      category category_word
literal 58
literal 32
subpattern 2
  literal 60
  min_repeat 0 65535
    any None
  literal 62
<_sre.SRE_Pattern object at 0x29f278>
>>> re.compile('Ahoy Globe', 128)
literal 65
literal 104
literal 111
literal 121
literal 32
literal 71
literal 108
literal 111
literal 98
literal 101
<_sre.SRE_Pattern object at 0x267920>
```

Isn't that neat?

5. enumerate

enumerate is very similar to the zip function that Eric talked about in his post. It is useful in those cases where you want to know the index of something in a list, but don't want to do i += 1.

```
>>> buddy_list = ['frank', 'liza', 'bob']
>>> for love, person in enumerate(buddy_list):
...     if love > 1:
...             print "%s is not loved" % person
...     else:
...             print "I love %s" % person
```

```
...
I love frank
I love liza
bob is not loved
>>> for place, person in enumerate(buddy_list):
...     print place, person
...
0 frank
1 liza
2 bob
```

That's it for today. As Eric said (not talking in the third person), there are lots of little awesome hidden corners of Python. I'd love to hear about the things that you find really useful.

Gentlemans agreement on Django templates

There are a lot of reusable apps out in the Django Ecosystem. I wrote a previous post about why I think that reusable apps should come with templates. There is a problem about distributing templates that I want to address with this post: the problem of Django Template Block names.

There are already some unwritten conventions in the community in regards to block names, and I think that talking about it will help. I don't think that we're going to be able to come up with a way that everyone will follow, but I think it would be nice if we could create a way to easily redistribute templates.

The main reason that I have been thinking about this is because of Pinax, they use some different template block name than my apps. So in order to use PInax and my app, I needed to change all of the blocks of my templates! That is a lot of work that could have been avoided by some standardization.

There are a lot of different ways to think about the content of a page, but I'm going to propose some basic template blocks that most pages will have, and then talk about some more 'extended' blocks that might be useful.

Blocks we need.

{% block title %} This will be the block where you define the title of the page. Presumably your base.html will define your Site's name (perhaps even using the Sites framework) outside of this tag, to be places on all pages.

{% block extra_head %} I think that this is a good one that most people are already using in some form or another. In your base template you have a set of things in your `<head>` that every page will have. However, a lot of other pages need things that they also want to put in the head of a document, like RSS feeds, Javascript, CSS, and all the other things that should go in the head. You can and probably will have other specialized blocks (like title above) that will fill in other parts of the head too.

{% block body %} This tag will be placed around the entire body section of the page. This allows you to create pages in your app that replace the entire page, not just the content. This won't be used a lot, but it's a really handy tag to have when you need it. If you haven't noticed, I've been trying to keep tag names consistent with their html tag names whenever possible.

{% block menu %} This would be where your menu goes. It would be the site-wide navigation, and not a per-page type of navigation.

{% block content %} This will be the place where you define the content on a page. This will presumably change on every page. It will not include any site navigation, headers, footers, or anything else that would belong in a base template.

Other possible blocks

{% block content_title %} This would be where the "title" of a content block would be. It includes the title of a blog. It can also include some kind of navigation between content, or other things like that. Presumably something that isn't the main pages content. I don't know if this should go inside the `content` tag and have a `main_content` as opposed to the `content` tag proposed above.

{% block header %}{% block footer %} Any text area in the header of footer that might change on a page-by-page basis.

{% block body_id %}{% block body_class %} This would be used to set the class or id of the body tag in the document. This is useful to set for styling and other properties.

{% block [section]_menu %}{% block page_menu %} This would be opposed to the `menu` block proposed above. It would be for the section or page.

Edit: Updated back to include _'s. Because I think thats more pythonic and looks better. The Django Admin isn't meant to be a reference implementation of the templates.

A lot of these ideas have been taken from Nathan and his base.html for basic-blog. I'm sure that he and Christian have put way more thought into this than I have. I'm just curious what people think, and if there's something crazy that I have missed..

Luck and a New Life in Lawrence

Note: This isn't a technical post. If you don't want to be getting posts like this, you can sign up for just my Django feed. This is my personal blog, so stuff like this pops up from time to time :).

Most people believe in luck. If you believe in luck, presumably the chance of luck happening is 50/50 on the side of good or bad luck. Each theoretically has an equal chance of happening, if luck is an abstract quantity.

However, a big part of luck is being in the correct position to embrace it. There are a lot of lucky breaks that people have encountered that they have said 'no' to. Even more where they could have had an incredibly lucky thing happen but it just wasn't right. There are a lot of people who say that they have bad luck, but perhaps they aren't setting themselves up to be able to be lucky.

To take a personal example, I feel incredibly lucky to be where I am at the moment. There is not a lot of luck involved in it really; I did a lot to get where I am. First off, I had to do a lot of work by myself in college. Nobody taught classes in Django or anything like that. I was incredibly lucky (in the real sense) to have an adviser and professor who was amazing. He taught a Perl/Python class, an Ajax class, and lots of other great classes that have contributed to my knowledge. But if I hadn't taken those classes and seized the opportunity to learn about web development, I wouldn't be here.

The next big step was moving to Kansas! From Virginia, by the ocean, having lots of friends, working in the same town I went to college in, for a Python shop. That was the offer that I received after college...Or move to Kansas and work for a Newspaper of all things! Doing tech at a newspaper, how horrible...But this was no ordinary newspaper, and this wasn't Kansas. It was Lawrence, a long time stronghold for liberals since the Civil war. It was the newspaper that invented Django, and was one of the first to integrate the newsroom. A great place to work and a great place to live.

Plus I had to be in a place in my life where I could up and move. Had I had a girlfriend or some other commitment then I probably would have stayed. There are a million things that could have kept me from achieving what I know I should. From fear of the unknown, being a thousand miles away from any family and a beach, to many other things. But I knew it was the right thing to do, for my career, and for my personal growth. That was it, and it was decided, but

it certainly wasn't easy, and a miraculously lucky event didn't just fall into my lap. It was brooded over many a night and weighed against the easy and obvious. Who knows what kind of Luck I would have had, had I chosen the other path?

Speaking of moving to Lawrence, it's amazing how a change in life (moving, new job, new friends, new everything) sends one's mind thinking into the depths of existence. Ever since ending up in Lawrence without any old friends and lots of time to think, I've been re-evaluating a lot of things in my life, and my mind has been working overtime.

Upon dramatic change in life simple assumptions no longer hold true. I get to choose my entire life all over again. Do I want to keep the same personality, same type of friends, same type of lifestyle; or do all of these get changes (for the better, or for the worse). Who knows, but it will be a hell of a ride trying to figure it all out. And I'm going to enjoy every second of it, because that's all I can do.

I have a great new group of coworkers that I have been spending a good amount of time with. Good people, and they have a lot of similar interests. My co-workers have an amazing depth and breadth of geek knowledge, while managing to maintain a social ability that is above par for a lot of geeks. It is quite impressive and says a lot about the town and people of Lawrence that they are able to keep such quality programmers employed, inventive, happy, and productive.

I think that progressive places have always been the breeding ground of great ideas. The company that you keep says a lot about you, and a town with such progressive and forward-thinking ideas and ideals about the way things should be; allows us to simmer in the creativity that abounds in many forms.

I was having a conversation with a new-found friend the other day. He was talking about the brilliance of the people in Lawrence, about how you can't think less of people. A simple example was that when we were eating he mentioned the server of our food was an amazing guitarist. In fact, we talked to him about it and apparently he was playing that night at a venue downtown.

In most situations and other places that I have lived, I tend to assume the worse of people. "People tend to suck" is one of my phrases that I say. Lawrence really gives me an appreciation of the ability of my common man, and the fact that they are impassioned and striving for something (anything) is highly inspirational. Surrounding yourself with people that are passionate about anything is a great way to develop passion within yourself.

Is it lucky that I've ended up in such a cool place. Or is it because it is such a cool place that I moved halfway across the country to live and work here? These are the kinds of things that I have been pondering as of late...

Now back to your regularly scheduled geekery.

Django Aggregator v2 now has tagging, and you should too.

I have been doing some more work on my Django Community Aggregator / Django People v2 project. A big feature that I want to incorporate is tagging. I want people to be able to sort data by tag, among other things. I think that this is a pretty killer feature.

This allows someone to say "I want to get all of the data about testing or debugging that the Django community is doing". However, if nobody is tagging their posts, then only services that provide tags will be available in those views. A lot of people are using django-tagging on the backend of their blogs, but they just aren't exposing that data in feeds.

Note: Yes I know the data can't be edited yet (on the aggregator). That is because it is a project that is just living on my site for the moment. Once it gets moved off and the Uber community of Django gets more off the ground, all of those issues will be solved.

Luckily, it is really easy to expose your tagging data in your Django feeds. This assumes you are using Django Tagging, however, it's really easy with anything else.

Say we have our feed class that looks like this.

```
class BlogPostsFeed(Feed):
    title = 'My awesome blog'
    description = 'My awesome blog"
```

```
    def link(self):
        return "http://mysite.com"

    def items(self):
        return Post.objects.published()[:10]

    def item_pubdate(self, obj):
        return obj.publish
```

That is a pretty basic feed, but much akin to what most people have. Now lets add some tagging in there! Assuming that you have the Tag model imported from tagging, you can simply do this:

```
def item_categories(self, obj):
    return [tag.name for tag in  Tag.objects.get_for_object(obj)]
```

You can test to make sure that your feeds are working by using feedparser

```
In [1]: import feedparser

In [3]: p = feedparser.parse('http://ericholscher.com/feeds/posts/')

In [4]: p.entries[0].tags
Out[4]:
[{'label': None, 'scheme': None, 'term': u'lawrence'},
 {'label': None, 'scheme': None, 'term': u'mediaphormedia'},
 {'label': None, 'scheme': None, 'term': u'philosophy'},
 {'label': None, 'scheme': None, 'term': u'post-a-day'},
 {'label': None, 'scheme': None, 'term': u'ramblings'}]
```

That's it! Now everyone go do that to their feeds, so that I can harvest your tags and make them useful :)

Testmaker 0.2: Rewritten and improved

About a week ago, I went ahead and re-wrote testmaker and moved it into my django-test-utils project on github. The syntax is now a bit different, and the whole thing is much improved. This is version 0.2. The **'screencast <>'_** from the last release still shows the gist of the project, except for the changed syntax.

Also note that my projects have permanent pages for documentation over at my projects page. This will stay up to date with the most current version of the software, and basically be a copy of this post for now.

Testmaker

What is does Django testmaker is an application that writes tests for your Django views for you. You simply run a special development server, and it records tests for your application into your project for you. Tests will be in a Unit Test format, and it will create a separate test for each view that you hit.

Usage Step 1: Add `test_utils` to your INSTALLED_APPS settings.

Step 2:

```
./manage.py testmaker APP
```

This will start the development server with testmaker loaded in. APP must be in installed apps, and it will use Django's mechanism for finding it. It should look a little something like this:

```
eric@Odin:~/EH$ ./manage.py testmaker mine
Handling app 'mine'
Logging tests to /home/eric/Python/EH/mine/tests/mine_testmaker.py
Appending to current log file
Inserting TestMaker logging server...
Validating models...
0 errors found

Django version 1.0.1 final, using settings 'EH.settings'
Development server is running at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Then, as you browse around your site it will create unit test files for you, outputting the context variables and status code for each view that you visit. The test file used is in `APP/tests/APP_testmaker.py`. Once you have your tests written, you simply have to add them into your `__init__.py`, and then run your tests.

Step 3:

```
./manage.py test APP
```

Things to notice This fixes a lot of complaints that people had about previous versions of test maker. This allows you to test apps that are anywhere on your Python Path (and in your INSTALLED_APPS), which makes life a lot easier. Each view also has it's own test name, which is a slugified version of the request path and the time when you hit it (because I needed something unique :)) You also may notice that there is rudimentary support for template tags; this will be explained upon in my next post. However, for now know that it only works for template tags that don't set a context variable, or use the format `as <context_var>` to set one.

Improvements over 0.1

- Each page request is in its own test, for easier debugging
- It will append tests if your APP_testmaker.py file already exists.
- You can now test admin views
- POST support is improved
- The code is cleaner and more readable
- Git!

Options

-f –fixture If you pass the `-f` option to testmaker, it will create fixtures for you. They will be saved in `APP/fixtures/APP_fixtures.FORMAT`. The default format is XML because I was having problems with JSON.

–format Pass this in with a valid serialization format for Django. Options are currently json, yaml, or xml.

–addrport This allows you to pass in the normal address and post options for runserver.

Future improvements

Force app filtering I plan on having an option that allows you to restrict the views to the app that you passed in on the command line. This would inspect the URLConf for the app, and only output tests matching those URLs. This would allow you to fine tune your tests so that it is guaranteed to only test views in the app.

Better test naming scheme The current way of naming tests is a bit hackish, and could be improved. It works for now, and keeps names unique, so it's achieving that goal. Suggestions welcome for a better way to name things.

Improve template tag testmaker It is a total hack at current, but it works. Certainly a first, rough draft.

The value of conventions, aka testmaker for template tags.

A couple posts ago, I talked about how we should have conventions for the names that we use in Django Template Blocks. Today I will be talking about the value that is gained from this kind of structure.

Use Cases

Template Tags My use case for Template tags is what started me thinking about this. Some of you may know that I have created a testmaker application for Django. This allows you to automatically test your view code, based on a browser session. Once I got most of the kinks worked out in this code, I started thinking about what the next best thing to test would be. I came up with template tags...

This is where I ran into a problem. With no context associated with a template tag, there is no way to create a tool which tests template tags well. At first I simply started out trying to test the template tags by pulling them out of the template verbatim.

```
{% load blog %}{% get_latest_posts for blog.post as posts limit 10 %}
```

However, when you try and test this, it doesn't work. That is because all this code is doing is settings a context variable, and not outputting anything. You can create tests for trivial template tags that just output a string, but a lot of template tags set context variables. So without some kind of convention here, it is impossible to write a tool that will automatically write a test for you. That sucks!

Luckily in Django, the above test is representative of a kind of convention in django template tags. Most template tags use the syntax `as [context_var]` to set a variable in the context. So I went ahead and wrote some code that parses template tags for these kind of strings.

This code is valuable for some people, but is worthless if people use another syntax for defining context variables. This I think is a really good example of where syntax (or convention) allow you to do more than you previously could.

You can take a look at the source code here. It's still a bit rough, like most of my first releases it is more of a proof of concept.

Template Blocks So if we create a convention for Template blocks like I proposed in my previous post, this gives us some really neat possibilities. We can now create a base template that "knows" what will be included in each of it's sections. So in turn we create a way to provide skins or themes for Django Sites, that would be portable between Installations.

Of course, how far we take these conventions will limit how portable, powerful, and easy to replicate the designs will be. If we say that all items in a `menu` block have to be `<ul class=menu_item>`, then we can provide more functionality in our portable base template. This is a bit too specific though, because not all menus are lists. However, even with just a simple structure around your base template, you can create some really nice portable templates. You can create 1 and 2 column layouts, simply based on where the menu, content, header, and footer are for example.

I think it will be interesting seeing where we can embrace conventions where possible, for the betterment of all. I think that having Django Skins would be really neat :). Also, having tests automagically generated for template tags is a big win. For no other reason than because it does the boilerplate stuff for you.

Other places I think that there are more places where conventions could benefit us. I think I'm going to create a section in my projects on this site dedicated to conventions for Django. Hopefully serving as a reference for other people who are trying to use conventions in their Django apps.

Introduction to Python/Django testing: Basic Unit Tests

Last post we talked about how to set up and use doc tests inside of Django. Today, in the second post of the series, we'll be talking about how to use the other testing framework that comes with Python, unittest. unittest is a xUnit type of testing system (JUnit from the java world is another example) implemented in Python. It is a much more robust solution for testing than Doc tests, and allows for a lot more organization of code. We'll get into that in the next post in the series, comparing Unit and Doc tests.

So we're going to assume that you are picking up after the previous post in this series. If so, you should have a basic tests directory, with an `__init__.py` and a `doctst.py` file inside of it. Today we are going to write some very basic unit tests, and figure out how to wire those into your existing test suite.

Writing your first unit test

Making a unit test is a lot like making a python class. As per usual, the Django docs have lots of great information and examples. They will show you how to do some easy stuff with your Django models. This tutorial will mostly be about how to use unit tests inside Django, irregardless of the data at hard. So let's start with a very basic unit test.

```
import unittest

class TestBasic(unittest.TestCase):
    "Basic tests"

    def test_basic(self):
        a = 1
        self.assertEqual(1, a)

    def test_basic_2(self):
        a = 1
        assert a == 1
```

This is a very basic unit test. You will notice it is just normal a normal python class. You create a class that inherits from unittest.TestCase. This tells unittest that it is a test file. Then you simply go in and define some functions (Note: they need to start with test so that unittest will run them), in which you assert some conditions which are true. This allows you a lot more flexibility in the tests.

Now if you try to run these tests, you will again not get have them showing up in your output! You need to go into your `__init__.py` in your tests directory. It should now look like the following (assuming you followed part 1 of this series):

```
from unittst import *

import doctst

__test__ = {
    'Doctest': doctst
    }
```

Unit tests are a lot easier to import than doctests. You simply do a `from <filename> import <testnames>`. I named my unit test file `unittst.py`, and python will import that from the current directory. You are importing the test classes that you defined in your file. So I could have as easily put `from unittest import TestBasic` and it would work. Using the `import *` syntax allows us to add more tests to the file later and not have to edit it.

You can go ahead and run your tests, and see if they're being properly imported.

```
[eric@Odin:~/EH]$ ./manage.py test mine
Creating test database...
Creating table auth_permission
[Database stuff removed]
...
----------------------------------------------------------------------
Ran 3 tests in 0.004s

OK
```

Awesome!

A couple neat features

There are some neat things you can do with basic unit tests. Below I'll show an addition to the above file, which is another test class, with a bit more functionality.

```
class TestBasic2(unittest.TestCase):
    "Show setup and teardown"

    def setUp(self):
        self.a = 1

    def tearDown(self):
        del self.a

    def test_basic1(self):
        "Basic with setup"
        self.assertNotEqual(self.a, 2)

    def test_basic2(self):
        "Basic2 with setup"
        assert self.a != 2

    def test_fail(self):
        "This test should fail"
        assert self.a == 2
```

Here you see that you can define a docstring for the tests. These are used when you are running the tests, so you have a human readable name. You'll also notice that I've used some more assertions. The python docs have a full list of assertions that you can make. The `setUp` and `tearDown` methods are run before and after every test respectively. This allows you to set up a basic context or environment inside of each of your tests. This also insures that each of your tests do not edit the data that other tests depend on. This is a basic tenet of testing, that each test should stand alone, and not effect the others.

This also seems like a good time to explicitly say that all of your test classes and files should start with test! If not, they will not be run! If you have a test not running and everything else looks right, this is probably your problem. Also note that they cannot be named the same thing! These will overwrite one another with the last one being imported into the file running. It is generally a good practice to name your tests something that is certain to be unique. I generally tend to follow whatever naming convention I've used for my named url patterns.

When you go ahead and run your tests, you should see one that fails (the last one).

```
[eric@Odin:~/EH]$ ./manage.py test mine
Creating test database...
Creating table auth_permission
[Database stuff removed]
....F.
======================================================
FAIL: This test should fail
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/home/eric/Python/EH/mine/tests/unittst.py", line 35, in test_fail
    assert self.a == 2
AssertionError

----------------------------------------------------------------------
Ran 6 tests in 0.003s

FAILED (failures=1)
```

You can see the value of unit tests here. Each test is run seperately, so you get a nice human readable error message when it breaks. You can go ahead and make that test pass (`self.assertFalse(self.a == 2)`). You get an OK from your tests, and we can go on our merry way.

Now you can see for yourself that there are a lot of differences between Doc tests and Unit tests. They each serve their own purpose, and in the next post in this series I will talk about when you should use each. Unit tests require a little bit more up front effort; you can't just paste something out of your python shell and have it work. However, they give you a lot more flexibility.

New Design

I just pushed my new site design live. My last post got lots of hits and I was tired of the comments about how horrible the site looks :). Please let me know what you think. There are still a couple rough edges, but I think overall it is a lot better!

I would like to thank my brother for putting together the design. I really like it! I hope everyone's eyes have stopped bleeding, and you can now enjoy reading the site :). There are a couple rough edges that will be worked out over the next couple days. Enjoy!

Post a day in review

It's the end of the Post a day for a month. I did pretty well, but fell off about 3 weeks in because of work. First some stats.

Post stats

```
In [4]: Post.objects.published().filter(publish__year='2008', publish__month='11').count()
Out[4]: 23L

In [10]: for post in posts:
    cont = post.body.split()
    sum += len(cont)

In [13]: sum
Out[13]: 18517
```

```
In [14]: sum / 23
Out[14]: 805
```

This doesn't really mean much, because it is simplistic len(body.split()), but it shows that I have been writing a ton over the past month. I have also gotten a ton of stuff done. I have re-written 1 project, started another, and put out some ideas into the community.

The above stats basically mean I posted 24 times (counting this one), averaged 800 words a post (with code examples probably inflating that)

I only missed my first day about 3 weeks in, and that was because of work. I was working non-stop trying to launch our main websites at work on Django 1.0, and just didn't have enough time (energy really) to do anything else. For Thanksgiving I was also up visiting family in rural Ohio for 4 days, so I didn't have an internet connection there, which made posting hard.

Traffic stats

Figure 2.8: Analytics stats

The day after that big spike I launched my redesign, so there are probably around 1,000 hits missing from that. I got around 30,000 Page views and 17,000 Unique visitors.

Pros and cons of post-a-day

Pros I have certainly enjoyed doing this. It has been a neat personal challenge, and I think that the quality of my posts hasn't really suffered. If anything, it has stayed the same, and I have just been posting more frequently. I had a couple of "cop out" posts (like this one), but they never really seemed to be much less popular than some of the ones I put some time into.

Having to do a post a day makes you do a lot more! You accomplish so much because you need something to write about. I think that this is probably the biggest advantage to doing post a day. You're forced to do something interesting each and every day, which is something that more of us should be doing.

I also think that it's interesting what kind of content you end up writing. I have done a lot more tutorials and things like that. I have found them to be easy to write, and really valuable for people to use.

Cons You write blog posts instead of doing things! I spent a ton of time writing instead of coding. A lot of my posts took around an hour or more to write, so that's a lot of time spent. However, I think that this is a good thing for the open source community, since the amount of documentation versus code produced is really unbalanced.

I really wanted to review some major django reusable apps and write up howto's and screencasts for them, but this proved to be really time consuming. So doing a post a day really limits your drive to do longer blog posts (because you have to do one again tomorrow!).

After about 2 weeks I agree with other people, where they say it turned from being fun into a burden. Not a huge burden, but it was enough stress (along with all my other real life stuff) that it was annoying. I missed a bunch of days at the end for this very reason.

Reflection

I'm really glad that I did it. I broke down somewhere through, but I still feel that I accomplished my goal of posting a lot of good content, and getting shit done. I am so happy that the month is over, and that Ellington is now running on

Django 1.0. My life has gotten a lot less stressful. I will be taking a lot of time off in December, so this blog will be a bit more quiet :)

Going forward, I will try and post at least once a week, and hopefully some more screencasts and longer form content. I think that the kind of content that was produced during this shows what's hard and what's not. Simple tutorials are great for things you know, but doing the research to do a screencast (that doesn't suck) or other kind of content like that takes a lot of time!

Hope everyone enjoyed these posts, and I'll try and keep up.

Introduction to Python/Django tests: Fixtures

In the first two posts of this series, we talked about how to get the basic infrastructure for your tests up and running. You should have a file with doc tests and one with unit tests. They should be linked into your django project with an `__init__.py` file. If you were feeling adverterous you may have even added some real content to them. Today we're going to start going down the road of getting some data into your tests.

This post will cover fixtures. Fixtures are how you create a state for your database in your tests. This will be my first post in the comparison between Unit tests and Doc tests. I will focus on fixtures, but some other differences between the two may become relevant throughout the post, and I will address them in turn.

How to create a fixture

Fixtures can go into a couple places, but generally it is a good idea to put them in your applications `fixtures/` directory. This makes all of your test data self contained inside your app, so it can be run when it is distributed. The loaddata command discussed further down specifies where fixtures can be placed to be loaded, if you're curious.

Before we go about trying to figure out how to use fixtures, we need to know how to make them. Django's docs on this are pretty good. Basically if you have an app that has data in the database, you can use the `./manage.py dumpdata <app>` command to create a fixture from the current data in the database. It's handy to note that you can use the `--format` tag to specify the output format, and the `--indent` command to make the output prettier. My preferred command is

```
#This assumes you are at the project level, right above your app.
#and that APP/fixtures/ exists
./manage.py dumpdata APP --format=yaml --indent=4 > APP/fixtures/APP.yaml
```

This makes for a really nice, readable fixture, so if you need to edit it later you can. Go ahead and run this command in your project directory, substituting your app in the appropriate places. Open the fixture if you want and take a peak inside. It should be a nice readable version of your database, serialized into Yaml. **Note**: If you don't have PyYAML installed, it will say that your serialization format isn't valid, `sudo apt-get install python-yaml` gets you the package on Ubuntu. If not, you can remove the format option and it will default to JSON.

Testing your fixtures (how meta of us!)

Django also comes with a really neat tool to be able to test and update fixtures. The testserver command allows you to run the development server, passing a fixture to load before it launches. This allows you to run your code base against the fixture that you have, in a browser.

This seems really nice, but the killer feature of this command is that it keeps the database around after you kill the development server. This means that you can load up a fixture, edit it in the admin or frontend, and then kill the server; then run dumpdata against that database and get your updated fixture back out. Pretty neat! Note, your normal database name will be prefixed with `test_`, so it doesn't overwrite your normal DB. This is the one you want to get data out of. (You may have to define it in your `settings.py` file to get dumpdata to use it. This seems like a little bit of a hack, and maybe something could be done to make this easier.)

Fixtures in Doc tests

In what will become a recurring trend, doing fixtures in Doc tests is a hack. Doc tests are made to be a simple answer to a relatively simple problem, and fixtures aren't a huge deal for them. So a lot of the functionality that we get for free with Unit tests, has to be hacked into Doc tests. I will just show how to do the basic things, because implementing anything beyond that isn't very useful for any of us.

```
>>> from django.core.management import call_command
>>> call_command("loaddata", "'' + 'fixturefile.json' + '", verbosity=0)
```

In this snippet you are basically calling it the way it is called within Django. Normally when you are using loaddata, you would be calling it as `./manage.py loaddata FIXTURE.`. Note that the loaddata docs talk about how to use loaddata normally. There are a couple of downsides to this method; The test is very fragile, if the fixture breaks, all of your tests fail. Also, you can really only call one fixtures at a time because there is no setUp and tearDown that will make sure your environment is the same for every test. Doing things this way just makes writing tests a whole lot harder. It is indeed a hack, and one that shouldn't really be used unless you have a very good reason.

Generally in Doc tests, you would create your content as if you were on the command line. This shows how doc tests are generally limited in their scope. You go ahead and create the objects that you care about in the test explicitly, and then run your tests against them. A simple example:

```
>>> from mine.models import Site
>>> s = Site.objects.create(url='http://google.com', query='test', title='test', content='lots of stu
>>> s.query
'test'
>>> s.save()
>>> pk_site = s.pk
>>> Site.objects.get(pk=pk_site)
<Site: test>
>>> Site.objects.get(pk=pk_site).delete()
```

This tests creating, retrieving and deleting an object. Not a lot of functionality, but if anything inside of the model saving code breaks you will know about it.

Django's Testcase

The fixture story in Unit tests is much better, as you would expect. However, before we go into how Unit tests use fixtures, there is something that I need to explain. Because of the fact that Unit tests are classes, they can be subclassed just like any other Python class. This means that Django has provided it's own Testcase class that we can inherit from and get some nice extra Django functionality. The official docs has it really well documented.

You'll notice that Django's Testcase has a section for the Test Client and URLConf configuration. We can safely skip those for the moment because they are geared towards testing views. The relevant sections for us at the moment are the Fixture loading and Assertions. I recommend that you actually read the entire testing doc, it isn't that long, and is packed full of useful information. However, knowing about all of the Assertions that are available to you will make testing a little bit easier.

Fixtures in Unit Tests

The big thing that the Django Testcase does for you in regards to fixtures is that it maintains a consistent state for all of your tests. Before each test is run, the database is flushed: returning it to a pristine state (like after your first syncdb). Then your fixture gets loaded into the DB, then setUp() is called, then your test is run, then tearDown() is called. Keeping your tests insulated from each other is incredibly important when you are trying to make a good test suite. Having tests altering each others data, or having tests that depend on another test altering data are inherently fragile.

Now lets talk about how you're actually going to use these fixtures. We're going to go ahead and recreate the simple doc test above. It simply loads up a Site object into the database, checks for some data in it, then deletes it. The fixture handling will handle all of the loading and deleting for us, so all we need to worry about is testing our logic! This makes the test a lot easier to read, and makes its intention a lot clearer.

```
from django.test import TestCase
from mine.models import Site

class SiteTests(TestCase):
    #This is the fixture:
    #-   fields: {content: lots of stuff, query: test, title: test, url: 'http://google.com'}
    #model: mine.site
    #pk: 1
    fixtures = ['mine']

    def testFluffyAnimals(self):
        s = Site.objects.get(pk=1)
        self.assertEquals(s.query, 'test')
        s.query = 'who cares'
        s.save()
```

As you can see, this test is a lot simpler than the above one. It is also neat that we can edit the object and save it, and it doesn't matter. No other tests (if they existed) would be effected by this change. Notice that in my fixtures list, I only had mine and not mine.yaml or mine.json. It you don't add a file extension to your fixture, it will search for all fixtures with that name, of any extension. You can define an extension if you only want it to search for those types of fields.

I hope that you can see already how Unit Tests give you a lot more value when working with fixtures than doc tests. Having all of the loading, unloading, and flushing handled for you means that it will be done correctly. Once you get a moderately complicated testing scheme, trying to handle that all yourself inside of a doc test will lead to fragile and buggy code.

Making a Django Uber-Community

My workload at work is about to get a lot less critical and time consuming, so I was looking for a project to start on. I am really interested in the social aspects of the web, and below I will outline an idea that I think will be my next project.

At Djangocon there was talk by Adrian and Jacob in their Future of Django talk about having a common identity for a person across all Django sites. I think that this would be a really interesting thing to work on, and make all of our Django sites much more approachable. So in this post I'm going to lay out what I think this would look like, how it would likely be done, and then hopefully get some other people that are interested in it to help me brainstorm. There is a ticket open about it currently.

Information Aggregation

So first off, we need to figure out what this is going to look like. I imagine there being a central site that would organize all of our Django related activity. The best option at the moment is Django People, because it already has a lot of that data. I talked to Simon at Djangocon about his plans for Django People v2, and it sounds like this is the direction he was wanting to go. So Django People could serve as a personal aggregator for people. I view Django People as kind of the "Profile Page" of a person in the realm of Django. The main page could also function as a kind of "Life stream" of the project, so people could see what is going on *Right now*. I think a killer feature would be to have people be able to join into groups, based on projects, and have a life stream for that project. This would give people an idea of how active a project is, how many people use and develop it, and other interesting information that weighs into whether we decide to use a project. Simon in his scary brilliant way already has most of this information on the site. We just need to build a way to pull information that we care about in, and display it well.

Then we need some kind of large aggregator of content from all of the people in Django. I think that This week in Django is the place to do that. The Django Community Aggregator on the official Django site is lacking. I think this functionality could be pushed off to TWID. I think that this aggregation site would replace the aggregation of blog posts. It would hopefully support tagging, syndicated comments, language preferences, and other thing. I have talked to the TWID guys about doing this, and they said it sounded like a great idea.

I view these 2 sites as the foundation for what I hope to build. Then the question is, what other sites do we include in this 'django information stream'? I'm going to list the ones that I think have relevant information, and I would love to have suggestions for other sites that would provide a service.

Sites to include

- Django Plugables - This site is a directory of Django reusable apps. It is a great resource, and better than searching Google Code for 'django-'. Now that projects are getting hosted on Github, Bitbucket, Google Code, Pypi, and others, this aggregator of projects is useful. We could bring in data about what projects a person owns. The site currently supports handling commits, and it would be really neat to have able to pull in all of the commits to a project, as that is the big thing that this is all about, code.
- Django Snippets - This is a great site for Django. It allows people to submit snippets of useful code for others. We could bring in data about their posts and tags.
- Django Sites - This is a site that lists all of the sites that are made in django. We could pull in the Django Sites that people have made. We could also use this data to determine what host each site uses, and have a tally of hosts of Django web apps.
- Django Gigs - The place to look for Django work. Pull in people looking for work, and people with job postings.
- Other non-django sites - We could also pull in data from popular sites about Django. Think the Django popular section of Delicious, Reddit Django subsection, Twitter Search for Django, etc.

Ponies

Luckily in this day and age, most of the data we want is available in RSS feeds. For other things, there are APIs available. Luckily, most of the owners of the sites are members of the community, and I'm hoping they would be willing to have the information aggregated. I don't think that there is much of a technical challenge behind this, it is mostly just social and getting people to push their data in suitable formats.

I think it would be interesting to talk about microformats, and other ways of having the data on the public sites be available to be pulled. I think it would also be interesting to have OpenID enabled on all of the sites (Django people already has it). This would allow you to have a kind of floating profile. If a layer was built on top of this (white listed information providers?), you could edit your information on one site, and it would be sent to all the others in the inner circle.

This is mainly just a braindump, but I am willing to take the lead on this project and get people talking. The code would be released open source and I think we could even make it into a series of reusable apps (and maybe APIs) that could be extended to other communities. I think we could create a best of breed/proof of concept implementation of a community linked shared profile.

Using OAuth to share the information across the sites, and as authentication for that and more is also another thing that I think would be possible. I haven't implemented anything like this before, so I don't know if it's the correct technology.

There is a lot of possibility for this project, and I am really excited to get started on it. This is just mainly a feeler to see who would be interested in participating, and to get the ball rolling. I'd love to hear people's comments and criticisms.

Software that I use: Essentials 2008

Stealing an idea/meme from Mark Pilgrim I'm going to do a post of the essential software that I use in a day to day basis. Justin also did a similar post a couple days back. I think it is interesting to talk about what kind of tools you use, because it gives people an understanding into how you work, and also some pointers at stuff that maybe they too should be using.

I'm going to split my lists up rather arbitrarily, so here goes.

On the Server

- Slicehost - I love these guys. I've had my slice for a good 8 months, and they are hands down the best web host I've ever had. Respond to tweets or e-mails within an hour, great customer support, and rock solid hardware. I highly recommend them for any sysadmin minded developer. It's a great way to learn a little sysadmin skills, have root on a fast box with a fat pipe, and is generally just awesome. Best part: $20 a month for 256MB of RAM.
- Vim - The venerable text editor and perennial love of my life. It's great for making little quick fixes to files, and the key bindings are burned into my brain.
- Django - Big surprise there. This site is running on Django, and I work and post mostly about it. Yet it goes to show how good the software is that I still love it, even when it's my day job.
- Varnish - This is a really nice "state-of-the-art, high-performance HTTP accelerator". It sits in front of my Apache pages and caches them, making the site blazingly fast. At least that's what it claims. My sysadmin at work recommended it and it's really nice.
- Apache/mod_wsgi - The up and coming way to run your Django apps on Apache. It's a great way to host, and makes configuration and management a lot easier. Again, sysadmin recommended, but IANASA (I am not a sysadmin)
- Ubuntu - My favorite Linux operating system these days. I run it for all my Linux needs, desktop and server. It makes everything really easy, and I understand it well since I've been running it for a couple years.
- screen - Screen is the sysadmin and programmers best friend. If you aren't using it on your remote servers, you're doin it wrong. It gives you some really nice ways to attach and detach long running processes (think IRC clients, DB migrations, etc), basically gives you a terminal window manager, and lots lots more.
- ssh - Everyone favorite work horse. I use it for the usual things like system administration, but also some other neat things like SSH Tunneling, X forwarding, and as a poor mans VPN.
- bash - I don't use any of those fancy shells out there. bash with screen is more than anyone should need.
- Fabric - I am just starting to use this as a deployment tool for my Django applications. It makes life a lot easier and I'm really enjoying being able to automate simple repetitive tasks.
- git - I jumped on this bandwagon a week or 2 ago as well. It seems to be becoming the defacto DVCS tool for the Django community, and Github is a really neat tool.
- Feedburner - This is a neat app that gives you services associated with RSS feeds. They tell me how many subscribers i've lost with my pointless ramblings on a daily basis :). I also use it as an abstraction above a feed url, so if my feed url scheme changes on the backend, I just update Feedburner to point to the new one and nobody has to change their feeds.

On the Desktop

A note about my development environment. I try to only use tools that are available on Linux and OS X, because that gives me the mobility of being able to develop easily on both. Things like MacVim are neat because they give you

Vim but in a Mac friendly way. However, software like Textmate and Coda I don't want to get used to, because I think that Linux is the better choice for developing software (at least for Django/Python).

- Firefox - The awesomest web browser ever. I don't know what we did without Firebug. It's great for web development, and lots of other stuff. The extensions community is great, and they do some good work. Vimperator is also really neat, it gives you Vim key bindings in Firefox ;)
- Komodo Edit - The 5.0 version of this just got released, and I'm loving it. This is the open source and free version of the great Komodo IDE, from Activestate. I use it mostly because it's cross platform, and because it has some great Vim key bindings. I get the convinces of an editor, with good key bindings, and not being tied to any platform. I highly recommend it for anyone doing web development, and I'm even considering getting the IDE version which includes Source control management and debugging support.
- Xchat - The venerable IRC client. I've been using it on Linux since I began using it, and the Aqua port for OS X is a little lacking, but still has everything you need.
- Adium/Pidgin - The greatest piece of IM software to be invented. Called Pidgin on Linux, they provided the libpurple library, which is an abstraction of their IM connectivity layer. On OS X, Adium uses this and gives you a great UI on top. You can connect to lots of IM networks, all in one buddy list.
- Quicksilver/Gnome-do - These launcher-style programs are so integrated into my everyday habits, I don't know how we lived without them. Quicksilver is the original version (that I know of), and Gnome-do is a well done Gnome version of the same ideas. They allow you do basically run without an Applications menu and just use a key command based launcher to do things. If you're not using one, I highly recommend checking them out.
- iTunes/Amarok - Everyone needs a good audio player. iTunes and Amarok are the best of breed for OS X and Linux respectively. Amarok is a KDE project, but I use it because it is a damn fine media player.
- Terminal.app/gnome-terminal - I used to use iTerm on OS X, and there are still a couple of small things I like better on it (key bindings mostly). However, Terminal,app has gotten nice enough that I can use it, and it makes it easier to use other people's machines. Gnome-terminal is my choice on Linux, because it's a great one.
- vlc/mplayer - For your video playing needs, you can't beat these two open source projects. They both will play almost anything, and I tend to use vlc on OS X, and mplayer on Linux, because of their respective UIs. If these won't play a media file, then almost nothing will.
- sshfs/macfuse - I love sshfs. It uses the FUSE library to mount an SSH drive on your current filesystem. There are OS X and Linux versions of it, and it is insanely useful.
- Skitch - This is a really nice tool for sharing images and screenshots. It allows you to capture them super simply, annotate them, and upload them for others in around 5 clicks. Great for showing website brokenness and other general stuff.
- Twitterrific - A pretty good Twitter client for OS X. It isn't amazing, but it's good enough and it does what I need. I love me some twitter, and this keeps my addiction fed.
- iShowU - I use this to create those screencasts that you all love :) It's a great program for doing screencasts, it's pretty simple, and does one thing well. I'd also be curious if anyone has any free alternatives, or linux based screencasting apps that they can recommend.
- Transmission - A bit torrent client for the mac. It's simple and easy to use, I like it a lot. It was actually ported to Linux and included in Ubuntu I do believe.

Apps in the Cloud

- Google Reader - My current RSS reader. It's simple, does what I need, and generally stays out of my way.
- Gmail - My e-mail client of choice. It's just a great way to do e-mail, I can access it from everywhere, and the spam filtering is amazing. I've gotten like 1 ever, and my e-mail is right on the bottom of this site :)

- Google Analytics - What seems to be the big name in web analytics. Yahoo has a competing offering that they launched recently, which has kicked google into gear with new features. Competition is a great thing, and we'll see if it's worth switching over time, but for me it's still Analytics.
- Delicous - The great bookmark sharing service. I was using Ma.gnol.ia for a while, but most people at work are on delicious. I recommend culling a small network of like minded folk, and getting your network links in RSS. It is by far the best link feed I have, and beats any impersonal aggregator.
- Last.fm - I have over 32,000 tracks 'scrobbled' on their site. They know my taste of music scarily well, and it's just really neat data to have in public. Plus they have some good APIs and feeds for accessing it.
- Pandora - These guys have a brilliant music recommendation engine. I am constantly delighted and amazed by what music thay choose to play. You give it an artist and it plays similar music. I use this when my library is becoming stale, or I'm looking for good new music.
- Facebook - I like it less and less everyday, but the utility in it can't be denied. Keeping track of far away friends, old friends, and generally most of the people I know socially is key. I really hate how all the data is locked up and all that, but everyone uses it, so there isn't much you can do.
- Programming Reddit - I'll check out the front page something, but the programming section seems to have some quality content a majority of the time. The Python and Django sections also have a decent signal to noise ratio.
- Hacker News - I don't use reddit or HN that much, but Hacker news consistently has interesting information. I don't get the RSS, but they are really nice resources when you're bored, or looking for inspiration.
- Kayak - The best way that i've found to find flights online. Great tool for traveling.
- Craigslist - Everyone's favorite classifieds site. I bought a Wii for super cheap recently with lots of games. The free section is also a favorite.
- Freecycle - A personal favorite. It's like recycling, but people give stuff away for free. It's like craigslist's free, but generally less sketchy. This is how we got most of our furniture in college, it's generally in good shape. People are usually just happy to see it go away to good people. Highly recommended!

Dot files

Brian also posted this similar post yesterday. He included his dot files, so I figured I would share mine.

This is my .bash_profile:

```
export PYTHONPATH=$HOME/Python:$HOME/Python/Modules
export PATH=$HOME/bin:$PATH
export DJANGO_SETTINGS_MODULE="settings"
export HISTFILESIZE=10000000
set -o vi
export EDITOR=vim
export PS1="[\u@\h:\w]$ "

alias rs='/usr/bin/python ~/EH/manage.py runserver 67.207.139.9:8000 --settings settings_debug'
alias mp='/usr/bin/python ~/EH/manage.py'
alias sp='/usr/bin/python ~/EH/manage.py shell_plus'
alias bkup='/usr/bin/python ~/EH/manage.py dumpdata'
alias destroy-pyc='find . -name \*.pyc -delete'
alias dj='cd ~/Python/Modules/django-trunk'
alias a2='sudo /etc/init.d/apache2 restart'
alias tm='/usr/bin/python ~/EH/manage.py testmaker 67.207.139.9:8000 --settings settings_debug'
alias p='python'
alias x='exit'
# ^l clear screen
bind -m vi-insert "\C-l":clear-screen
```

```
# ^p check for partial match in history
bind -m vi-insert "\C-p":dynamic-complete-history
# ^n cycle through the list of partial matches
bind -m vi-insert "\C-n":menu-complete
```

My terminals look like this: `[eric@Odin:~/Python]$`. I use Vim keybindings in my terminal as well (I'm addicted, what can I say). I also use similar git commands to Brian, so I'll just let his stand as the original awesomeness.

I hope you all find these links useful and interesting. It gives you a little peek into how I spend my days. I'd love to hear what everyone else does. If you have any suggestions for things that I should probably be using, please feel free to let me know.

The problem with Django's Template Tags

There are a lot of things that I love about Django. Template tags are one of them. However, they do have a couple of warts that bother me. I know that there's a problem when I actively look for another way to accomplish something instead of writing a template tag. I view them as a kind of last resort; thinking 'can't we accomplish this with a Manager instead'? I think that we need to work on making useful template tags a little bit easier to make. Django goes a long way in doing this with the simple_tag and inclusion_tag types of tags. However, I think there needs to be something more.

When I look at how template tags are implemented, it seems that most of the Node classes are implemented the same way. This means that this implementation is probably sane. A lot of the differences I see are in the way that input is parsed. Template tags are basically just a string that is then passed to a function. The template tag function is responsible for parsing this string correctly and passing it off to a Node to be rendered. Because a template tag string can in theory contain anything, it makes it really hard to parse these strings in any kind of standard way. I think that this is generally a good thing, because this flexibility is nice when you're trying to do really complex things. However, I think that we can create some relatively simple tools that will help us wrangle 95% of common cases.

I don't know if the right answer is to include something in Django. So I'm going to look through the different approaches that I've seen taken to parsing template tags, and try and figure out the best way to do it. If there's another way to do these, then please let me know.

If len(bits) == MAGIC_NUMBER

I'd say that this is probably the most pervasive way of doing template tags. The django source uses it in some places, and a lot of people do it this way. It basically involves breaking the incoming string into an array, and checking to see what is in each index of the array. For example, if your tag syntax looked like `do_something for app.model [as contextvar]`, which has a default context variable if you don't pass one. The code to this approach would look like:

```
def parse_stuff(parser, token):
    bits = token.contents.split()
    if len(bits) == 5 and bits[1] == 'for' and bits[3] == 'as':
        return FooNode(bits[1], bits[3])
    if len(bits) == 3 and bits[1] == 'for':
        return FooNode(bits[1])
    else:
        raise template.TemplateSyntaxError, "%s: Fail" % bits[0]
```

As you can see, this quickly gets cumbersome with larger tags, and makes tags annoying static. It's a pain to have to remember the order of arguments and other things that don't need to matter.

Regular Expressions

Another way that I have seen these tackles is with the use of regluar expressions. They look a little something like this:

```
def parse_stuff(parser, token):
    import re
    default = re.compile('tagname for (\w+) as (\w+)')
    no_as = re.compile('tagname for (\w+)')
    if default.match(token):
        return FooNode(group(0), group(1))
    elif no_as.match(token):
        return FooNode(group(0))
    else:
        raise template.TemplateSyntaxError, "%s: Fail" % token[0]
```

Again, this has the same problems as the above approach. It's a little bit more annoying because of python's regular expression support lacking, but it gets the job done. I don't want to be writing a regex for every possible pattern though.

What I propose

I say that if we standardize the variables used for certain things in templates, then we can make some really simple parsing utils that will do our job for us. There are already a certain amount of best practice with template tags for what to use as command variables, and below I will list out the commonly used ones.

- as (Context Var): This is used to set a variable in the context of the page
- for (object): This is used to designate an object for an action to be taken on.
- limit (num): This is used to limit a result to a certain number of results.
- exclude (object): The same as for, but is used to exclude things of that type.

This is just a basic set of common variables that are 'special'. I think that it makes sense to start parsing template tag input strings for these strings. I wrote a little snippet to do this for you.

```
def parse_ttag(string):
    #This could be token.contents.split()
    bits = string.split()
    tags = {}
    possible_tags = ['as', 'for', 'limit', 'exclude']
    for index, bit in enumerate(bits):
        if bit.strip() in possible_tags:
            tags[bit.strip()] = bits[index+1]
    return tags
```

And when I run it on a simple example, you see the value in this approach:

```
>>> parse_ttag('test as word for for.bit limit 23')
{'as': 'word', 'limit': '23', 'for': 'for.bit'}
```

This approach is nice, because it doesn't matter what order the arguments are in. It simply returns a list of the keywords that you care about, and what their value was. I think that this makes it a lot easier to make a template tag, at the moment. This could also be extended to support multiple uses of each keyword, by using a list instead of a string as the value in the dictionary.

This is a simple little solution, and there is plenty of room for improvement. I think that there are some other ways to do much neater things with this, but it has worked. We could even go out and write some of the most common use

cases for template tags into a function that simply parses them and returns a node.

```
def parse_ttag(string):
    return context_for_object(token, FooNode)
```

Where this would call FooNode with the correct arguments. It would know that the correct syntax was [WhateverTag for Whatever as Context]. This would then just pass into a FooNode(Whatever, Context), where it could then do the actual action that was taking place. The template tag parsing doesn't need to care what the objects are, it is just parsing strings, and making sure that certain values are passed into the correct argument.

Here is a very basic implementation, that does nothing, but shows the ideas behind what I'm talking about.

```
class FooNode():
    def __init__(self, por, _as='default'):
        print "Making Node: for:%s, as:%s" % (por, _as)

def parse_ttag(string):
    bits = string.split()
    tags = {}
    possible_tags = ['as', 'for', 'limit', 'exclude']
    for index, bit in enumerate(bits):
        if bit.strip() in possible_tags:
            tags[bit.strip()] = bits[index+1]
    return tags

def some_random_tag(parser, token):
    return context_for_object(token, FooNode)

def context_for_object(token, Node):
    """This is a function that returns a Node.
    It takes a string from a template tag in the format
    TagName for [object] as [context variable]
    """
    tags = parse_ttag(token)
    if len(tags) == 2:
        return Node(tags['for'], tags['as'])
    elif len(tags) == 1:
        return Node(tags['for'])
    else:
        #raise template.TemplateSyntaxError, "%s: Fail" % bits[]
        print "ERROR"

>>> some_random_tag('fake','test as word for for.bit')
Making Node: for:for.bit, as:word
<__main__.FooNode instance at 0x23aaa8>
>>> some_random_tag('fake_parser', 'fail whale')
ERROR
None
```

Notice how easy and logical the implementation is using the parse_ttags function, I think that the tags['for'] abstraction is a really good one. It takes the template tag string and parses out what you really care about. Now if we just write these for the most common cases of Template tags, we could make our lives a lot easier. I also assume that this can probably be done with this template parser in Django, but I've never really seen it used, or used it myself. Hopefully this is already done for us, and just not well documented.

The times, they are a changin

A couple posts back, I was talking about software that I use all the time. I was going through and linking to all of the software. I would go to google, type in the project name, go to the first result, and copy that URL back into my post. I figured that there had to be a better way. Any software project worth it's name owns the top result in google.

The web is a dynamic place, and websites move, change, and disappear all the time. The popularity and importance of some things change over time as well. So I was thinking about how to go about linking to the most popular thing for a search. It also happened to be useful for my last post and linking to things. A similar approach could be used with Wikipedia. Creating something that just linked to somethings wikipedia page.

Show me the code.

I couldn't use Google, because I couldn't find a good web search API for them. Yahoo however has a really nice one that is out there to use. This, combined with James Bennett's awesomely useful template utils allowed me to whip this up in about 10 minutes. Here's the code below.

```
from template_utils.nodes import ContextUpdatingNode
from yahoo.search.web import WebSearch

class SearchNode(ContextUpdatingNode):
    def __init__(self, search):
        self.search = search

    def get_content(self, context):
        srch = WebSearch('EricSearch', query=self.search)
        res = srch.parse_results()
        return {'top_url': res.results[0].Url}

@register.tag
def first_yahoo_link(parser, token):
    return SearchNode(token.split_contents()[1])
```

Easy as pie, and awesome. This ties into my previous post about template tags being hard to write. If you just want to make a template tag that sets a context, it's as easy as making a node and returning a dictionary in the `get_content()` function. This isn't a super robust solution, but now i can do `{% first_yahoo_link "search terms" %}` and `{{ top_url }}` will contain the Url of it!

Also note how easy it is to use Yahoo's search api! That's awesome. If this was on a more highly trafficked site, you would want to cache the results (maybe daily, because they shouldn't change much). I may go ahead and do a tutorial on how to do caching with template tags and template-utils if people are interested in it.

The observant will note that this doesn't help me writing a blog post, because there's no way to call a template tag from within. That might be something for me to cook up later this month. The template tag is still neat however, for introducing yahoo's web api, and template-utils.

Starting a Django Conventions Project and Reference

During the last month I have proposed some conventions for Django, mostly in the realm of templates. In doing so I have looked around for other documented places where conventions are mentioned. I haven't found a really good reference for Django conventions. Brian's post was a good example of reusable app conventions, and the Pinax Project is a great reference implementation. However, I couldn't find any simple reference for regularly used conventions in the Django world.

I don't know if this will be useful for people, but I think this goes along the whole convention/pattern ideal. If we all use a common naming, syntax, and style in places where they can be arbitrary, then we gain a lot of value of being

able to understand whats going on in others code. So I have started a project that hopefully will act as a reference for Django Conventions.

Currently it is pretty sparse, but I think that having that document in any form is a great step. I'd love to hear some feedback, and it needs a lot of work, so feel free to email me or leave comments here with your additions and criticisms. If this idea becomes useful, I would be fully in support of including it in the Django Documentation or something along those lines as well, but I don't know how "official" this will really be. For the moment just consider it my hair brained idea of how things should be done :) Cheers.

Year in Review

Well it's been a crazy roller coaster year for me. So this post is going to be the typical recap of what's gone on with my life and my blog over the past year. I'm really happy with where I'm at both professionally and personally, and 2008 has been an interesting year for me.

Last semester and Graduation

It's kind of surreal to think that at the beginning of this year, I was still a senior at the University of Mary Washington, living in Virginia with some of my best friends in an awesome house. My final semester senior year was a lot of fun, I took very little in the way of course work. I was employed by CACI, which is a defense contractor at the Naval Surface Warfare Center Dahlgren. I was doing Java, Javascript, and a little Perl code for a Navy portal at the base. It was a really fun job socially, but the technology (other than Javascript libraries) was pretty dull.

At school I was finishing up my senior/honors project using Django. This turned out to have probably been the most important decision of my college career (hindsight being 20/20). I went to the National Conference on Undergraduate Research to present the ideas from my senior project. This was a really neat place, and I was exposed to a lot of interesting things other people were doing.

We had a really crazy Primary Election season, which included Bill Clinton talking at my school, which I saw. Obama also later spoke at UMW (His famous in the rain speech), which i would have died to see.

I applied for a couple different jobs all over the country after I graduated. I had phone interviews with Yelp, a Wiki startup, and a couple other places. I interviewed and was offered a job by Zope which randomly is based in the town I went to college in. I also applied at The World Company, the birthplace of Django in the middle of the country, Kansas. I got offered the Jobs at Zope and the World Company, so I had to choose which to pick. As I've talked about before, I chose Kansas, and it has all been a blur since. I graduated from UMW.

The epic journeys

I accepted the job right before graduation at the end of April, subsequently quitting my old job at CACI. I arranged to start in Lawrence on July 1, giving me all of May and June to enjoy summer. I had some decent savings and decided to move to Lawrence with no money, and to travel a bunch before I went.

Over those 2 months I took a bunch of different trips. I went to Boston for a week, going to Barcamp Boston, which was the first conference I'd ever been to. I met some amazing people and got really excited about the culture that surrounds the profession that I had chosen. I also just got to tour around MIT and Harvard, met some great people through friends, and just had a great experience.

I also went down to North Carolina, and to the Outer Banks. The Outer Banks are one of my favorite places on earth, and totally recommend them to anyone. They have some of the best waves on the East Coast, and a completely relaxed and beautiful beach atmosphere. I really hit the beach hard because I was moving to KANSAS!

I went to Maryland to a friends late graduation party, and to visit family that I don't see very often, even less often now that I live in Kansas. I also went all around Virginia, to Nelson County to visit my friend Josh's house. Charlottesville

to see some music and visit friends. Virginia Beach (Home) a little bit to go surfing and visit friends and family. Berryville (where I grew up) to visit old friends and the rest of my family.

So at the end of June, I move out of my house, and leave for Kansas. Great first half of the year.

Lawrence Chronicles

I was in love with Lawrence on the first day. It's a great town and I love it to death. I moved into a Co-op for July-August so that I could find a place to lease and have an instant social network. The people I lived with were amazing, and it turns out that one of my co-workers had lived there while she was in college! Cool.

The job at the World Company turned out to be amazing, working with lots of brilliant people. On my second week on the job, DjangoCon was announced. It's really neat feeling like you fell right in the middle of something amazing going on. I had a hunch from afar, but it turned out to be more true than I could imagine.

My birthday was on July 9, and right around my Birthday I was added to Django's Community Aggregator. This was the first time that my blog had ever gotten more than 20 hits a day (and all those 10+ were when I sent my resumes around). I started getting people reading the things I was writing, and actually appreciating what I was saying. It's really neat to have a way to talk to people, and have them be excited and listen to what you say.

At the end of july, I released my first open source project, testmaker. This was met with a great response from the community. This is where I really started to appreciate and understand the value of the open source community. I got great feedback, inspiration from comments, and a great dialogue around the project. It is tiny compared to some of the things that people do, but I was floored with the response.

Djangocon was in September and was more amazing than I could have anticipated. I gave an incredibly nervous lightning talk (having broken my demo 5 mins before I went on, and fixed it). I learned so much, got to meet some amazing people, and had lots of fun. I see now that I didn't do a writeup post, which is sad.

November was post-a-day month that was a lot of work but very rewarding. I did it with a lot of other people in the community and I think it was a great effort and it worked out really well.

December was really laid back. I started a couple more projects which are going to be getting some love in the new year. I went to Jamaica on vacation, and went back to Virginia to visit friends and family for 2 weeks.

Wow

A lot of things have happened this year, and I'm grateful for where I am and what I'm doing. It's been an amazing ride, and I can only guess as to how 2009 will be. I'm sure it will be another great year, and things are only looking up. I want to thank everyone for reading my blog, and I want to wish everyone a happy new year full of blessings and insights.

Happy new year!!

2.3.7 2007

Updating website

I'm starting to update my website, moving everything from .shtml over to cgi's because it's easier. Also building out my web-based lyrics script to include saying lyrics are bad or good. Also trying to figure out a good way to automatically get all of the lyrics from an artist whenever one is found. Then run this in the background and they will be in the database cached when requested.

Also thinking of displaying a directory of artists, or even an AJAX interface to typing in the artist names and it has suggestions. That is seriously cool and might be able to be accomplished, and done well. We'll see how things go, I might even get credit in my AJAX class for doing the lyrics thing. I only had time to do research getting all of an

artists songs today. In the spare time i have between 15 credits and 12 hrs a week at my internship. Time is scarce, and needs to be spent doing creative things. :)

Good Software is SO hard to find..

I just installed Songbird which is a really neat music player built on top of XUL (of firefox fame). It's cross platform (yey good Linux support) and is currently only a developer release. It's working great for me and I'm excited about the possibilities. One pet peeve is that it didn't have a systray icon for it, one feature that i've grown accustomed to. Browsing their forms someone pointed to Alltray, which allows you to launch a program "alltray program", and it will automatically create a systray icon for it. COOL! How have I not heard of this before?

iPhone

Apple just released the iPhone today. This looks like a paradigm shift in the world of mobile phones. It's amazing how much a company can innovate when it doesn't have it's own silly motives to protect. Most other companies have 'walled gardens' or their own internet that they are trying to make money off of, so they don't offer Wifi access. It runs OS X, how long until this thing gets VoIP compatibility? They are Partnering with Cingular, so they may nix that idea, but it has to be on everyone's minds. The iPhone looks damned impressive, and if you go to their site you can see the amazing prototype.

People

It's amazing the difference having one person in your life can make or break your entire existence. Usually this would apply to a significant other, but a best friend is just as valuable if not more. Went to JMU for less than 24 hours with a good friend from school; to meet my best friend since second grade. We all got along marvelously and it was one of the best nights i've had in a long while. My spirits are high, I feel motivated, and my faith in humanity has once again been restored.

Music

We had a little jam session at my house over the past weekend. My roomate Tessie recorded it and here is a link to the mp3: http://ericholscher.com/music/us.mp3 Hope you enjoy it, its lots of drums, a theramin, and a bass.

Network KVM

This neat little program lets you use your network as a KVM. You set up a 'server' computer where you use the mouse and keyboard, and then 'client' computers on the right of left of your server, and when you go off the screen of the server, it automagically goes to control the mouse and KB of the client machine. Really neat.

http://synergy2.sourceforge.net/

Goal

I hope to write atleast one post a day, saying what I learned from that day. Mostly like a journal, and not super interesting to most people. I feel like this will help me improve my writing and give me content to write about. (I hope my days aren't so boring that the old adage doesn't apply)

UMW Blog Ring

Another idea to write up:

Start a blog writing website with fellow ambitious and interesting UMW students (Jeff, Lewis, Sam, Joel to start?)

Note: WRITE blog instead of read blog

Note2: WRITE about blogs you READ!

Note3: umwblogs.org exists, why were we not informed?

Digg/Wordpress plugin ideas

A lot of websites have those annoying 'digg this' buttons, with 0 diggs on it. How silly that makes them look. I feel like an idiot reading a web page that nobody else cares about...

How about implementing a feature (wp plugin?) that checks the diggs for each of your posts, and only includes the digg button on it when the number of diggs reachs N (20?).

Also, what if digg/reddit/etc. created a protocol that notifies a website when a post of theres is promoted to the front page, and even maybe integrate it into wordpress (blogging engine x), where it would be alerted if the post was on the front page (like a trackback), and then automatically make the page static instead of dynamic, presumably not crashing the server...?

Firefox Extensions I Use

Basically just a knowledge dump of The firefox extensions I use and where to get them for future reference.

Download Statusbar: https://addons.mozilla.org/en-US/firefox/addon/26

FireGPG: http://firegpg.tuxfamily.org/ : allows you to use GPG in forefox, useful for Gmail and Ubuntu's Launchpad (need it for their e-mails)

Adblock Plus: https://addons.mozilla.org/en-US/firefox/addon/1865 : also need the updater: https://addons.mozilla.org/en-US/firefox/addon/1136

Firebug: http://www.getfirebug.com/ : must have extension for Web/JS/CSS development

Cool site: archive.org

The Internet Archive is one of the neatest sites on the internet. I like them for a variety of reasons. First and foremost is the live music archive, they currently have 44,134 live concerts posted on their website. Completely free to download/stream til your hearts content. Most of the bands I like these days are on there, and they have an extensive Grateful Dead collection.

They also archive lots of other things, including websites, video, and documents. The main page gives you a partial list of all of the things that they are archiving. Really neat stuff.

Writing Advice?

The story starts out unusually. One of my friends is trying to write a very important letter to a family member. He doesn't know how to write it. He has the outline, but is very worried about the implied psychological impact. They worry about the reader thinking too much; "was he trying to be so nice and just said all nice things", or on the inverse "Wow, how hateful, full of hate he must have been"... We brainstorm the answer to the question which will seem obvious.

My advice: Just write. How are you feeling? He was feeling hatred at the time; for having to write the letter and towards the situation in general. That's the advice I give....RANT, spell out everything bad about the situation, the aunt, and writing in general if you will. Then move on....

It's a Computer Science idiom...Take the top and the bottom, then figure out what actually works. Write that rant full of hate. Then turn it on it's head, write that letter full of apology, understand the other persons point of view, get on the same page. Then write your actual letter.

Once you get past the abstract idea of the top and bottom, into the physical representation, it is much easier to analyze. Once you know what you said in your utmost anger and your utter sympathy, you know that you can't say that in actuality.

Once you know what not to say, you can narrow down what to say. Writing an emotionally charged letter like that isn't about saying the right thing; it's about NOT saying the wrong thing. Once emotions are involved, you have to direct them. If you can keep them from going down the rabbit hole of love or hate, you are in an infinitely safer territory. Emotions have to be kept within bounds, not exasperated and poked until a response. In situations like these, no response is a good response.

My friend was having trouble writing the letter, that's where my advice came from. I told him that it would be easier to write a rant than to not write at all. He was flustered by being stuck at the same point, at an outline with no perfect paper written. I told him to avoid perfection, work outside in. Write the top and the bottom of the emotional plane, and then work your way inside. His default emotion was perturbed, so the end result will end up more angry than passive, but you can't achieve perfection in a single keystroke. You have to have a start somewhere. The easiest way to get over writers block is to write what you feel, then don't send it, and actually write what you really feel. This context is uber-important. You don't send an e-mail when you're angry, you just write the first draft.

Write to the top, write to the bottom, write to the middle. Figure out where your allegiance lies, and decide appropriately from there. It's amazing how much easier it is to edit a paper when your reference is a very strongly worded statement, instead of an ideal idea. Making yourself sound more tempered from absolute crazy is trivial; making yourself sound level-headed for 2000 words on your first try is quite the accomplishment.

Last semester in stone

Registered for classes today, pretty excited about my schedule. My last semester senior year isn't going to be a cake walk like it should be (because i'm lazy), but it's going to be much better than this one. Taking the second part of my Physics lab (required) and 2 PE classes, tennis and weight training, fulfilling all of my required classes for the school. I'm going to be continuing my senior Independent Study Project, for another 3 credits. I'll be taking a 300-level CS database class to round out my Compsci Education. I will then also either be taking discrete math or intro to film studies, pass/fail. If discrete doesn't look like it'll be too much work i'll take them, if not i'll do film studies and coast.

I'm excited about taking the database class. I feel that is one of the few wholes in my CompSci theory education, and it was the one class I was sorry I hadn't gotten to take. I don't understand DB's well at all, and this class should fix that. It will make me a better web developer, and understanding of DB logic is applicable to a lot of the field, in the form of data storage and data relationships.

Schoolwork

Implemented Background processes, the ps command, and the kill command in GeekOS today for my Operating System class (hard shit!). Really neat stuff though.

I am also hacking arounc w/ Django to implement my independent study. So busy...

Fall is coming (and good content)

It's almost daylight savings time, this weekend. That makes me sad, I hate it getting dark at 5pm. It will make work much more manageable though. My current schedule is working 2-8 Monday's and Wednesdays. I adopted this schedule to make sure that I actually got some hours over the school year. 12/hr work weeks aren't too bad, but it is pretty hard with my hardest ever semester of school.

Today I read more about entrepreneurship, an issue that vastly interests me more and more every day. I feel like I have the right personality to start a company, and the skillset of engineering to do it. The computer is my canvas, the internet my vocabulary, and programming my paint. The huge freedom and flexibility afforded by the computer and internet are awe inspiring, but also very intimidating. I can most literally do *anything*, so what does one do?

I guess this situation is better than the people who got liberal arts degrees, namely most of my friends. They are in the same situation as I am, except without the skills to create something new, and without the vast salary potential. I am grateful that I am blessed to enjoy doing something that most other people do not. The power of computers and the internet is so awesome.

To start a company I need to have people to start it with. The CS department at Mary Wash has been slow in giving me people who I feel like I could do that with. I get along with some of the people, but don't consider a large amount of them good friends though. A lot of people seem to not actually be interested in CS, or at least not for the right reasons. A lot of 'suffer through CS education to get paid a lot' type, like doctors or lawyers. I have a couple of prospects for co-founders, but I wish I had a bigger network of Computer Science folk.

Ideas need context

Having this project to work on gives me more ideas. The ideas have context. Context makes them more valuable than abstract ideas that they once were. My context allows others to relate my ideas to their context easier than abstract ideas. Nice abstract idea isn't it? :)

Getting Real

Just started reading a book by the guys over at 37signals. It looks amazing. Completely free online. I'll get back with a review once I'm done, and hopefully with a finished website as well :)

http://gettingreal.37signals.com/toc.php

Browser Tabs

My first four browser tabs have been the same for the past couple hours. The first is the root of all evils, and the other three are productive!

Tab1: Google Reader (1000+) Tab2: Beautiful Soup documentation: awesome HTML parsing library that i'm trying to learn to scrape events websites. Tab3: Dive Into Python: Time to learn Python as well as learning Django... Tab4: My website! w00t! Making small progress....

Updated the OpenID libraries to make them smell better. Added little parts to my blog, and around the site, little small updates.

Lego Lovers

I know you loved Lego's as a kid. I wasn't hugely into them, but this site makes me wish I was.

http://www.brickshelf.com/

Awesome!

Python Easy Install

EDIT: Hey everyone, I wrote an updated post that actually tells how to setup a django app (and any python app) using easy install. Check it out!

I found a neat python module that lets you install other python modules. I have set it up on here, and it makes it really easy to install all of the things that I need for my projects. Yey..

http://peak.telecommunity.com/DevCenter/EasyInstall#traditional-pythonpath-based-installation

Merry Christmas

Merry Xmas everyone!

Stanford U

Just found that there is lots of awesome college content on iTunes. Standford U has some awesome stuff, I recommend watching Steve Jobs 2005 Commencement speech. Inspiring.

Django

Wordpress you were good to me. I'm going to migrate all my posts over to a new Django blogging app I'm writing. Part of my website for the Event Calendar and learning Django Done soonish hopefully

Weee

First Post

Testing Django Awesomeness