POSSE

Release

Nathaniel Case

June 12, 2012

CONTENTS

Welcome to POSSE RIT!

IRC: #teachingopensource on irc.freenode.net

CHAPTER

ONE

SYLLABUS

1.1 What You'll Do

Blog updates – students are required to keep a blog to which they post updates about their investigations, progress, success, and pitfalls. This blog can be hosted anywhere, but should be added to the TOS planet

Blogging is good for you and good for the FLOSS community at large.

1.2 The spirit of the course

This should have things in it.

1.3 Schedule

- *Day 1: Introduction to Open Source*
- *Day 2: Collaboration*
- *Day 3: Deep Dive*
- *Day 4: Community*
- *Day 5: Open Source in the Classroom*

CHAPTER

TWO

DAY 1: INTRODUCTION TO OPEN SOURCE

2.1 Suggested Readings

- Teaching Open Source's Introductions to wikis and IRC
- *Getting set up on IRC*

2.2 Outline of Discussions

Take the survey

Note: Lib arts components today/tomorrow, transition to tech matters later in week.

2.2.1 Introductions

Greetings and Introductions, survey of participants' skill sets

2.2.2 What is Open Source?

History and Terminology in the Morning: What Open Source Is

What Open Source is about; intro to the Fedora project; our teaching model; learning plan for the week

2.2.3 IP and Open Source History

Then we jump into History of IP

Stallman started copyleft notion of software copyright. MIT media Lab.

2.2.4 Working with colleagues exclusively online

- IRC/wiki lab/blogs&planets

Distributed Collaboration Fedora Classroom. Learning IRC and collaborative text editing alongside Fedora contributors.

After lunch, we'll be going over communication tools (IRC, Mailing Lists, etc...) and go into distributed collaboration and open source process. We'll talk about other tools as well, wikilab, basic communication tools, blogs, planets, and others. These are basic elements of FOSS collaboration and communication.

POSSE Blog Posts

- Create a blog or use pre-existing blog
- Get URL for RSS feed
- Log into teachingopensource.org
- Go to teachingopensource.org/index.php/planet_feed_list
- Edit "FEEDS" section
  - add your RSS feed URL
  - add your name
  - click "save page"

2.2.5 Finding things out

information-hunting and question-asking strategies. Being productively lost.

Lastly we'll talk about how to find things out on the web, and digging for answers. Where do you start, how do you find things out, and the concept of being "productively lost."

CHAPTER

THREE

DAY 2: COLLABORATION

3.1 Suggested Readings

- Git Cheat Sheet
- Git Book
- Github
- Examples of documentation
  - http://sphinx.pocoo.org/
  - http://www.zotero.org/
  - http://readthedocs.org/docs/sqlalchemy-stage/en/latest/
- Commarch Assignment

3.2 Outline of Discussions

3.2.1 Community Architecture

Introduction to an assignment on Community Architecture: finding an existing FOSS project, profile the contributors, arranged in groups of 2-4, and presenting the profiles on Friday.

Reading group: FOSS project case studies' What are the different types and characteristics of project communities out there for your students to contribute to? Give out Commarch Assignment

3.2.2 Git and Github

After that, we'll jump into version control with Git and Github.

3.2.3 Git Basic Commands

Refer to cheatsheet above for more.

Initialize a git repo	git init
Show current status of repo	git status
Add file(s) to local repo	git add <file>
Commit added files to local repo	git commit
Show log of history	git log
Show log by patches	git log -p
Add files by patches	git add -p

3.2.4 Making Your First Commit

Your first local commit, part 1 - picking a project and getting the code

Your first local commit, part 2 - building a local instance

After lunch, we're going to do a practice run on making changes to a code base. We'll walk you through a practice example.

3.2.5 Documentation

Manpages, documentation teams, and their workflows: how to find and contribute to them

After we'll jump into documentation, the need for it, the form of it, and it's importance in the context of FOSS development.

Essential Documentation Elements of Small Open Source Projects

- Design Documents
- Git log
- Write Wiki's
- Blog

CHAPTER

FOUR

DAY 3: DEEP DIVE

The “Deep Dive”. We have a few specifics about Patches/Pastebin/Tickets and other aspects, then we jump in.

Patches, pastebin, tickets, bots - and how to love your infrastructure team (Remy)

DEEP DIVE with WebWorks!

CHAPTER

FIVE

DAY 4: COMMUNITY

We'll be talking about the social aspects of collaboration, and a little bit today. Broadly speaking, it has it's own culture, and many subcultures within each project. Depending on the nature of the project, there are differences.

5.1 Outline of Discussions

5.1.1 Handling Flames and Forks

What do disagreements look like in open source, and how can we help our students cope with a sometimes unpredictably blunt world?

Flames

Everyone has used mail lists yes?

Everyone in open source has to have a thick skin

Imagine you have an open source project that is your baby

how would you respond to the same questions over and over again, year after year?

Folks entering open source project asking questions that are readily answerable with publicly posted information are subject to flames.

Elitist mindset is common to many major projects

Not saying this is a good or bad thing, just something to be expected.

New contributors have to "Do his/her homework"

Perhaps you've heard the advice "DON'T FEED THE TROLLS"? i.e. ignore flames or Do not respond to trolls publicly.

People in the community usually know who the trolls are.

If you are moving in a direction and there are no objects in your way then you are not actually moving. Haters, therefore, are one type of indication of success (of sorts).

If you have operator privileges in IRC, you can ban or kick them from the channel.

If you want it to be or seem more equitable, you can set up the com channel so that people are only given voice through a formal request. This can pre-empt "Trollish" tactics.

IRC has a couple of ways of dealing with the problem child. On a mailing list it is a little harder to do. Make sure it is subscriber only for posting. This eliminates (most) spammers. Try to filter folks on the way in.

If a flame war starts, be honest, be yourself, if it is a problem of elitism, “everyone is learning” is usually the right stance to take.

Hackers will often tell folks “you’re doing it wrong” Don’t take it personally, just say “Thank you, this is an open environment, we are all learning.” This is a much better response than Blocking and kicking, which should only be used as a last resort.

These issues are not that common in a student or classroom environment. Even with a decently sized open source project. FOSS project selection bias tends to eliminate trollish folks.

Large programs or projects may have some Tired crusty greybeards :) https://plus.google.com/102150693225130002912/posts/UkoAaLDpF4i

Flames can be playful and fun. The Computer Science House at RIT (http://csh.rit.edu) has a separate mailing list just for flames so that flaming does not interfere with the normal conversation, and is segregated into an “appropriate” context.

Forks

back in the early days forking was a dirty word. I’m taking my ball and going home. This was a scary term in Open Source because it could take away core contributors. Still is an issue. But fork is no longer a dirty word. Look on Github there is a fork button. The way that you protect your project is by licensing. If you are going to take this code for your project and make patches on it, you have to send those patches back upstream. PErmissisive licenses like BSD. OSX runs Unix under the hood, right there in a Mac. BSD license. Remy likes it. Many folks in the FOSS community don’t like Apple because they don’t release much back upstream. NATE Apple does release things back upstream, by and large they are Copyleft licensed stuff. Cocoa and some of the graphics stuff? Carbon is. NExt step and GNU Step. All of the base libraries for OSX are available, but are not used much outside of OSX so their usefullness is questionable. Many folks in the OS community are pushing Apple to move patches back upstream. To avoid forks put the material under CopyLeft licencing. NATE: copyleft is not a guarantee that such licensed code will remain open. Oracle bought Sun and (oracle is hostile to OS projects that it consumes) MySQL, Java–forked to Open JDK, Jenkins, Hudson, So mysql is a long running open source project. There was an immediate fork of mysql called Maria. Very quickly the forked project picked up many of the mysql developers. Oracle bought Open Office, which was forked to Libre Office which everyone moved to (except lame folks like myself who haven’t had the time to switch). Again many of the developers have moved over to Libre. “Recursive public” Decisions in licensing software can have huge implications down the road. One can even have the license held by a separate entity, or you can have the rights held by each of the contributors. In the former, it is easy to make changes, in the latter each contributor has to agree to changes. Forks can often create confusion amongst user bases. Almost no one has heard of MariaDB; everyone still uses Mysql. How do you get users to understand the changes? There’s no mechanism to get the word out to end users. Which is why distros and bundles are so important–they reset the default version (or fork) of what folks will actually use. Firefox. Mozilla has very strict rules for distributing things that will be called “Firefox” Debian’s version of Firefox is forked to be “Iceweasel” because it is too different from Firefox for Mozilla. Does the public care and to what degree? There are plenty of people who are just consumers of software. Not that one is better than the other. We are all in this hybrid transitional state. Software is so essential to current life The endless September back in theearly days every September students would go to college and would get access to the internet for the first time and all of these students would come on line and would have to be taught the social rules and best practices. Then as more folks came online September became all the time. http://www.codinghorror.com/blog/2012/05/please-dont-learn-to-code.html Bloomberg wants to learn to code‽

NATE there are a lot of new projects coming lately that attempt to do what existing projects already do well. Folks are forking or recreating existing projects without need? Concern is that it could become code Babel. Not necessarily bad. Seeing lots of projects doing this which will not likely be there in six months. Students have to research the field of existing projects before they start new projects. Hostility can result, will result from folks who see projects rehashing old ground, precisely because it is old ground and effort is obviously wasted. Open Video Chat project was created as a video chat tool for ASL communication on the XO (OLPC) laptop. They hacked the project to double the frame rate (started at 5-6fps). There were many hardware limitations, but there were community limitations as well. Other groups in OLPC, there ended up being different release cycles for each of the interested communities. We had to learn the lesson of doing our homework over and over again, even setting up a mailing list, which duplicated existing lists. Setting up a new IRC channel was a source of flack. Eventually we had enough development that we became accepted, hackathons. The OLPC development team was able to see that the project was making progress. If you think about your project as an element in an ecosystem then the flames do not seem so intense. Real time video chat is still one of the first things that folks ask for.

5.1.2 Commarch Assignment

How do FOSS projects are presented/deisseminated, and how to attract contributors.

5.1.3 Picking Pertinent Problems

Articulating your work in a way the community cares about

You can go to the community and ask “What needs to be done?” Remy is talking about Open States. –scrapes various state pub

–“What can I do to help?” the question you want to hear from the public Sunlight foundation & James Turk. There is a standard way of finding the source data, but the more people you have in local communities they can help developers interpret locally based data.

Developers may need specific technical and non-technical data. Think about your project on a milestone basis. Inverted pyramid thinking. Start with a general description that you may tell folks about what you are doing. level 2, what are the main elements of the project? What are the specific needs to each element of the stack? How can folks without coding abliity contribute to the project? Segmenting out your stack so that it is obvious what the various pieces are(to a third party). This facilitates contributors jumping into/onto the project. Level 3 tasks, specific

http://openstates.org/

Segmenting the project. You want to break the project into tasks and then tickets. Very small, granular level bugs. Once you have your tickets in something like Track. Github itself has a project management elements now, “issue tracker” in Github can be used to track tickets. Bug trackers Bugzilla, Track,

Once you’ve got the project segmented then you have to present it to the public. openhatch.org

Find a trivial bug and fix it.

Projects need to post easy bugs for new contributors to fix–as a means of setting a low bar to entry. Fedora keeps people engaged by putting their name on it. Giving people a title or a role. For students telling them that they are “Core Developers” makes them feel good about it and keeps them around. GIVES THEM OWNERSHIP. “You are helping to keep Open Source alive.” Don’t just engage students when something goes wrong. Use the carrot rather than the stick. Redhat does pay folks but Fedora is a community managed project. Hirees come from the developer community. Folks who have proven themselves. 15 people on the paid Fedora engineering team. Those developers depend on thousands of folks who are not paid. Most folks in Open Source are “Scratching their own itch.” http://openhatch.org/search/?q=&toughness=bitesize http://openstates.org/api/ https://github.com/bksteele57/Commarch-Android https://docs.google.com/document/d/1Dp0s_sh2Ba-UNVf7vRLCLO10MXmP1rvhEBWiXj8FWbE/edit?pli=1 https://docs.google.com/document/d/1Dp0s_sh2Ba-UNVf7vRLCLO10MXmP1rvhEBWiXj8FWbE/edit?pli=1 http://teachingopensource.org/index.php/Main_Page http://teachingopensource.org/index.php/Main_Page https://workflowy.com/shared/3ea5abdc-2513-aafc-ccc0-20bc7cf22cfb/#

5.1.4 Bus-/Raptor-proofing

Leveraging project teams to future-proof your work

Bus/Raptor Tests, speak to the notion of sustainability in development/collaboration. We've left a block open to particpants to post topics and issues related to FOSS. We'll poll people for specific topics.

if your entire dev team was on a bus and went off a cliff, what are the chances that your project would survive. Raptor proofing is about if this happened to your chief dev, ... All of this comes under the heading of future-proofing. Will your project be able to survive. Making sure that you are distributing your infrastructure and your developers. Where are the dangerous places in your project. how to protect you against disaster. First tool gitbyabus https://github.com/tomheon/git_by_a_bus/blob/master/README.txt very simple and easy to use. Can be really useful to answer some of the Commarch questions. https://github.com/Frencil/MultiGource/blob/master/log_generator.php http://www.youtube.com/watch?v=YZ6ILsOIBgA http://en.gravatar.com/ http://code.google.com/p/gource/ http://zmoazeni.github.com/gitspective/ https://github.com/tomheon/git_by_a_bus/blob/master/README.txt http://narcissus.rc.rit.edu/map#2.10/35.80/-344.20 http://threebean.org/ http://readthedocs.org/docs/ritfloss/en/latest/lectures.html?highli

5.1.5 Open Block

Participants can work on deep dive, Commarch Assignment, or direct discussion on an unplanned OS topic.

5.2 Home Stretch Dinner

Thursday Night will be a celebration/graduation dinner.

CHAPTER

SIX

DAY 5: OPEN SOURCE IN THE CLASSROOM

In the morning we will talk about your particpation in FOSS, and incorporating FOSS in the classroom.

6.1 Outline of Discussions

6.1.1 How Would You Like to Participate?

Participant Questions about General Open Source Topics

6.1.2 Commarch Project Presentations

There will be presentations from Comm Arch assignment.

6.1.3 Curriculum Workshopping

Presenting FOSS collaboration tools in the classroom, and whoever else is using them in the classroom can present.

This year, created an afterschool program through a ChaseManhattan grant called "East High College Readiness program" designed an ARG (augmented reality game) with post-apocalyptic island in the south atlantic with robots. Students had to use math problems to advance. The idea was to have students blog, and then add their blogs to an aggregated planet. That program was a pilot, and will continue this fall and hopefully expand. All materials will be made available and publicized after POSSE is done.

Second stage of HFOSS course tried to used: Openshift https://openshift.redhat.com/app/ Did not work well, issues with environment. Much time productively lost. first course fairly project regimented second course more seminarish A third course has been put together by Cody VanDeMark Software development on linux. Open source software development. Course is on TOS http://teachingopensource.org/index.php/Teaching_Materials_Catalogue Mark to Steve: Do you have a vision for where you would like this to go? I have a secret plan for the future only until I have put you all under virtual NDA. Heheh. There are very few centers like the FOSSBOX. STEVE: I didn't know step one about open source until the OLPC did their "Give one, get one." Service learning/work is very important to me. Anyone who is lucky enough to get to college owes something to the larger world. Built many not for profit websites in the 90's and early 2000's. IF organizations wanted us to work ont heir stuff they had to send someone to the class. Started OLPC user group so I wd have folks brains to pick. Nate was in first class. Students at the end asked "where do we go from here?" We have coops. Sugar labs is a 501c3. RIT allows students to work at 501c3's. Students jumped in and recruited other students themselves. Organic growth of students working on projects, tkinng classes, not for profit coops, and paid coops. Little enging going. Benefits for students. papers for me. Undergrad fellowships to work on projects. 80 % project student originated. Occasionally some come from me. Remy came on board by stages from Red Hat. It works; got a couple of papers out of it. What it lets us do we have a got a group of students. RPI is doing this. Got a million bucks to start a lab froma n alum who made his money in OS. Oregon State University, Seneca College (Chris Tyler)–Mozilla funding. Canandian version of NSF just gave Chris a 5 year grant open hardware, porting red hat stuff to raspberry pi. Antonio Mondragon. Andrea Hickerson rise above the crowd, for Imagine RIT. She is trying to get journalism students to write about Open Source. South by Southwest women & sexism and obstructionism in the open source community.

CHAPTER

SEVEN

HELPFUL LINKS – A LIST OF EXTERNAL RESOURCES

7.1 POSSE

- Community-editable notes
- Full schedule

7.2 git

- git cheat sheet

7.3 vim

- vim cheat sheet

7.4 IRC

- IRC commands

7.5 Open Source Projects

- MakerBot
- WeBWorK schedule

7.6 Open Source Philosophy

- The Cathedral and the Bazaar
- How To Ask Questions The Smart Way

CHAPTER

EIGHT

GETTING SET UP ON IRC

IRC: *http://www.irchelp.org/irchelp/networks/*

What is IRC? How does it work? Use a client, or a web client:

- ex *http://webchat.freenode.net*

8.1 Useful IRC Commands

- /connect -ssl irc.freenode.net
- /join #name-of-channel
- /help - show commands
- /names - Who is in channel
- /query - Private Message a Nick
- /wc - close the window
- /window move # - move a chat window to a particular number
- /whois - show information about a particular Nick

8.2 Register your nickanme on freenode

- Overview of registering a nickname on freenode.net

It is useful to register a nickname with the IRC server you are using, first so that you can always have your nickname when you log on (other users will be kicked if they try to use your nickname) and so that others can be sure that they are talking to the same person each time.

The following directions are for Freenode only, though other IRC servers will have similar instructions.

Once you are connected to freenode with the nickanme that you would like to register, send the following text in any channel connected to freenode:

```
$ /msg NickServ register <password> <email>
```

This sends a private message to the NickServ bot which stores your nickname, password, and an email you can use to identify yourself with the freenode staff should you need to recover the password.

Additionally, you should receive an email verifying your address to finish the registration process. Once that process is complete you will have to use the following text to identify yourself to freenode each time you log in:

```
$ /msg NickServ identify <password>
```

Sometimes, due to a lost connection or another computer running IRC seperately, you will not be able to get your main IRC nickname. In this case, most IRC clients will try appending an underscore (_) to your desired nickname. You can add these names to your registered nick without registering a new nickname. To do this, change your nickname to the next nickname, identify with NickServ again, and then group your current nick with your registered nick:

```
$ /nick <new nickname>
$ /msg NickServ identify <registered nickname> <password>
$ /msg NickServ group
```

Note that this time you specify the nicname you registered with NickServ, as it is now different than your current nickname.

8.3 About IRC Cloaks

http://meta.wikimedia.org/wiki/IRC/Cloaks

http://fedoraproject.org/wiki/FreenodeCloaks

CHAPTER

NINE

LAB: MAKING YOUR FIRST COMMIT

This repository has been used for a number of classes at Rochester Institute of Technology, and is designed to be an easily extensible primer on using git.

This lab assumes you already have a working git environment, as well as a Github account.

9.1 Getting the Source

The source for this documentation is currently found on Github. Every time a change is *pushed* to this repository, those changes are automatically propagated here. But we don't have to worry about that just yet. For now, we want our own copy of the documentation so we can make changes to it.

From the Github page of the project, look for a button labeled *fork* in the upper right of the screen. If you are logged in, this will create a fork of the project in your own account, where you can make and share changes without worrying about breaking the upstream's code.

Note: A *fork* refers to a repository which has been been copied from another repository and has possibly diverged from the original project. The two branches can be visualized to split, or come to a fork at this point.

Now that you have a fork you can commit changes to, we want to check out a local copy, so that we can actually get to making those changes. To do this on a UNIX clone (Mac OSX, Linux, etc.), open a terminal and run:

```
$ git clone git@github.com:<username>/posse.git
```

Alternatively, you could use the Github GUI on Windows or OSX to get a copy of the repository.

Now that you have a copy of the source for this documentation, we are going to edit one of the files inside. Using your favorite text editor, open up the file `data/students.yaml`

The `data/students.yaml` file is a structured data file that keeps track of all the students in the class and metadata about them. Using this file and the bindings in `lib/posse/model/students.py` we can build scripts that count how many lines of code each student modifies each week, or how many words/blogpost, or whatever we like.

What we are going to do is add our information to the file. There should already be a few entries in the file, so just copy one of those and add your details. You don't want to remove any of the ones currently there, but rather add yours.

Once you have done this, save the files, commit your changes, and push, either with a *git commit -a; git push* or through the Github GUI program.

Congratulations! You've made a change to an open source project, and shared it with the world. Now we need to notify the project we forked from that we have changes we'd like them to incorporate. Back on the Github page of your fork of the project, there should be a *Pull Request* button just to the left of where the fork button was that you used to make this fork of the repoitory. Pressing this button sends you to a screen giving you a chance to summarize the changes you've made and essentially justify why you feel this should be included. Submitting this opens a pull request to the upstream project, allowing others to discuss the proposed changes. As this is a trivial change, it is unlikely that much discussion will take place in this case.

9.2 Setting up your environment

Before you can do anything with this (build the documentation or run any of the scripts) you'll need to setup and activate a python virtualenv. Run the following at the command prompt...

9.2.1 On Linux/Mac OS X

If you don't have virtualenv installed yet, try:

```
$ sudo easy_install virtualenv virtualenvwrapper
```

If you're using a distro like Fedora or Ubuntu, you should try this instead:

```
$ sudo yum install python-virtualenv
```

Once you have virtualenv installed, you should be able to run:

```
$ virtualenv --no-site-packages -p python2 sphinxenv
$ source sphinxenv/bin/activate
$ git clone git@github.com:YOUR_USERNAME/posse.git
$ cd posse
$ python setup.py develop
```

9.2.2 On Windows

At the windows command prompt:

```
$ virtualenv --no-site-packages -p python2 sphinxenv
$ sphinxenv/Scripts/activate.bat
```

In msysGit or git-bash:

```
$ git clone git@github.com:YOUR_USERNAME/posse.git
```

Back in the windows command prompt:

```
$ cd posse
$ python setup.py develop
```

9.3 Building the "Documentation"

The "documentation" for the course (the syllabus, all the homework assignments, notes on the lectures) are all kept in the `doc/` directory of this repository. The files all end with the extension `.rst` which is the file extension for the reStructuredText markup language. They are all furthermore tied together the the *sphinx* framework for building integrated docs.

You might notice that the syllabus, et. al. is hosted on http://readthedocs.org/. The upstream github repository has a hook installed that automatically triggers a `git pull` at http://readthedocs.org from http://github.com. Thus, every time we change the docs here, they are automatically re-built into HTML for us and posted online. Awesome!

This however means that we should be careful before we push anything to github, or it will 'go live'. To be careful, you should rebuild the documentation locally (on your machine) to check that whatever modifications you made to the `.rst` files actually renders into the HTML that you want.

In order to do that, first make sure you have your virtualenv activated.

Being certain of that, in the root directory, simply run:

```
$ sphinx-build -b html doc html-output
```

The html documentation will be generated in `html-output/`. Check `html-output/html/index.html` to see if it exists.

Note: If your machine complains that 'sphinx-build' is a command that could not be found, try running "$ python setup.py develop" in the root of the posse repository first. That `setup.py` file contains information about all *other* open source projects that are *required* for this project, and will automatically install them from http://pypi.python.org/

9.4 Validating the `data/students.yaml` file

In order to ensure that you don't introduce any unparseable errors into the file, there is a script in `lib/posse/model/validate.py` that reads in the file and checks each entry. You should run it after every time you edit `data/students.yaml`.

In order to run the `validate.py` script, make sure you have your virtualenv activated.

In the root of the cloned source directory, run:

```
$ python lib/posse/model/validate.py
```

The data format (YAML) can be a little prickly though. It is *whitespace-sensitive*, meaning that how many spaces you put before an entry on each line has an impact on how the data is interpreted. It also means that tabs and spaces are distinctly different in their meaning. It also means that editing such a file is easy to mess up.