Yet Another Git Introduction

Alexey Veretennikov
alexey@veroveli.com

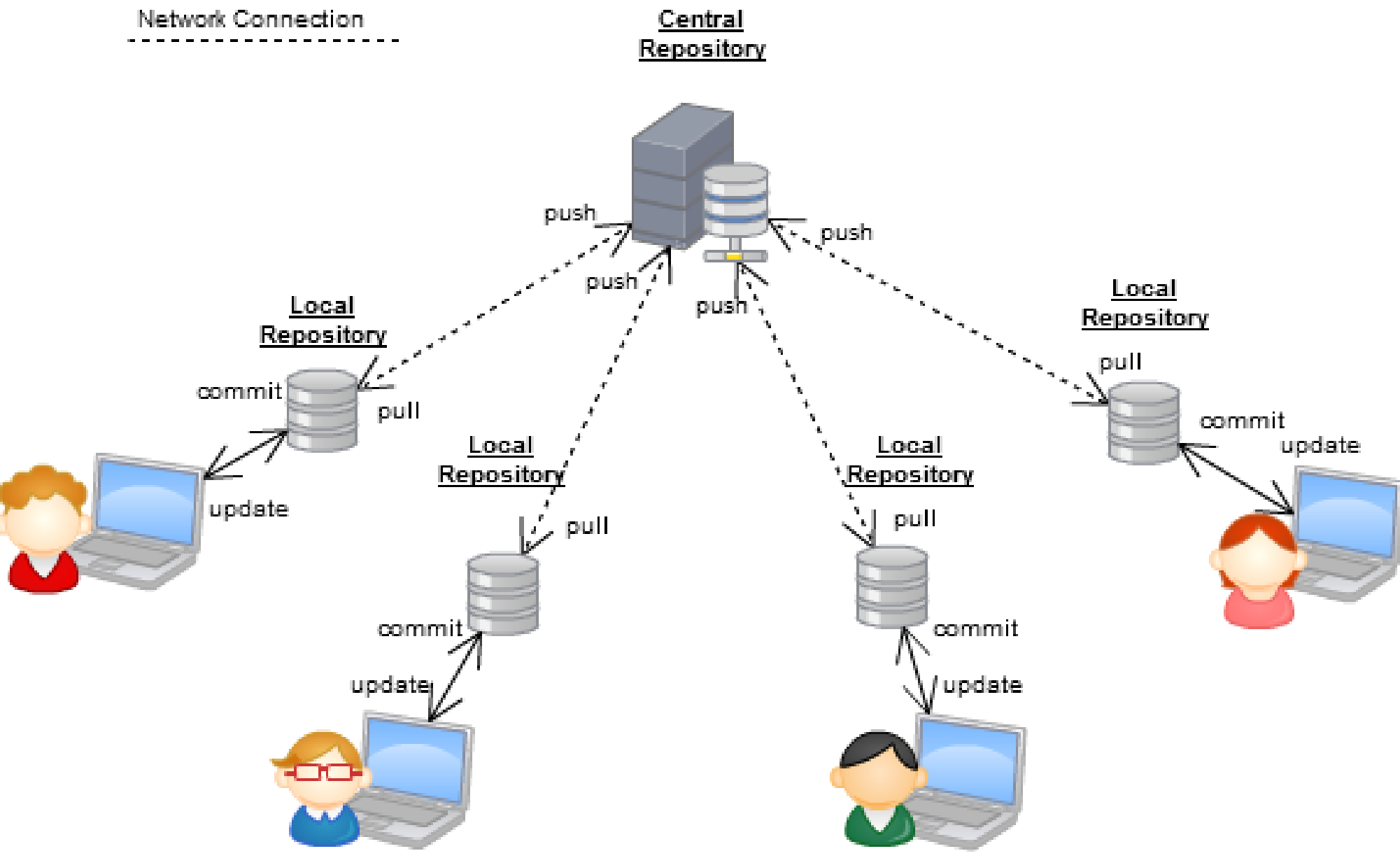
Veroveli AB

February 16, 2016

What is Git?

- Distributed Version Control System (DVCS)
- Started in 2005 by Linus Torvalds to replace BitKeeper
- Design goals:
  - Distributed workflow
  - Fast (patch,branch,merge)
  - Data Integrity
- Development started 3 of April:
  - Announced 6 of April
  - Self-hosted 7 of April
  - 16 June managed Linux kernel 2.6.12 release

Distributed Version Control System



Performance

- Create branch doesn't cost anything (branches are local)
- Apply patch/merge *extremely fast*
- View history etc *extremely fast* as well
- Order of magnitude faster than other DVCS(`https://web.archive.org/web/20100529094107/http://weblogs.mozillazine.org/jst/archives/2006/11/vcs_performance.html`)
- The only costly operation - send over network (but we are distributed, hey?)

Data integrity

- Every git object(commit,tree,blob) has SHA1 code calculated
- Commits and tags could be signed using GPG
- Very hard to do something unrecoverable (but possible) because
- Git generally only adds data

What else?

- It is a standard de-facto. More and more companies switch to it
- It is scalable. From 1 person project to projects of Linux Kernel scale
- Great tools support (IDEs, standalone GUI, code review, extensions)
- Open Source/Free Software already there
- Allows to work with others VCS like SVN (git-svn)
- Everything you can ever imagine in a VCS is there
- (Did I mention it is fast?)

Disadvantages

- Obscure terminology used only in Git (legacy)
- Command-line knowledge (not really)
- Complex commands for simple operations(could use aliases)
- Complicated tool itself (but hey, source code management *is* complex!)

Start to work with local code

First go to the directory with your source codes. Then:

`git init`	Initialize the empty repository in current directory
`git add myfile.c`	Add **myfile.c** to the *staging area*
`git commit`	Commit files from the *staging area* to repository

2-stage commit

1. One marks the changes one want to commit with `git add` command
2. One *commits* the changes to the local repository
3. (optional) If necessary, one *pushes* the changes to remote repository
4. (optional) But if the branch is out-of-date, one pulls changes from remote and repeats **step 3**

Git commit vs other VCS commit

Git commit is a *snapshot* of the data in directory, not *differences*.

- On commit git takes an overview of files and store reference
- If file is not changed, it is not stored again(link to previous file)
- Git thinks about *data*, not files
- For every commit SHA1 hash is calculated - hence the *commit-id* which looks like this: **345d29426c478379fff538fa9898965f78895690**
- commit-id is a 160 bits SHA1 hash
- Every commit (except first) has **parent** commit[s]

What is git repository after all

Git repository is just a directory .git in the root of your project:

HEAD	Branch you currently in
config	Contains project-specific configuration
description	Used by GitWeb, Git web frontend
index	Staging are, changes to be commited
hooks/	Hooks (like post-commit etc.)
info/	Only *exclude* file for ignored files
objects/	All contents of the database
refs/	Pointers to commit objects(branches)

How to access remote repository?

Create locally, copy anywhere. Access using:

- HTTP[S]
- FTP
- rsync
- Git protocol over SSH
- Git protocol over plain sockets
- mail with patches (popular in open-source/free software community)
- File system of course!

Typical workflow

Here we assume what the local repository is a clone of remote one

git status	Verify the status of local repository. Oh! Changes! I started to do the job with this bug #151!
git checkout -b bug_151	Create a new branch and move all not commited changes there
git commit -a... git commit -a	Do the job and commit as often as you want!
git checkout master	Job is done, lets switch back to **master** branch
git merge bug_151	Merge changes from **bug_151** to **master**
git pull	Take stuff from the remote
git push origin master	Push all stuff from master branch to remote

Recommendations

- Commit often. Commit any meaningful changes. Don't keep your stuff not commited
- Create as many branches as you need, typically one branch per task or proof-of-concept. It is cheap!
- If you are afraid of *polluting* the history with small commits, don't be: you can either *squash* all your commits while merging your branch to master(hey you are using branches, aren't you?) with `git merge --squash` , or do the interactive rebase with `git rebase -i` and squash several commits to one

If something goes wrong

Don't stress it up! All your changes are local! Even if they were pushed to the remote, git is distributed, remember? Someone still have the good repository :)

`git checkout – filename.java`	Revert modified file to the previous state
`git clean -f -d`	Remove all not tracked files and directories
`git reset –hard commit-id`	Destroy all commits after *commit-id*
`git clone ...`	You can always remove local repository and clone again :)

Useful things 1. Aliases

I hate to type long commands!

Git allows to define aliases for long commands, i.e. *git co = git checkout*. The **~/.gitconfig** could contain a following section with aliases, for example:

```
[alias]
ci = commit
st = status
co = checkout
subupd = submodule update --recursive --init
last = diff HEAD^ HEAD
br = branch
```

Useful things 2. Ignored files

I don't want to see *.obj, *.lib etc files!

Place a **.gitignore** file in the root of your repository and add all file names/wildcards you want to ignore. There is a great project with templates of this kind for possible projects:
`https://github.com/github/gitignore`

Useful things 3. Branches

It's annoying to type long branch names. And what branch I'm at right now?

- Use `TAB` key in command line to auto-complete branch name
- To find what branch you are in, use `git branch`
- `git branch -a` shows all local and remote branches

Useful things 4. Editor

When I do *git commit*, it opens something what only beeps so I can't type a commit message!

Git tries to run some editor so you can enter a meaningful commit message. It looks for:

- **GIT_EDITOR** environment variable
- *core.editor* Git configuration value
- **VISUAL** environment variable
- **EDITOR** environment variable
- Gives up and tries **vi** if nothing else works

Useful things 5. Stash

I don't want to commit my changes but I have to do some other job and commit it. What to do?

- Use the `git stash` command. It will save your local modifications and reverts the repository to the clean state. Then you can do your changes.
- After you have done and commited your other changes, you want to restore your uncommited ones. Do the `git stash pop` to restore them.
- However if you've decided what they not worth it, do `git stash drop` to cancel them permanently.

Useful things 6. Difftool

Any way to compare graphically?

Use `git difftool` for this. For example for Beyond Compare:

```
git config --global diff.tool bc3
git config --global difftool.bc3.path
  "C:/Program Files/Beyond Compare 4/BComp.exe"
```

And now you can run like

```
git difftool commit-id1..commit-id1 [-- myfile]
```

Useful things 7. Revert

Need a specific version of a file?

Simply checkout:

```
git checkout commit-id file-name
```

Useful things 8. Revert whole repo

But how about the a specific version of a whole repository?

One also checkout:

`git checkout commit-id`	1) Checkout to the version
`git checkout -b branch-name`	2) Create a branch ouf of it
`git checkout master`	3) Return to master

Useful things 9. Cherry-pick

I made a commit with some good fix in another branch. How could I take it and apply to my current branch?

Use git cherry-pick:

```
git cherry-pick good-commit-id
```

Useful things 10. Commit part of the file

I don't want to commit all my changes in file at once, rather few lines! How?

Do the

```
git add -p filename.java
```

It will interactively walk throug the file and ask which chunk should be staged.

Useful things 11. Bisect search

I've found a bug in my code. How to find when it was introduced?

Use git bisect for this:

git bisect start	Start bisect search
git bisect bad	Mark what the current commit is bad
git bisect good commit-id	Mark the commit we knew was good and jump somwhere to the middle
git bisect visualize	Look at where we are
git bisect bad	Mark bad commit
git bisect good	... Or good one
git bisect log	Take a look at the list of commits marked
git bisect reset	Return to original state

Or you can even automate the search:

git bisect run mytest testargs	Run **mytest** to determine good/bad

GUI

I hate command line! Any mouse stuff around?

Plenty of them:

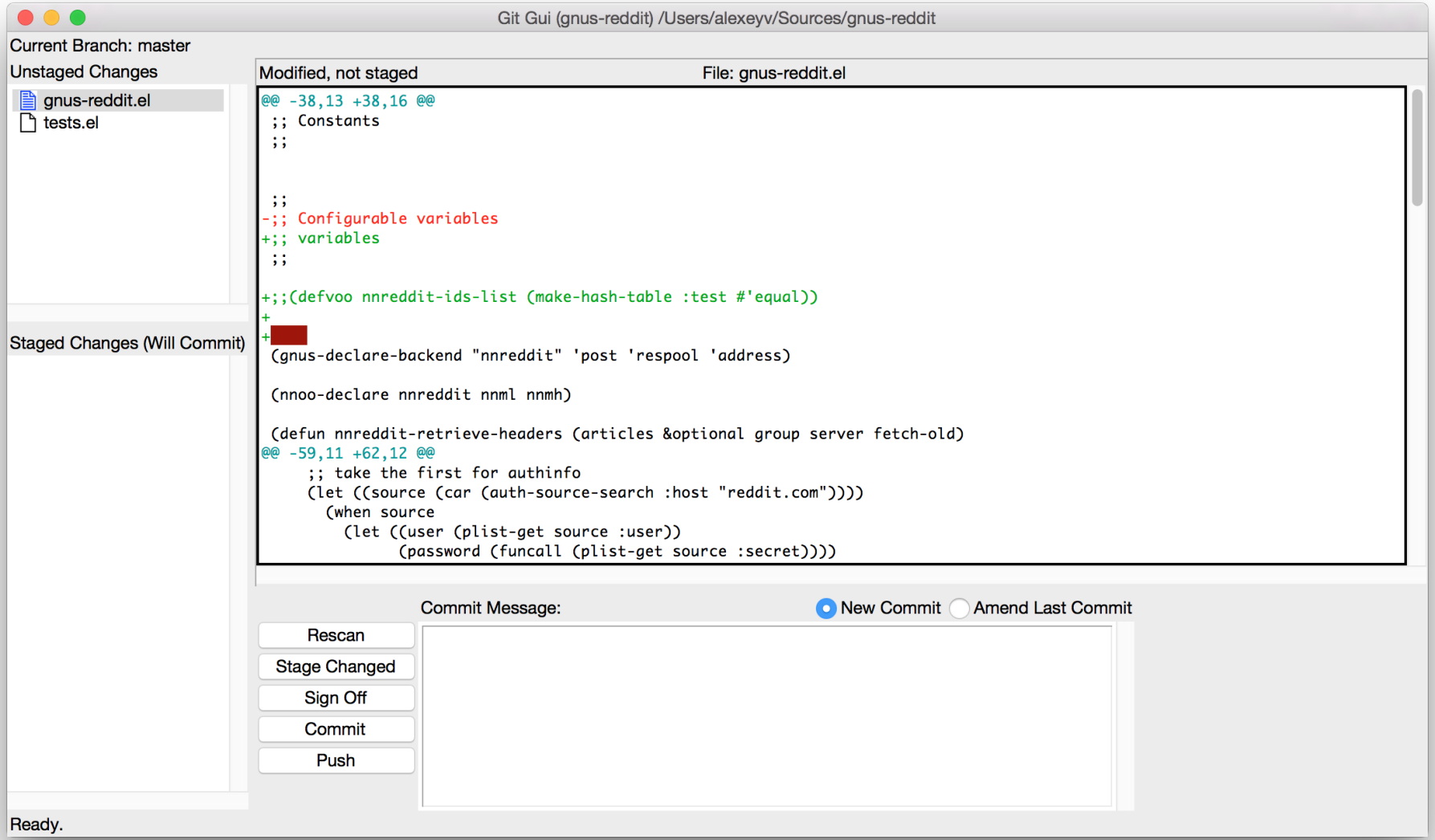
`http://www.git-scm.com/downloads/guis`

But it is worth to learn command line interface:

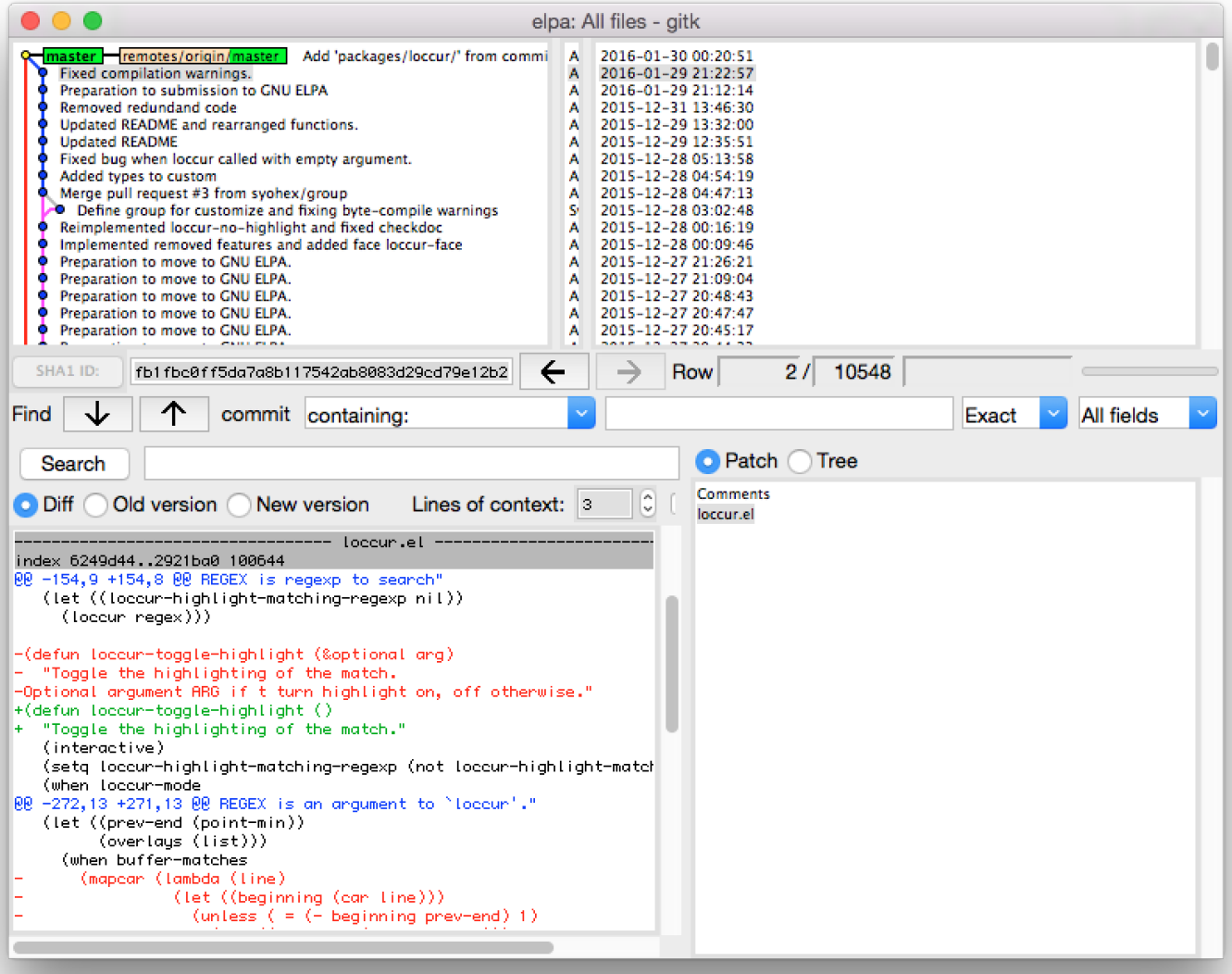
- It always works, no man-in-the-middle doing smart stuff for you involved
- You can copy-paste snippets from Stackoverflow
- History of all your git operations available:

```
history | grep git
```

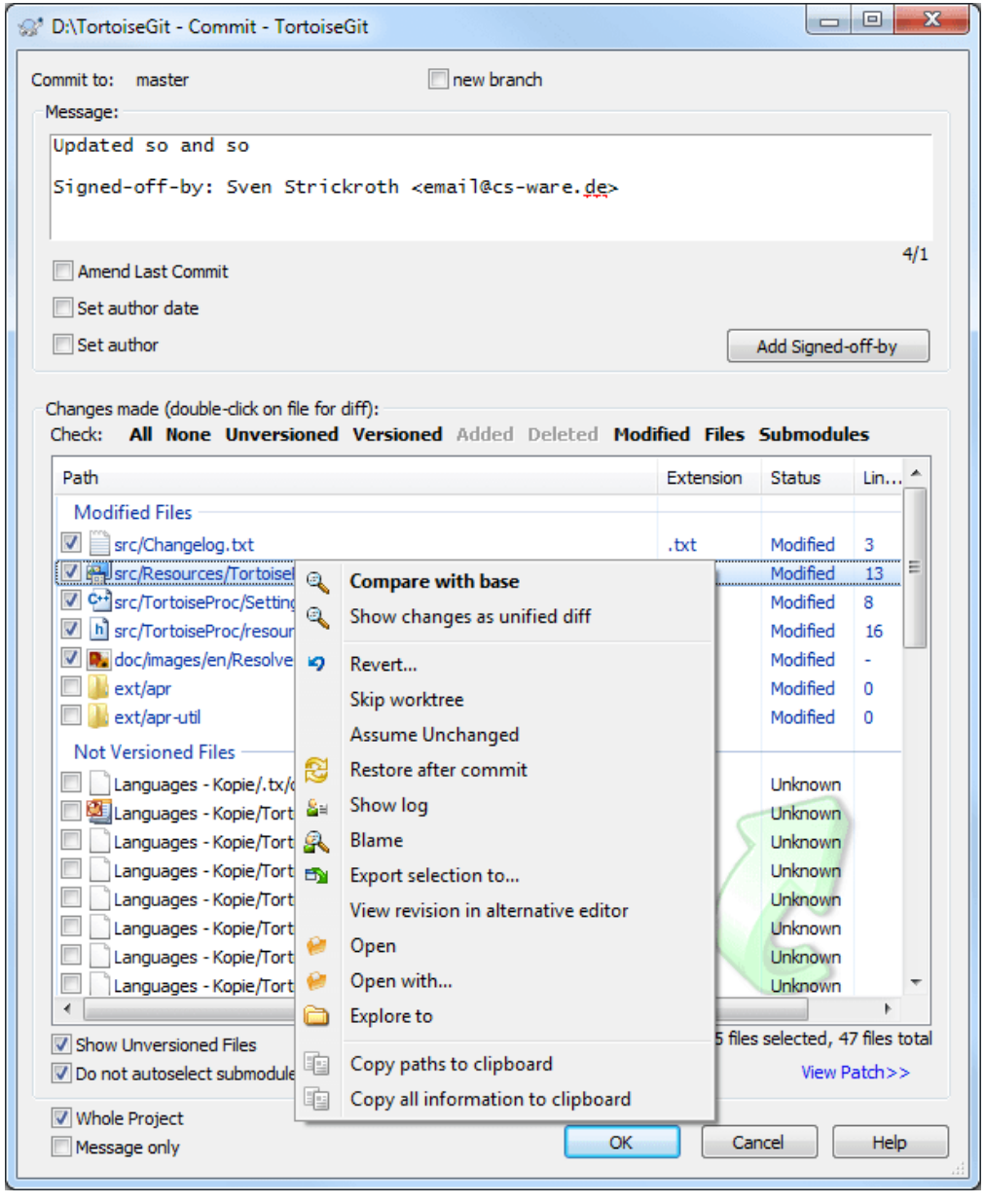
Default Git GUI - “git gui”



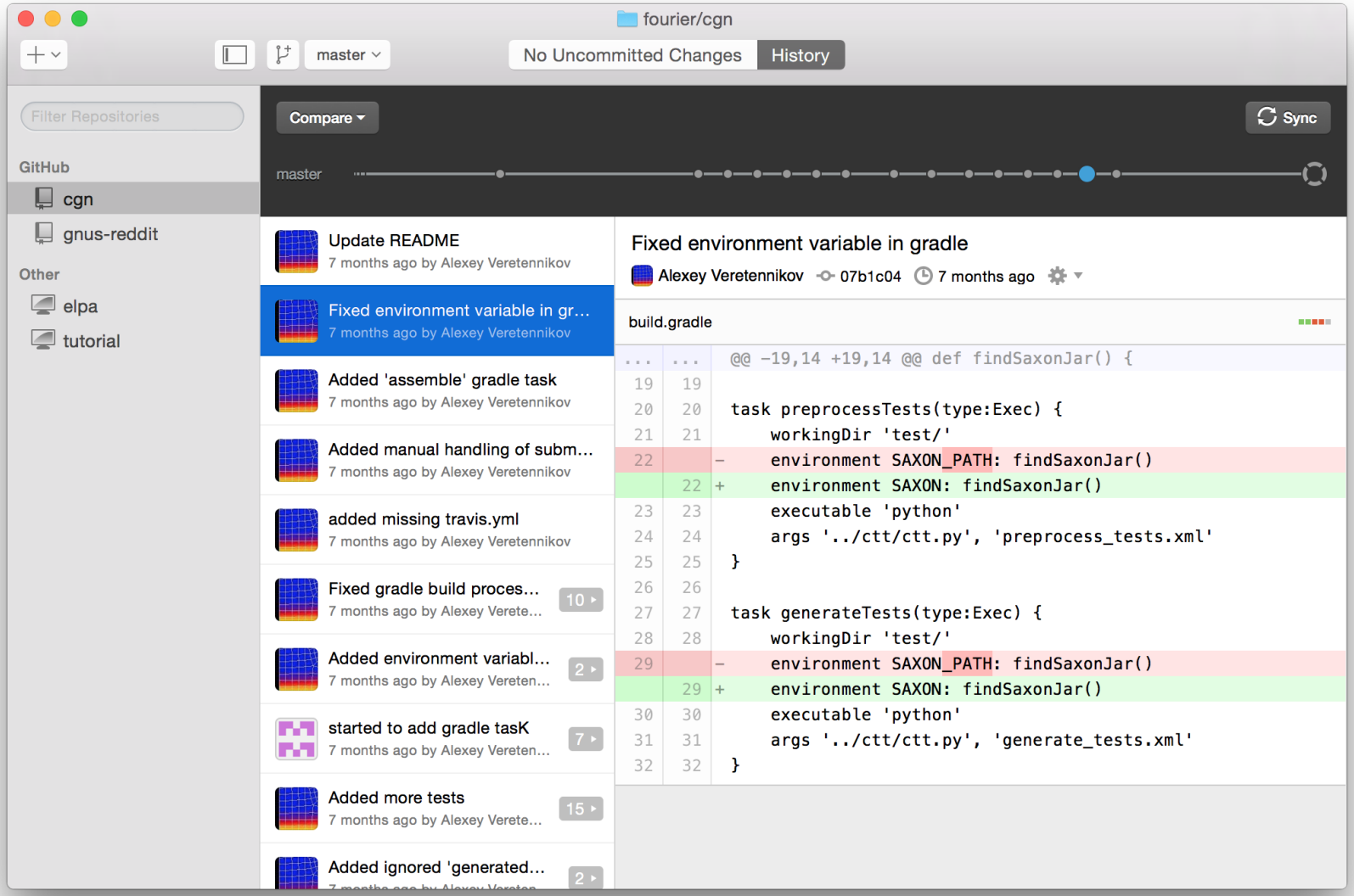
Default Git GUI - “gitk”



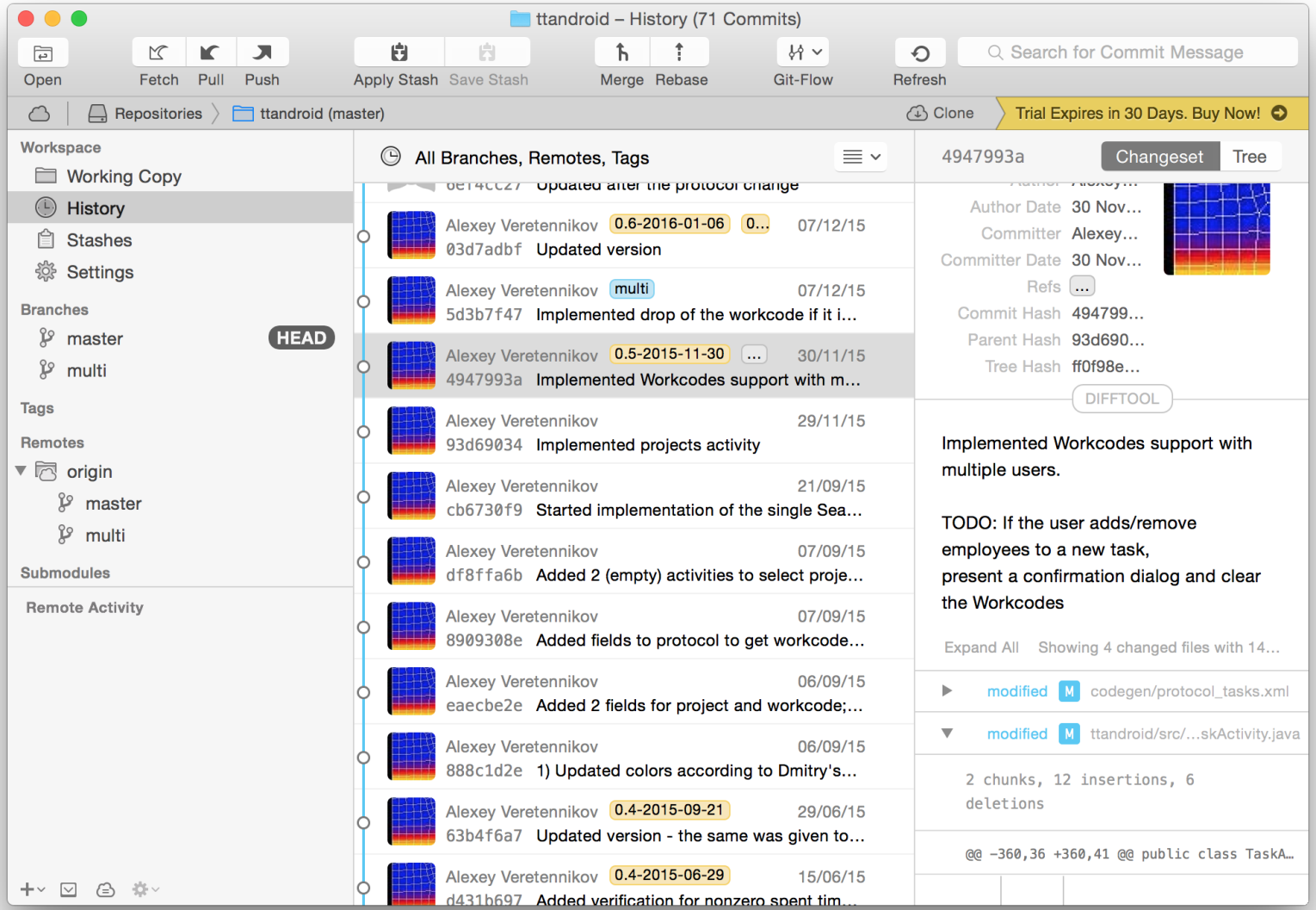
TortoiseGit - https://tortoisegit.org



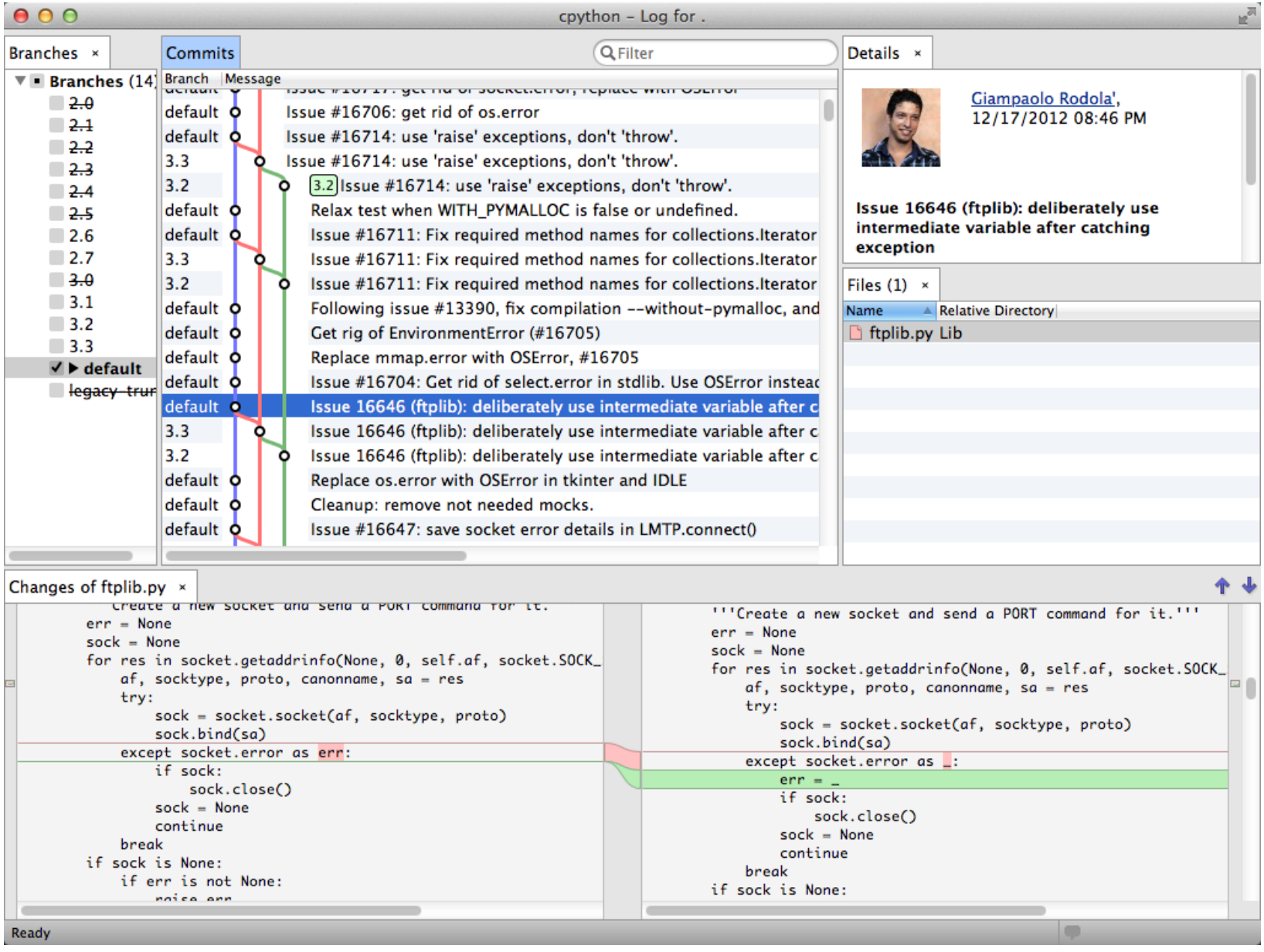
GitHub Desktop - https://desktop.github.com/



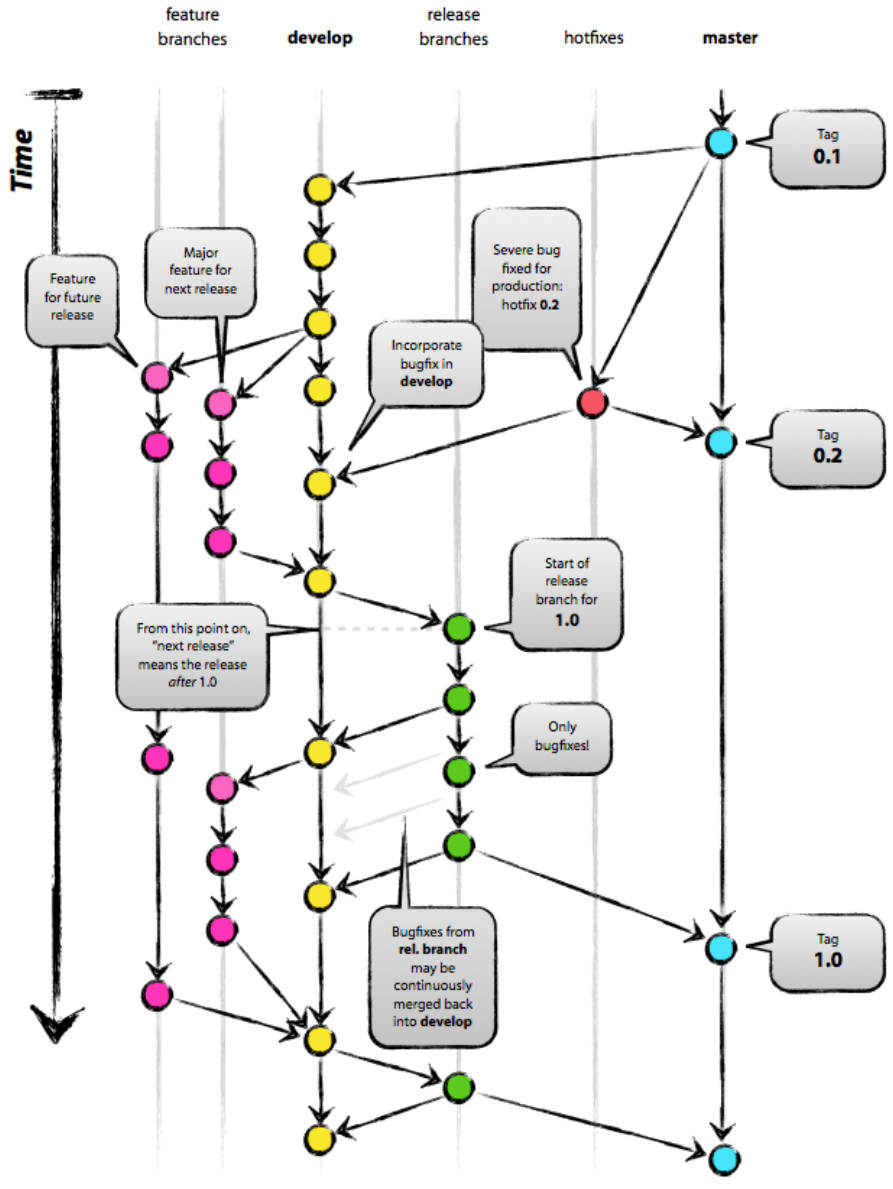
Git Tower(69$) - https://www.git-tower.com/



SmartGit(99$) - http://www.syntevo.com/smartgit/



Git Flow 1. http://nvie.com/posts/a-successful-git-branching-model/



Git Flow 2

- An organized way to work with the branches.
- Works in Git for Windows!

git flow init	Initialize git flow support in repo
git flow feature start cool	Create a feature branch
git flow feature finish cool	Close a feature branch
git flow release start 1.0.1	Start release branch
git flow release finish 1.0.1	Make a release
git flow hotfix start 1.0.1	Start hotfix for release
git flow hotfix finish 1.0.1	Finish hotfix for release

The end

And of course this presentation was made in LaTeXand version-controlled with Git!
Questions?