An introduction to Git

January 7, 2015

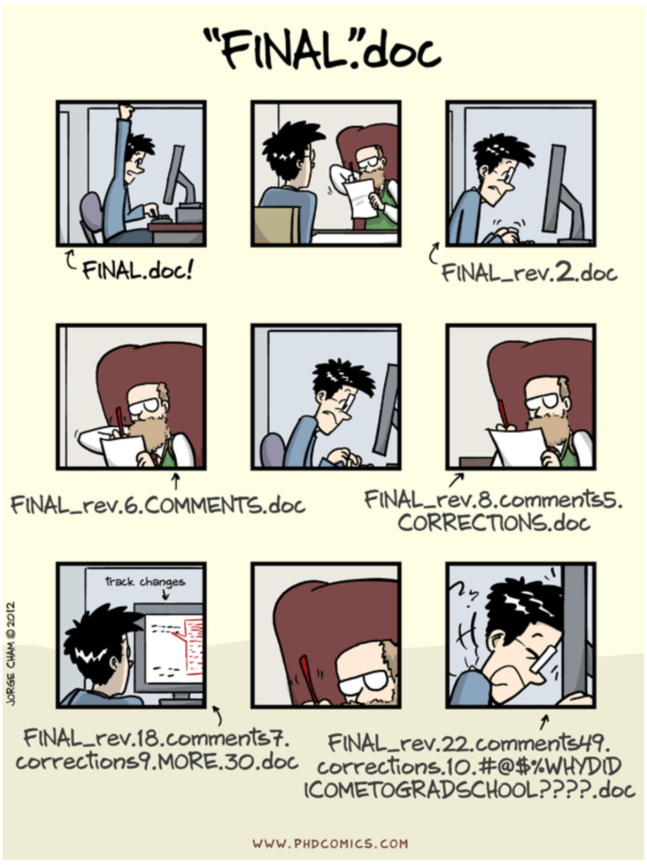
Department of Aerospace Engineering, IIT Bombay

Author: Prabhu Ramachandran

Agenda

- Introduction to git
- Using github and collaborating
- Workflows

Motivation


"FINAL".doc
FINAL.doc!
FINAL_rev.2.doc
FINAL_rev.6.COMMENTS.doc
FINAL_rev.8.comments5.CORRECTIONS.doc
track changes
FINAL_rev.18.comments7.corrections9.MORE.30.doc
FINAL_rev.22.comments49.corrections.10.#@$%WHYDID ICOMETOGRADSCHOOL????.doc
JORGE CHAM © 2012
WWW.PHDCOMICS.COM

Credit: www.phdcomics.com

History

- VCS: Version Control Systems
- RCS
- CVS
- SVN
- Centralized repositories

Distributed VCS

- Peer-to-Peer system
- Darcs
- Bitkeeper
- Mercurial: hg
- Monotone
- Bazaar: bzr
- Git

Collaboration

- Sourceforge.net etc.
- github/bitbucket etc.

Introduction to git

- Version control
    - save work
    - review changes
    - do not lose history
    - share with others
    - reduce mental burden
- Distributed workflow
- Requirement for modern software development!

Basic model

- A series of changesets (commits)
- HEAD is the last commit

Getting started

Setup your details:

```
$ git config --global user.name "Guru Programmer"
$ git config --global user.email "your_email@youremail.com"
```

Create a repository

Create a repo:

```
$ cd my_project
$ git init
```

Note that a `.git` directory is present!

Help!

Find help:

```
$ git help
$ git help merge
```

Status

Helpful status of repository:

```
$ git status
```

Often provides hints

Basic commands

Add a file:

```
$ vim readme.txt
$ git add readme.txt
$ git status
$ git commit
```

Changing the default editor

`commit` will use `$EDITOR`. Change this with:

```
$ export EDITOR="emacs -q"
$ export EDITOR=nano
```

Or

```
$ git config --global core.editor "emacs -q"
```

A note on commit logs

```
First line brief <= 50 chars

Detailed information below. Ideally wrapped to 72 cols.

- ALWAYS leave a good log message.

- Bullet points are fine.

- Multiple paras separated by blank line.
```

Review history

What happened:

```
$ git log
```

- Note the commit "ID"
- These are unique IDs

Notes

- What happens when you commit?
- What happens when you add?
- The staging area
- New files always must be added
- Remember to `git add`!

Making changes

Make changes:

```
$ vim readme.txt
$ git status
$ git diff
$ git add readme.txt
$ git commit
```

Some useful options

- Add all changed files and commit:

  ```
  $ git commit -a
  ```

Commit log on command line:

```
$ git commit -m "Fix for bug #123"
```

See changes in log:

```
$ git log -p
```

Exercise

1. Create a dummy repo.
2. Add some files.
3. Make different changes and commit them.
4. Review the log.

History

- `HEAD` is the latest
- `HEAD~1`, `HEAD~2` is one/two changes before
- You can use the commit IDs (or a unique substring)

```
$ git diff HEAD~1 readme.txt
$ git diff 737e86dd9 readme.txt
```

Differences between two points:

```
$ git diff HEAD~2..HEAD~4 readme.txt
```

Recovering old versions

Get the previous version:

```
$ git checkout HEAD~1 readme.txt
```

Same rules as before apply