A short introduction to version control with Git

Git, learn most of it

Simone De Agelis[1]

[1]Barts Cancer Institute
Queen Mary University of London

Bioinformatics cafe, 2021

Introduction

The Bioinformatics cafe

- Open to everybody with an interset in bioinformatics. from post-docs in computational labs to wet-lab students insterested in knowing more about bioinformatics
- For our purposes bioinformatics includes modelling/computationa biology
- We meet each Tuesday 4-5pm in the seminar room
- Email me to join our maling list
- or Slack workspace
- simone.deangelis@qmul.ac.uk

What is git?

What is git?

- Version control system
- Designed to track code changes in complex, collaborative projects
- Very valuable to track the state of smaller problems
- can roll back to an earlier verion when we break something

How will Git help me right now?

Advantages to version control

- Backing up your work.
- Undo-ing changes you've made.
- Sharing work with others.
- No more need to email code with successive renaming.
- Automatic handling of line endings between Windows and Mac/Linux.

What is github?

- like Git, but on the internet.
- User-friendly interface including visualisations.
- A social network of sorts.

Advantages to github

- Collaboration
- Share code with reviewers/readers
- Off-site backup of code..
- Open-source contribution.

How will Git help me right now?

github.com

- Open to the whole world
- Public repositories are visible to everybody
- Free but private reposititories require a premium plan

Advantages to github

- Fully managed by QMUL
- Suppported by QMUL RITS
- Unlimited private repositories
- Public repositories
- Free core service

Lets begin (0)

Git workflow

- Basic operations in git
- we'll see the idea behind it,then apply in the tutorial
- Same commands on Linux and MacOS

repositories

repositories are the basic units of git

- A folder to contain our files in
- Each repository can point to multiple remote locations (be it github.com or another disk)
- Git is distributed: each copy has the full history up to that point

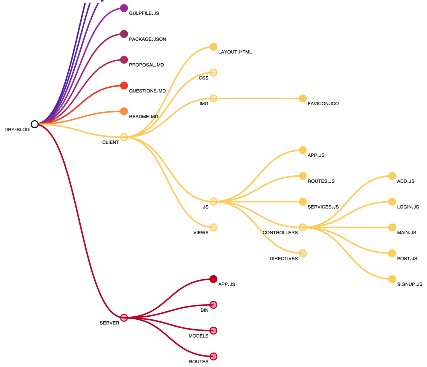
git follows a tree

Git workflow

- Our root is the first commit
- We can add changes on the same branch or diverge to form another branch
- We tell git what changes to record and where

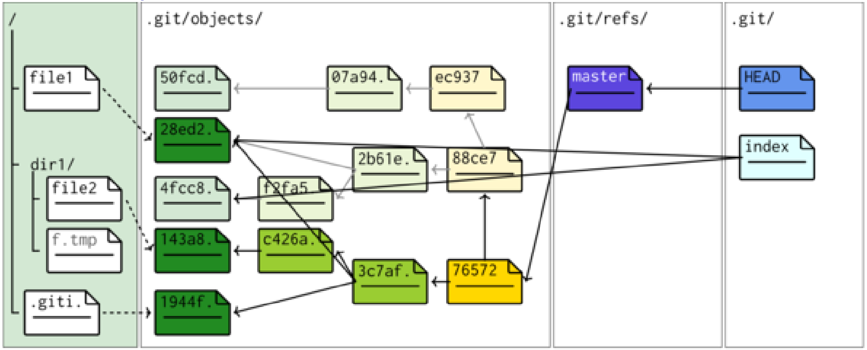
git follows a tree

a nice, smple tree



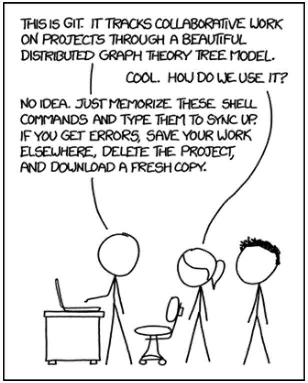
more complex tree

A more complex tree



so far so good?

Yes, we've all been there too


THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.

Let's begin (1)

Listing 1: we can clone a repository from somewhere

```
git clone <repository>
```

Listing 2: or create a new repository

```
git init
```

Making changes

Listing 3: Making changes

```
git status
git add <your filename>
git status
```

Differences and Reset

Listing 4: Making changes

```
git diff --staged
```

Listing 5: Reset changes

```
git reset <myfilename>
```

Forking on Github

Forking

- Not strictly a git command per-se. A github.com feature
- Creating our own version and copy online on github
- Related to the original

Remote Copies / Repositories

Remote

- A copy of the repository, complete and somewhere else.
- Could be github, or another directory on the same disk.

Listing 6: Reset changes

```
git remote add cis <address>
git@github.com:MYUSERNAME/gitclass.git
git remote --help
```

Committing changes

commit

- Possibly the most used command
- Staged changes are 'committed'.

Listing 7: git commit

```
git commit --help
git commit -a -m "<my message>"
```

Pushing Commits

push

- Pushing your commits to a remote repository.

Listing 8: git push

```
git push cis master
git diff
```

Checking History

checking

- Looking at the history of commits
- List of git commit messages and unique IDs

Listing 9: git log

```
git log
```

Removing files

Listing 10: Removing Files

```
git rm <your file name>
git commit -a -m "removed our new file"
```

Its all gone wrong(0)

wrong

- What do we do if we delete or change things and we want to go back
- We need to think about pointers and commit IDs

Its all gone wrong(1)

Listing 11: restoring things

```
git reset --hard HEAD~1
git reset --hard HEAD
```

Branching (0)

branching

- Probably the heart of Git - different code with a common history.
- Can 'branch off' from the main codebase to test things.
- Can merge back at a later date. Everything recorded.

Branching (1)

Listing 12: branching

```
git branch morespeare
git branch
```

Listing 13: make some changes

```
https://archive.org/details/
  gutenberg?and[]=shakespeare

git commit -a -m "Added more shakespeare to use"
git push origin morespeare
```

Merging

Merging

- The counterpart to branching.
- The trick is to recognise and resolve conflicts

Listing 14: make some changes

```
git checkout master
git merge morespeare
```

Conflicts

Listing 15: possible result of a merge

```
Auto-merging shakespeare_corpus.txt
CONFLICT (content): Merge conflict in shakespeare
Automatic merge failed; fix conflicts and then co
```

Listing 16: a conflict

```
<<<<<<< HEAD:index.html
<div id="footer">contact : email.support@github.c
=======
<div id="footer">
 please contact us at support@github.com
</div>
>>>>>>> iss53:index.html
```

Collaboration

Using remote repositories with correct access controls we can work collaboratively.

collaboration

- github.com is perhaps the most widely known.
- git-lab.
- gitolite with keys and a server.
- Any remote repository where you can get access.

Advanced Topics

Practical Exercise before we move onto advanced topics.

Stashing

Sometimes you want to temporarily store changes and come back to them later without commiting.

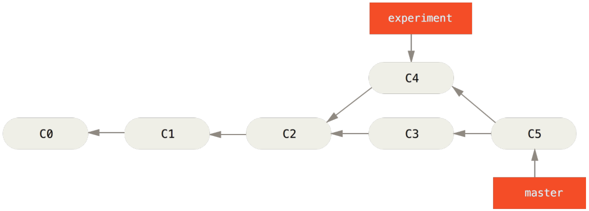
Listing 17: stashing

```
git stash
git status
git stash list
git stash apply
```

rebasing

Rebasing, in some ways, is another way to perform a merge (amongst other things). It *replays* changes on-top of a common ancestor.

example with merging

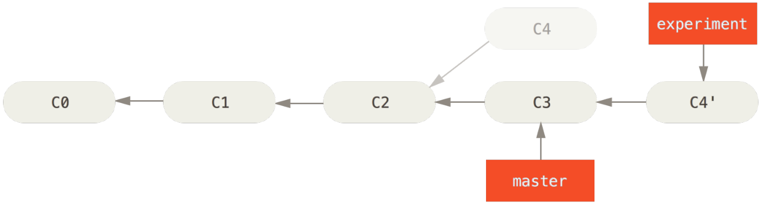


rebasing 2

Listing 18: rebase

```
git checkout experiment
git rebase master
```

example with a rebase

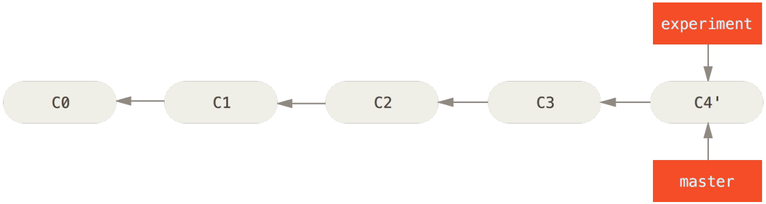


rebasing 3

Listing 19: rebase

```
git checkout master
git merge experiment
```

example with a rebase



tagging

Tags are good ways to mark commits. Also useful for people when they checkout your code.

Listing 20: tagging

```
git tag
git tag —a v1.4 —m "my version 1.4"
git show v1.4
```

Can create lightweight tags (just the checksum) by not adding -a.

diff

Tools to see the differences and create patches from these differences.

Listing 21: diff examples

```
git diff
git diff HEAD
git diff HEAD^ HEAD
```

Various other tools like vimdiff, and more advanced diffs across branches

Listing 22: diff examples 2

```
git difftool --tool=vimdiff --no-prompt \
  origin/togusa:.vimrc .vimrc
```

Flows

A way to organise your work. Use branches and tags to keep work organised.

- development/trunk
- stage/pre-production
- production/live

Integrating Git with workflow

Git and github work well with other tools. Automatic build-tools

- https://travis-ci.org/
- https://jenkins.io/index.html

Integrating Git with workflow 2

Can also automatically test code upon commit ...