Git Gud or Git Goin'

An introduction to the Git version control system for managing scientific code and experiments

Stephen Malina

May 19, 2020

What we're going to talk about

1. Introduction
   - What is 'version control' and why should I care?
2. Git Basics
   - Git Concepts: A *Very* Brief Overview
3. Sharing is caring: Collaborating with Git(Hub)
   - Overview of Github
4. Best Practices Speedrun
5. Advanced Resources

Introduction

What is this 'version control' thing?

Literally:

- Track changes to a set of files over time
- Manage multiple partially overlapping versions of a project
- Allow people to collaborate on a project in a distributed, asynchronous fashion

Figuratively:

- 'Track changes' for code
- A phylogenetic tree for your project
- A digital lab notebook

Why bother?

Have you ever:

- Accidentally deleted a file?
- Changed some code, run an experiment, and realized something went wrong but you don't know what?
- Wished you could remember how you were doing something two months ago?
- Forgot to save the results of a (computational) experiment?
- Gotten frustrated trying to share code or a manuscript by emailing versions back and forth?

If yes to any of the above, then version control is for you!

What about this 'Git' thing?

Git is a *distributed* version control system[1].

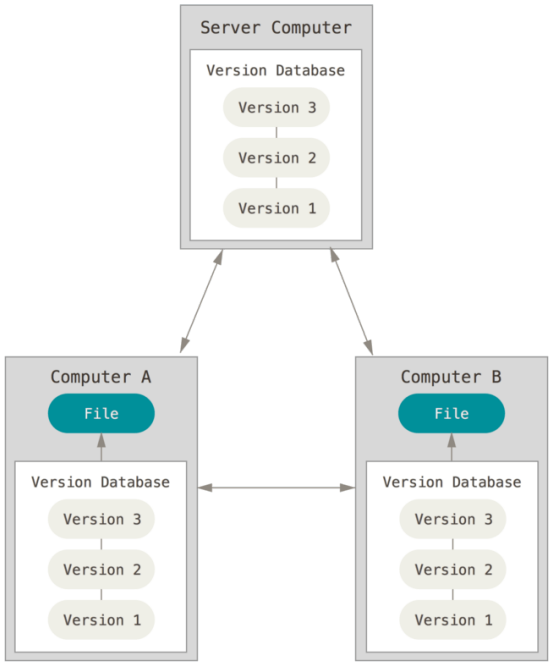


Figure: Source

[1]This is *not* important to know as a user of Git. I have used Git for 7 years now on a quasi-daily basis and still couldn't tell you much about differences between Git and other version control systems like CVS, SVN

Meta: What are we hoping to accomplish today?

- Learn to use Git well enough that you can use it for your own projects
- Understand conceptually the data structure that Git uses to manage your project
- Get exposure and pointers to additional resources for more advanced and specific use-cases

I tried to write this as though I was writing to myself when I first starting using Git.[2]

[2] My Git learning process was a more like being thrown into the deep end, but that's a story for another day...

Git Basics

Git Concepts: A *Very* Brief Overview

The minimal set of concepts you'll need while working with Git[3]:

1. Repository ('repo')
2. Staging area
3. Diff
4. Commit

Let's start by using them in a sentence. "Open up our project in the *repo*, change some code, look at the *diff*, add changes to the *staging area* and *commit* them."

[3] Locally that is. We'll get to how this changes once collaborators enter the mix later.

Repositories

Repositories are folders in which Git manages a set of files as a single unit

Easy way to think about this is that each repository is typically one project

Using the phylogenetic tree analogy, repository is to Git as ecosystem is to evolution

Using the lab notebook analogy, repository is to Git as individual notebook is to lab notebook

Repositories

Let's make one

1. Open up the command line (Terminal on Mac)
2. (Optional) Change to a directory in which you want to create a new sub-directory (I use `~/dev/`)
3. Create the new directory which will be our new repository

```
mkdir -p ./git-tutorial
cd git-tutorial
```

4. Check that you have Git installed (you almost definitely do)

```
git --version
```

5. Initialize the Git repository

```
git init
```

Diffs, staging area, and commits

From the top down

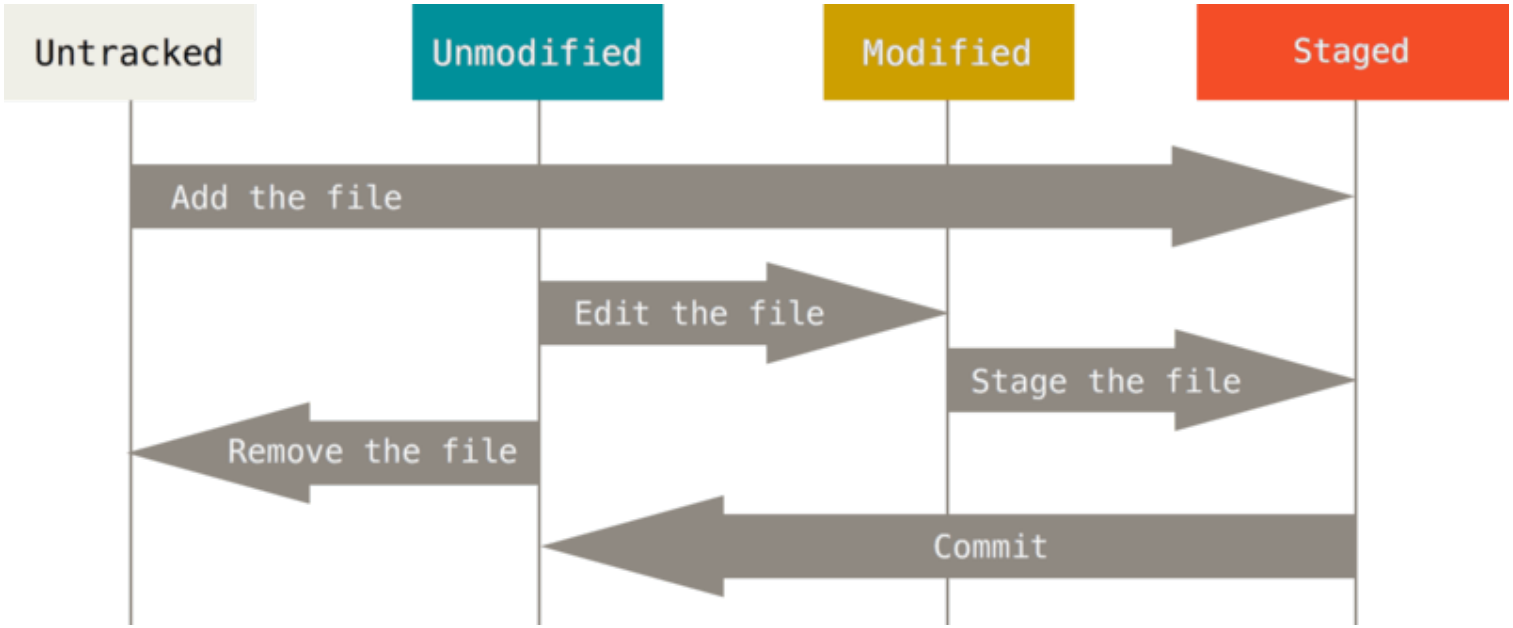


Figure: Source

Diffs, staging area, and commits

Do it ourselves

Imagine we have some code in a repository

We currently haven't made any changes

Now, we let's change some code

Diffs, staging area, and commits

Inch by inch

1. Create a new file and write something inside it, e.g.

   ```
   echo 'Stephen rocks!' > stephen-is-awesome.txt
   ```

2. Now let's see what Git thinks of this

   ```
   git status
   ```

Diffs, staging area, and commits

Row by row

Our goal is to package this code up in a unit of work (*commit*)

First, we have to decide which code is going to go in this unit of work

That's what the staging area's for

Diffs, staging area, and commits

Gonna make this garden grow

1. Git status is Git's way of telling us its understanding of the current state of the repository

```
git status # again
git diff
```

2. Add all files to the staging area

```
git add .
```

3. Check again

```
git status
git diff
git diff --cached
```

4. Notice the change

Diffs, staging area, and commits

All it takes is a rake

1. Now we're ready to package up our work

```
git commit
```

2. Let us look upon our work

```
git add .
```

3. Check again

```
git status
```

4. Notice the change

Diffs, staging area, and commits

Let's commit this! Apparently this will not be challenging for any of you based on the survey results...

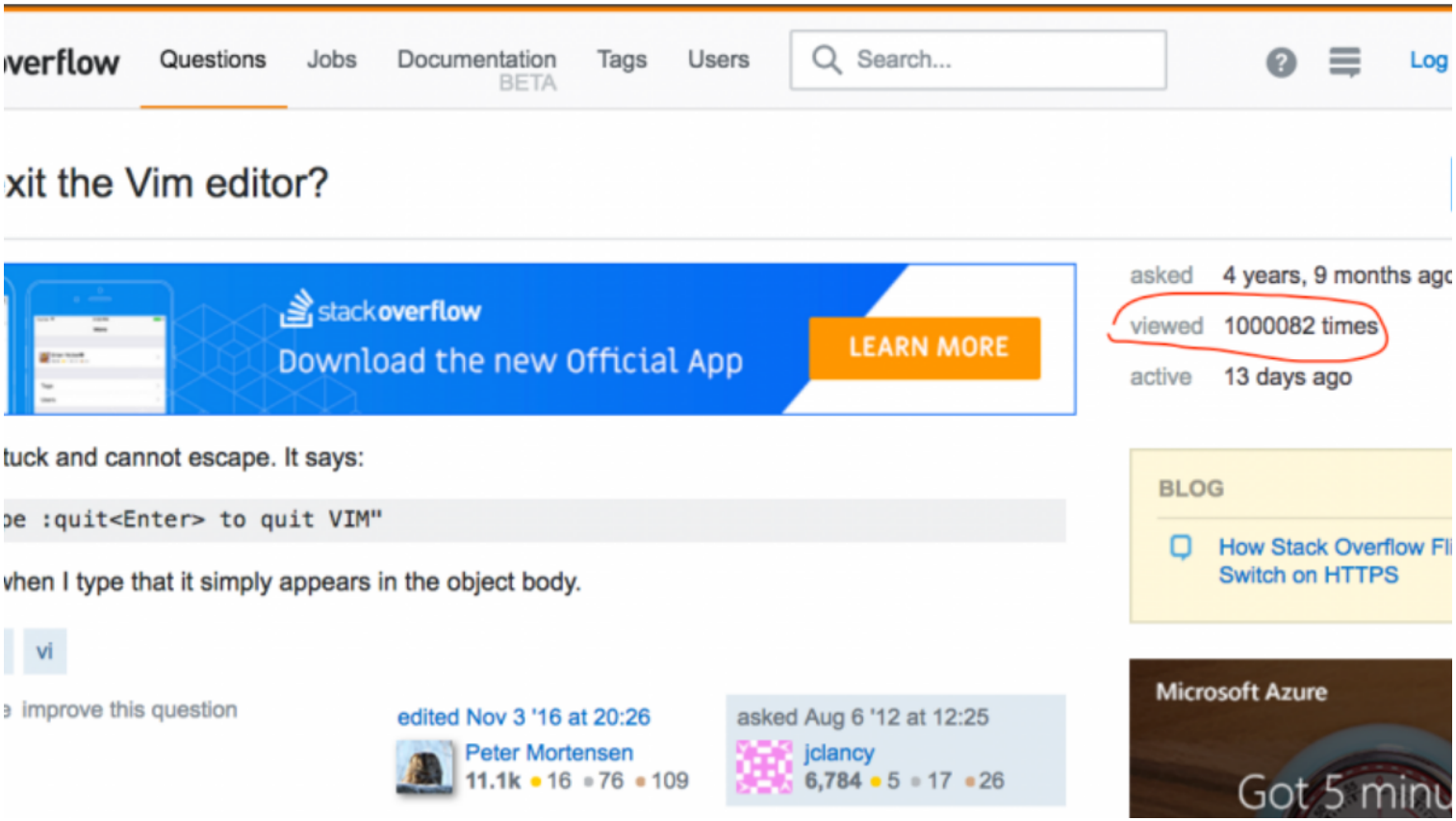


Figure: Source

Commits, diffs, & staging area

1. Commit

```
git commit
# Write a message
# Exit your editor
```

2. Check out the log

```
git log
```

3. Check out the log (V2)

```
git log --pretty=oneline --graph
# Reminder: show a fancier log
```

A little more about commits

Commit = unit of work (set of changes to files)

Commit structure:

```
A<-B<-C<-E<-F
      ^     |
      ^--D<--
```

Commits have: authors, timestamps, SHAs

```
commit f7a7660158b969a10c8787d9d2a152fa005f9f6e (HEAD ->
master)
Author: Alex Facciorusso
Date:   Thu Mar 15 00:00:00 2018 +0000

    Chirp chirp; closes QWAK-123
```

You're an expert on local development with Git now!

Exercise

Pick a project on your computer, initialize the Git repository and create your first commit.

Things to watch out for:

1. You'll have a lot of files, try only adding a few
2. If the repository contains non-text files like PDFs, you may want to consider adding them to gitignore

Misc notes on local development with Git

Git log is very customizable

```
* 525a852 - (3 years, 5 months ago) notification test - German Laullon (origin/test_on_10.8)
* 58b5a57 - (3 years, 5 months ago) morden Objective-c - German Laullon
* aef1d39 - (3 years, 5 months ago) first build. - German Laullon
*   bff6661 - (3 years, 5 months ago) Merge pull request #169 from taybin/patch-1 - German Laullon (HEAD, origin/master
|\
| * ef61b97 - (3 years, 9 months ago) Update Rakefile to work with ruby-1.9.2. - Taybin Rutkin
* |   3b06317 - (3 years, 5 months ago) Merge pull request #175 from barrywardell/master - German Laullon
|\ \
| * | 907e967 - (3 years, 11 months ago) Fix bug where submodules were incorrectly grouped when the first part of their
* | |   dd1b324 - (3 years, 5 months ago) Merge pull request #195 from jphalip/master - German Laullon
|\ \ \
| * | | 8ebb58c - (3 years, 6 months ago) Added "Copy Reference to Clipboard" context menu item in sidebar. - Julien Ph
| * | | 238a97a - (3 years, 6 months ago) Fixed a tiny typo. - Julien Phalip
| * | | 3386fc7 - (3 years, 6 months ago) Make sure the commit view gets refreshed when 'Stage' gets selected and the a
| * | | 7fafdb8 - (3 years, 7 months ago) Tweaked capitalization of words in contextual menu items. - Julien Phalip
| * | | 5958f5b - (3 years, 7 months ago) Don't display all the files selected for deletion to avoid the confirmation s
| * | | 3575e87 - (3 years, 7 months ago) Prevent large files from getting loaded in the commit view to prevent the app
| * | | 748621c - (3 years, 7 months ago) Made the "(Un)stage lines" functionality in the commit view work only if the
| * | | d249fd6 - (3 years, 7 months ago) When a branch gets selected in the sidebar, make sure its corresponding commi
| * | | 839c9b6 - (3 years, 7 months ago) Made the diff tables adapt nicely to the window's size. Fixes #50. - Julien P
* | | |   8badd8a - (3 years, 5 months ago) Merge pull request #196 from Kyriakis/master - German Laullon
|\ \ \ \
| * | | | af773ba - (3 years, 7 months ago) No check for refs on remotes while deleting them - Robert Kyriakis
| |/ / /
* | | |   47a8a1a - (3 years, 5 months ago) Merge pull request #197 from Uncommon/arcfix - German Laullon
|\ \ \ \
| * | | | d1a8ba9 - (3 years, 6 months ago) fix for ARC errors and other warnings - David Catmull
| |/ / /
* | | | b94248c - (3 years, 5 months ago) [y so slow] :D - German Laullon
|/ / /
* | |   8ce13ad - (3 years, 7 months ago) Merge pull request #194 from jphalip/master - German Laullon
|\ \ \
| * | | 3e045a4 - (3 years, 7 months ago) Ensure that the previously selected commit remains selected after refreshing.
|/ / /
* | |   aae5e7c - (3 years, 7 months ago) Merge pull request #192 from jphalip/master - German Laullon
|\ \ \
| * | | cd2e8de - (3 years, 7 months ago) Made the sign-off button optional and hidden by default, as this is a feature
```

Almost everything is reversible

Roll-back staged changes with `git reset` and `git checkout --`

Roll-back commits with `git revert` and `git reset`

Rewrite history with `git rebase`

If you've committed it, it's almost always recoverable

`git reflog` is your friend

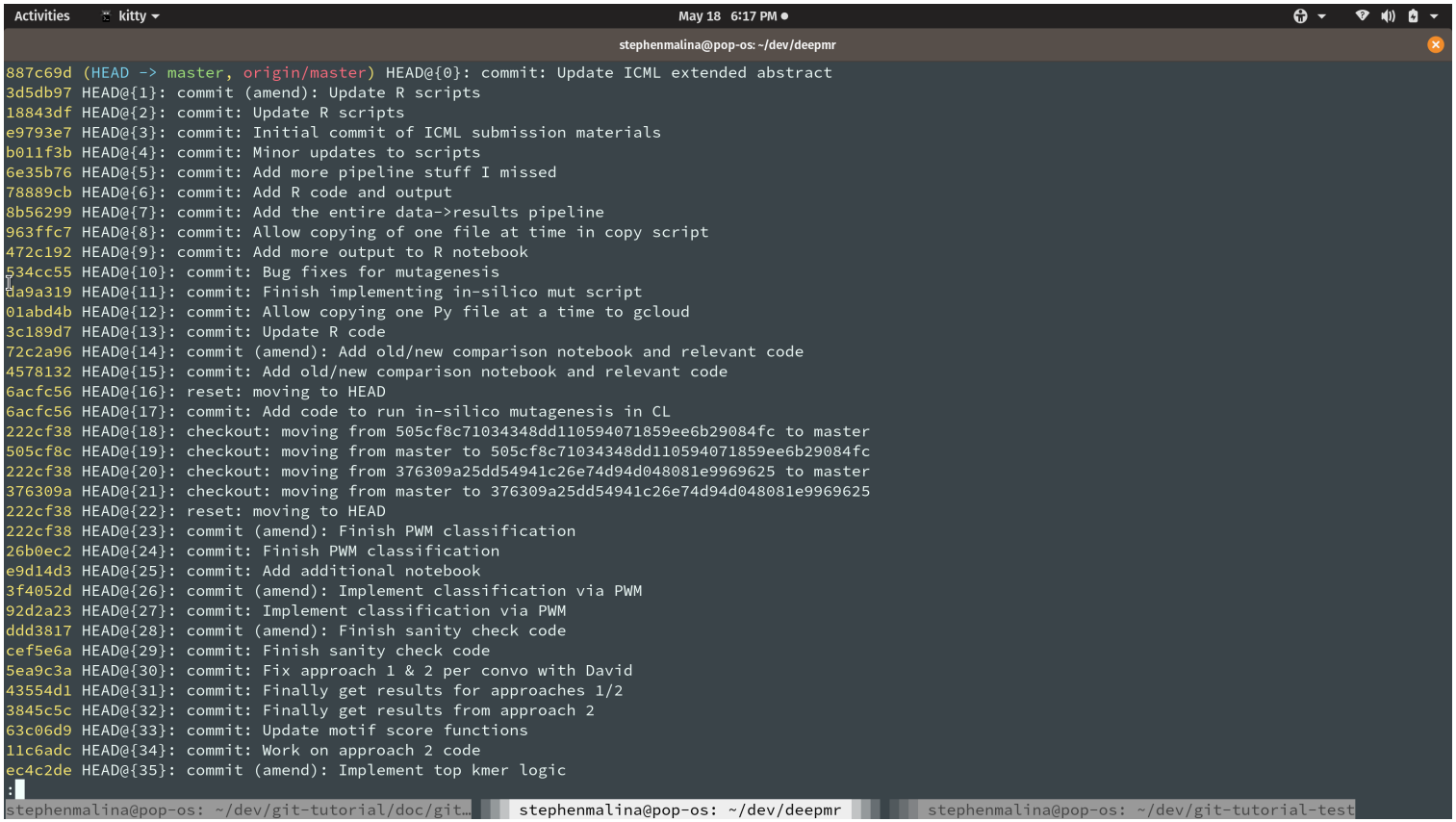


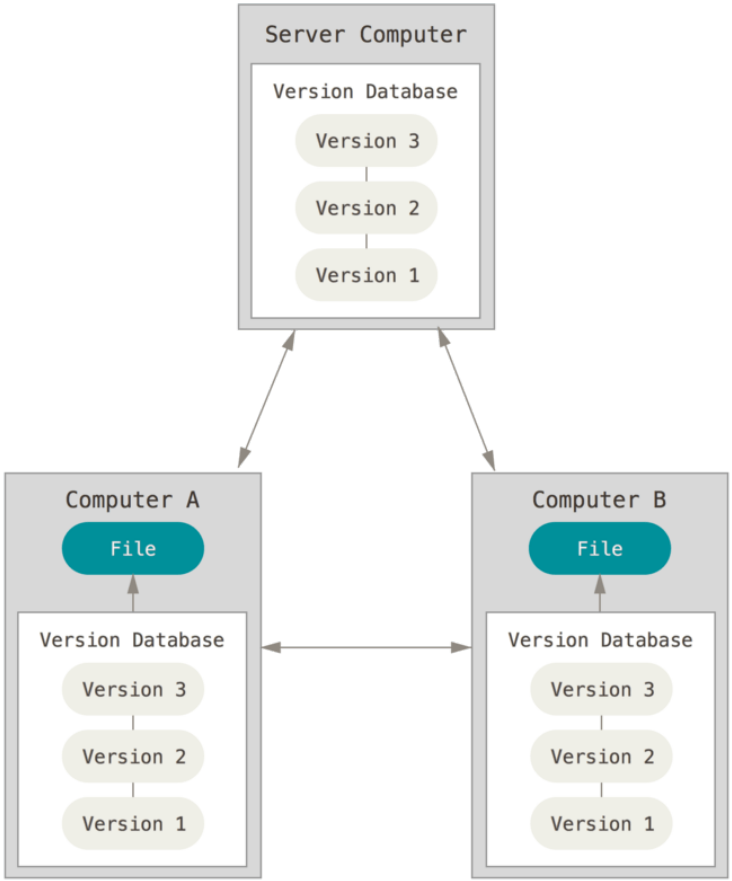
Figure:

Sharing is caring: Collaborating with Git(Hub)

Collaborating over the interwebz

So far, we've been working with local repositories

Now we're going to talk about interacting with the second portion of this picture



What are we trying to do when we collaborate?

Fundamental challenge: We're all writing code at the same time and trying to incorporate each other's changes.

But if we just combine our different versions, then it'll be mayhem

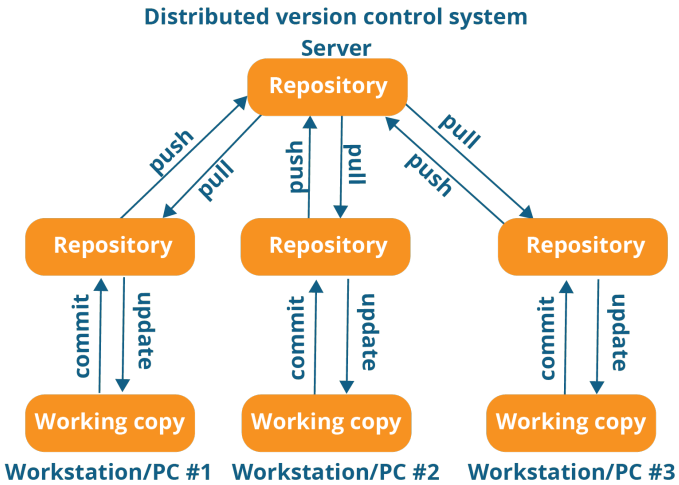
Git solves this with remotes and branches (remotes)

Solution: Moar (local **and** remote) repositories

We call the copy of our repository on a server a *remote*

Syncing to the remote = *push*ing

Syncing from the server = *pull*ing

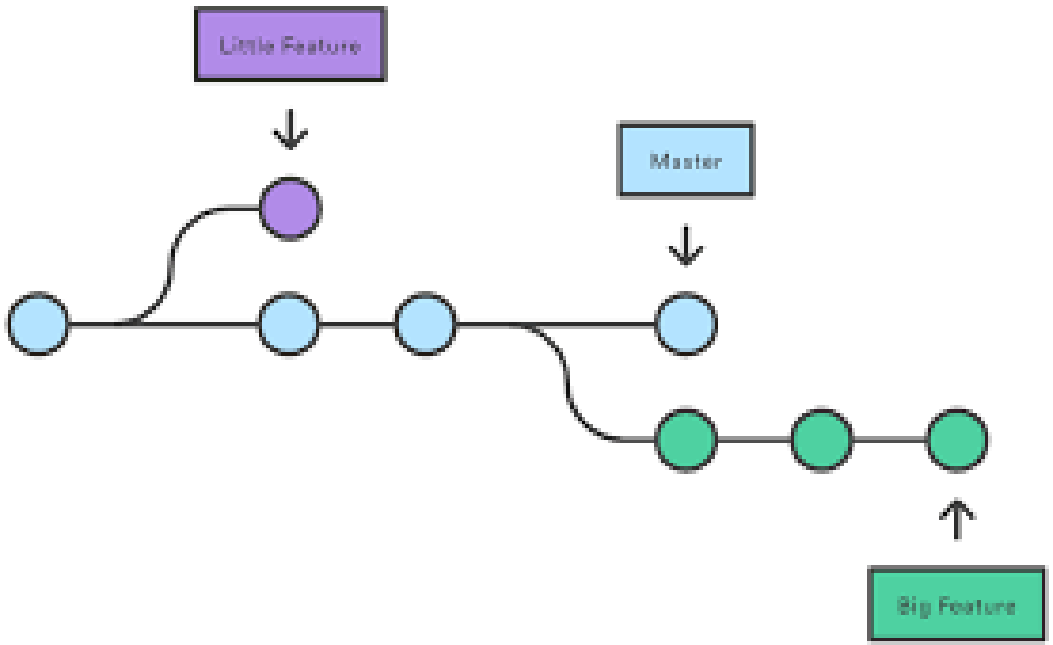


Git solves this with remotes and branches (branches)

Up until now, we've been treating Git history as a series of linear commits that build on top of each other

However, there's often not just one version of our code locally and on the remote

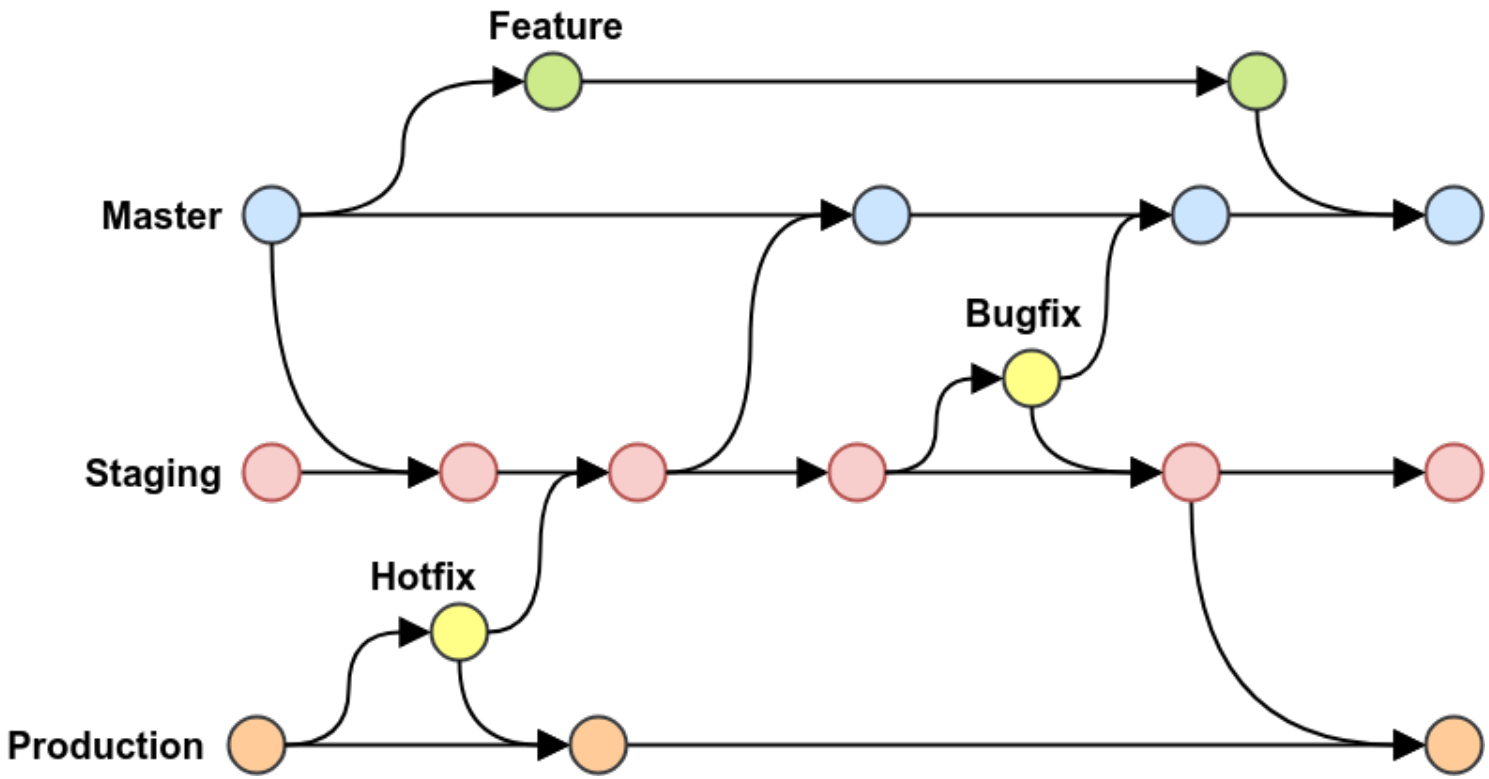
Instead, branches allow us to split our code into multiple concurrent versions both locally and on the server



What are branches good for?

Branches can be useful for:

- Working on multiple different things in parallel
- Sharing experimental code with collaborators without permanently adding it to the project
- Cutting versions of a library



Merging, the final piece of the puzzle

But of course, we also need a way to take whatever we've worked on in a branch and recombine it with our main (`master`) branch

Merging lets you do that

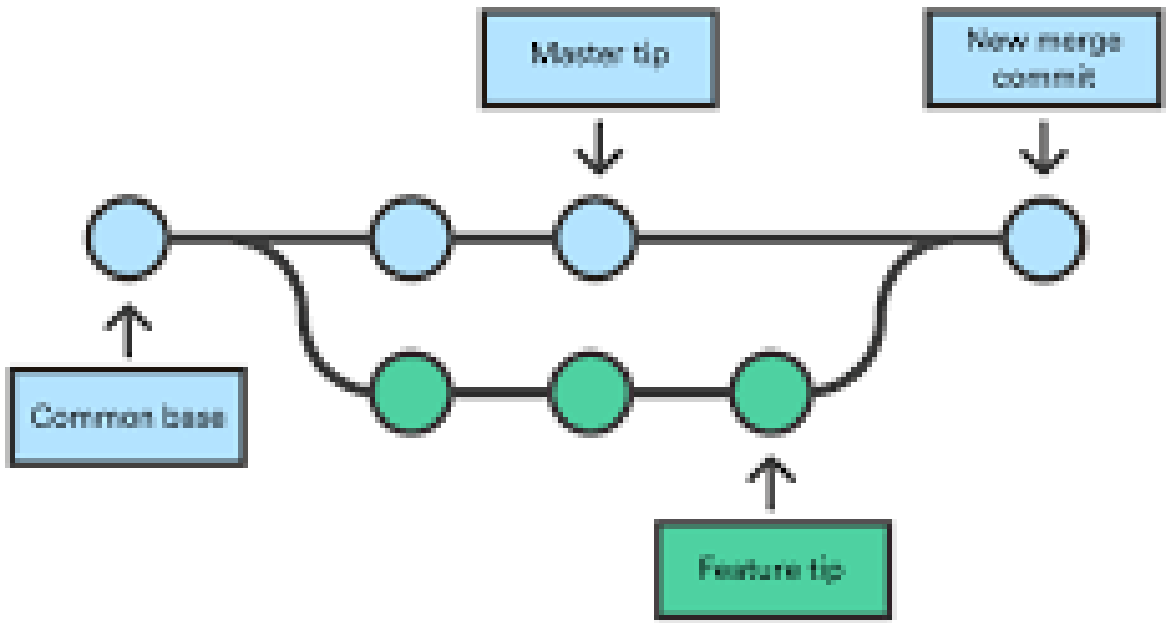


Figure: Source

Github

Github is just a hosted Git remote that does all the configuration for you

Pull Requests

Self-reminder: Linux kernel anecdote

Pull requests are a way to merge code into your own or someone else's remote repository through Github

At the core, they rely on the same fundamental concepts we described above

Enough talk, let's try it! I

In this exercise, you're all going to create a pull request to the repository in which I created this project.

1. If you don't already have one, create a Github account
2. Go to this link and click 'Star'
3. Now, click 'Fork'
4. Go to your own version of this repository and click 'Clone or download'
5. Open your command line and do

```
git clone <URL you copied>
```

6. To be safe, pull from the repo to make sure you're on the most up-to-date version

```
git pull origin master
```

7. Open README.md and add your name there

Enough talk, let's try it! II

8. Go through the same steps we took before to create a commit
9. Create a new branch

```
# Branch name is arbitary
git checkout -b <your initials>/acknowledg
```

10. Push the branch

```
git push origin <branch name from prior co
```

11. Click the link it gives you for creating a pull request and follow the remaining instructions
12. Pull again to update your local version of the code

```
git checkout master
git pull origin master
```

Yay! Now you can go and fix all the broken software on the internet!

Summing up

We've learned a basic workflow for working locally with Git, pushing to and pulling from a remote Github, and contributing to other projects. We discussed:

- Diffs
- Staging area
- Commits
- Remotes
- Branches
- Merging

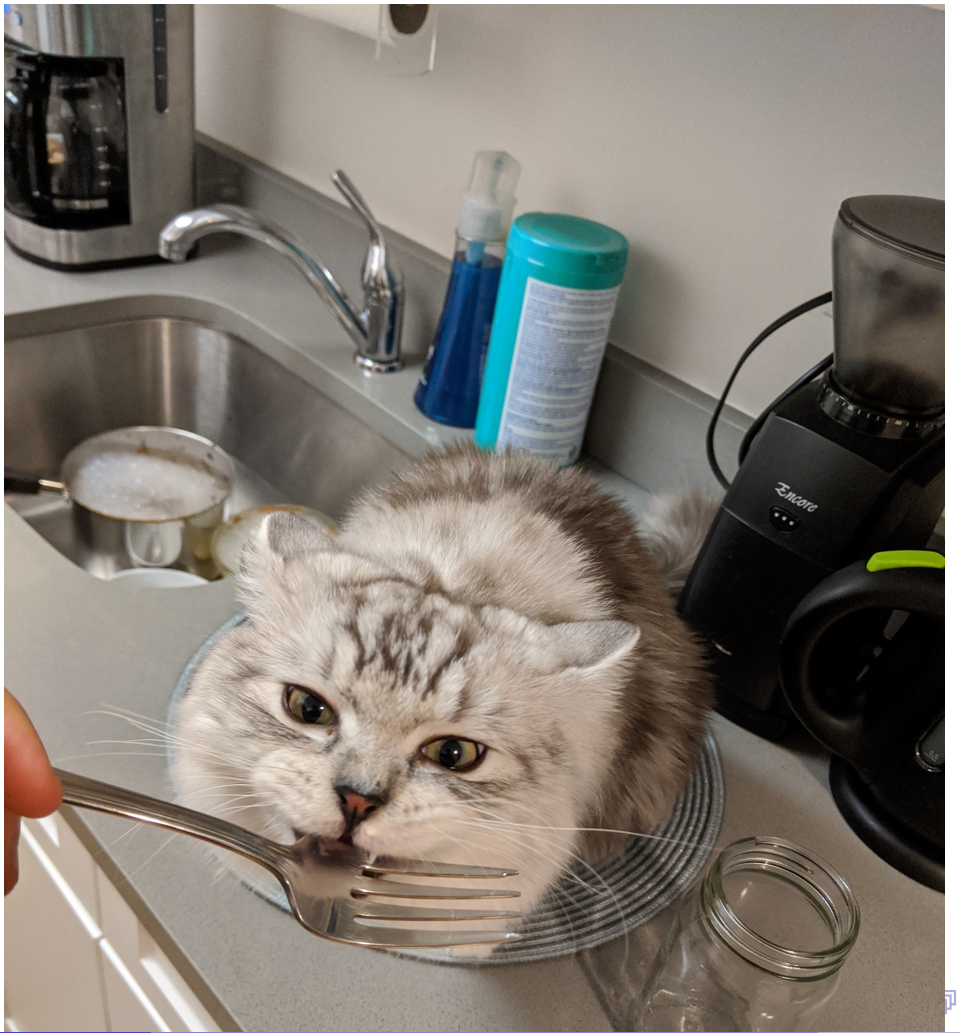
Best Practices Speedrun

For local development

- Commit early, commit often
- Try and visualize what you want to do in terms of how it changes the Git tree and then backchain the operations you need
- Be nice to your future self
  - Write informative commit messages
- Let the commits reveal the bugs (advanced)
- Follow a consistent commit message format to make git log look nice
  - See here and here

For collaboration

- Push early, push often
- Pick a Git flow you like (see here for options) and make sure all of your collaborators use it [4]



Advanced Resources

Deep dives into Git

- MIT lost semester blog post on Git: Good, slightly more in-depth description of Git. Also has lots of links to even more in-depth resources
- Explain Git with D3: Interactive playground that lets you visualize what commands do to the Git tree as you run them. Conflict of interest notice: I'm friendly with the author (former coworker).
- Pro Git: Detailed book describing how to do almost anything you could ever want with Git and also a bunch about Git's internals.
- Architecture of Open Source Applications chapter on Git. Describes the fundamental data structures that underlie Git and compares their performance to those of other VCS like Subversion.

Git for Science

- Greene Lab's Github Page with examples of using Git(Hub) for:
  - Reproducible computational experiments
  - Github-based Onboarding
  - Manubot for collaborative manuscripts
  - Deep Learning for Precision Medicine Review Paper written entirely on Github
- Github plus static site hosting as a lab notebook

Questions, Comments, Etc.

Figure: