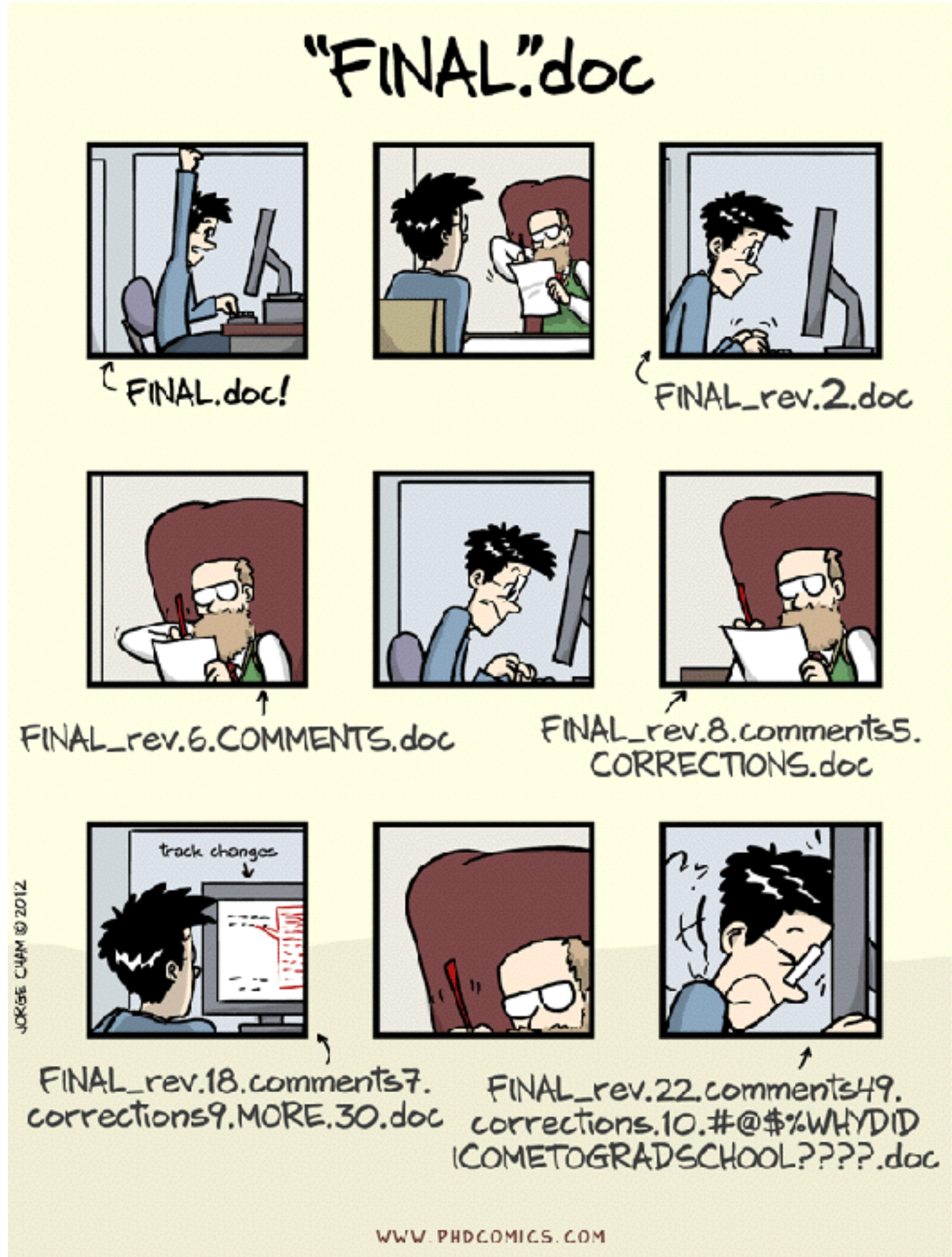
Version Control

Brittany Howard
30 January 2019



Learning objectives

- What is version control? Why should I use it?
- Overview: how does version control work?
- How do I set up and use my own repository?
- How do I use version control to work with a team?

What is version control?

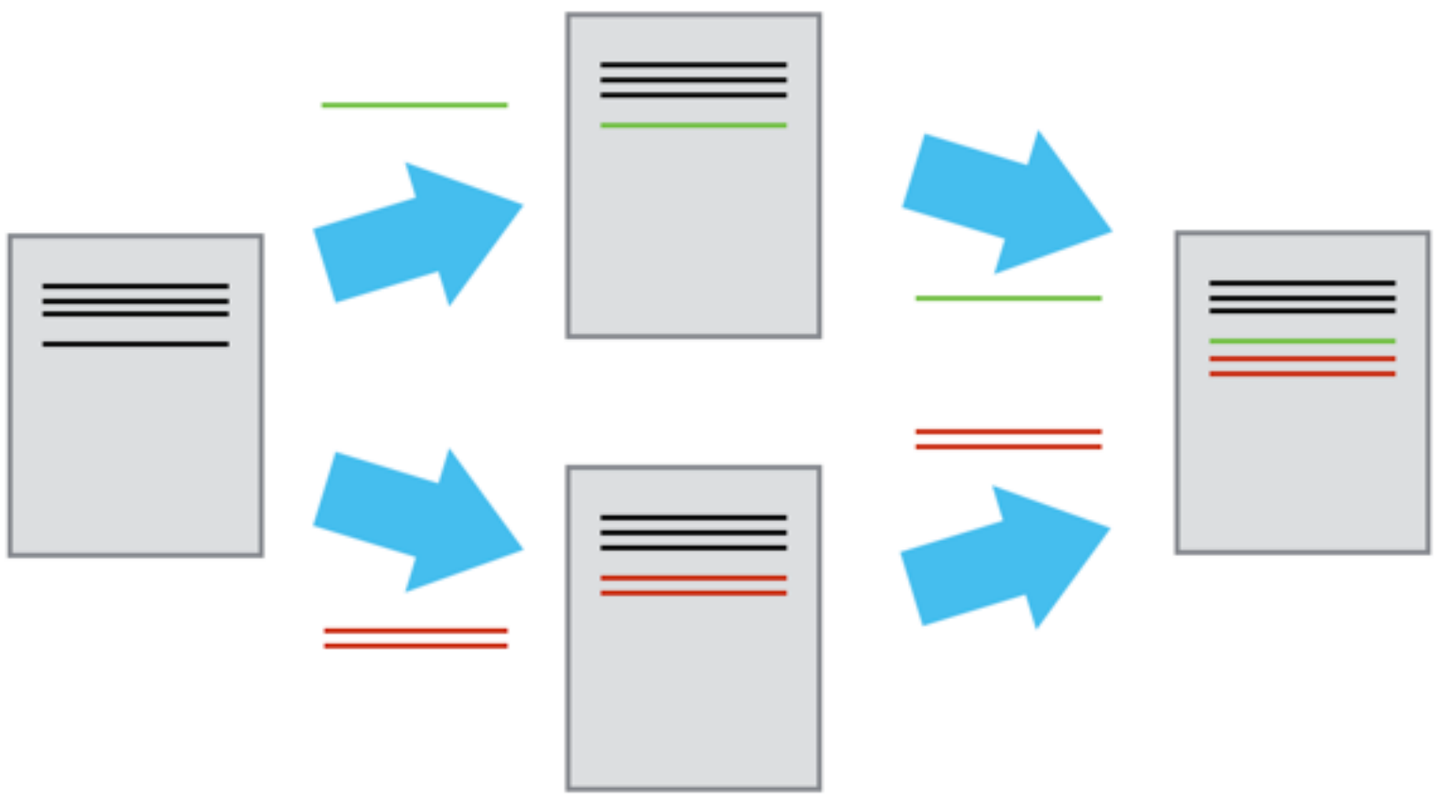
- A means of tracking the changes made within a repository.
- Allows many people to make changes to the same document simultaneously.
    - Integrates everyone's changes to make one master document that evolves with time.
    - Like a more powerful Google Docs.
- Useful for coding projects, but also for other projects like papers/theses.

For a single user

- Version control is like an unlimited ‘undo’.
- A digital lab notebook -> reproducibility.

For a team of users

- Version control allows many people to work in parallel.



Version control software

- Git
  - A little more technically complex, but more powerful.
  - Can convert to Mercurial with no data loss.
- Mercurial
  - A bit more streamlined, user-friendly.
  - Cannot convert to Git without data loss.

Getting help

- If you forget a command:
  - $ git config -h
- To see the manual:
  - $ git config — help
- To what's going on with your repository:
  - $ git status
- Note: for this presentation, things between <brackets> are not to be typed verbatim. They represent some value you need to enter.

Setting up Git

- To install Git: $ pip install git
- Set your name and username:
  - $ git config —global user.name “Brittany Howard”
  - $ git config —global “brittanyhoward@uvic.ca"
- To check your settings:
  - $ git config —list

GitHub

- Online host for your repositories.
- Unlimited public and private repositories.
- In the tech industry, often serves as a portfolio.
- Tools for collaboration: issues, milestones, etc.
- Make an account here.

Setting up a repository

- First, we'll start an empty repository on our local machines.
- Make a new repository on your Desktop:
  - $ cd ~/Desktop/
  - $ mkdir practice
  - $ cd practice
- Add a text file, foo.txt
  - $ echo 'testing123' >> foo.txt
- Initialize it as a repository
  - $ git init

Setting up a repositor

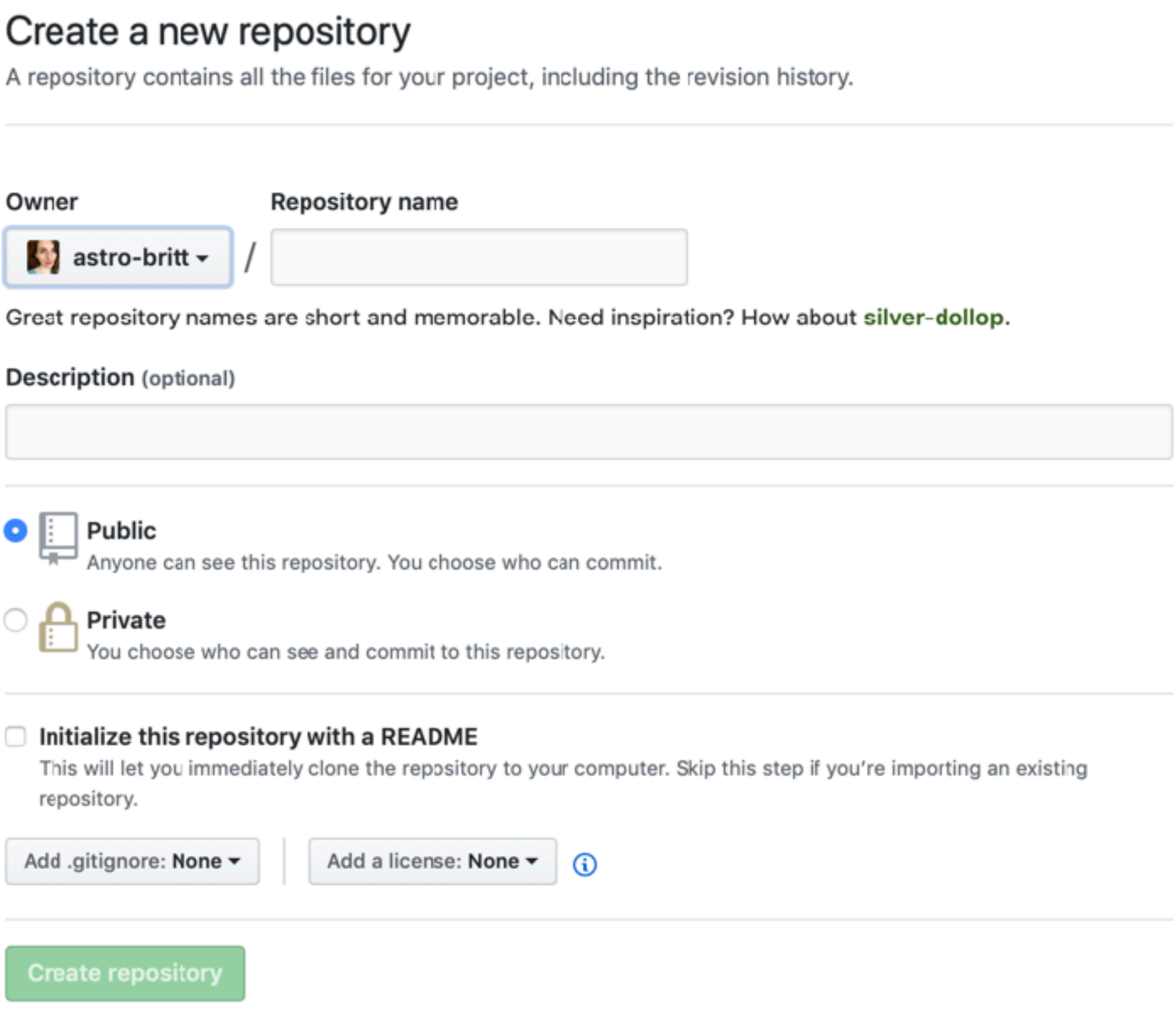
- First, we'll start an empty repository from scratch.
- Make a new repository on your Desktop:
  - $ cd ~/Desktop/
  - $ mkdir practice
  - Initialize it as a repository

```
brittanyhoward @ ~
 [1] → cd Desktop/

brittanyhoward @ ~/Desktop
 [2] → mkdir practice

brittanyhoward @ ~/Desktop
 [3] → cd practice

brittanyhoward @ ~/Desktop/practice
 [4] → git init
Initialized empty Git repository in /Users/brittanyhoward/Desktop/practice/.git/

brittanyhoward @ ~/Desktop/practice
 [5] → ls -a
total 0
drwxr-xr-x   3 brittanyhoward  staff    96 Jan 15 10:58 .
drwx------+ 41 brittanyhoward  staff  1312 Jan 15 10:58 ..
drwxr-xr-x  10 brittanyhoward  staff   320 Jan 15 10:58 .git
```

- A hidden directory .git/ is created to keep track of changes.

- Do not remove this directory.

Connecting this repository to GitHub



- github.com > + > new repository
- Don't initialize with a README. We'll add one later.
- Must use the same repository name.

astro-britt / practice

Unwatch 1 | Star 0 | Fork 0

Code | Issues 0 | Pull requests 0 | Projects 0 | Wiki | Insights | Settings

Quick setup — if you've done this kind of thing before

Set up in Desktop or HTTPS SSH https://github.com/astro-britt/practice.git

Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.

...or create a new repository on the command line

```
echo "# practice" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/astro-britt/practice.git
git push -u origin master
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/astro-britt/practice.git
git push -u origin master
```

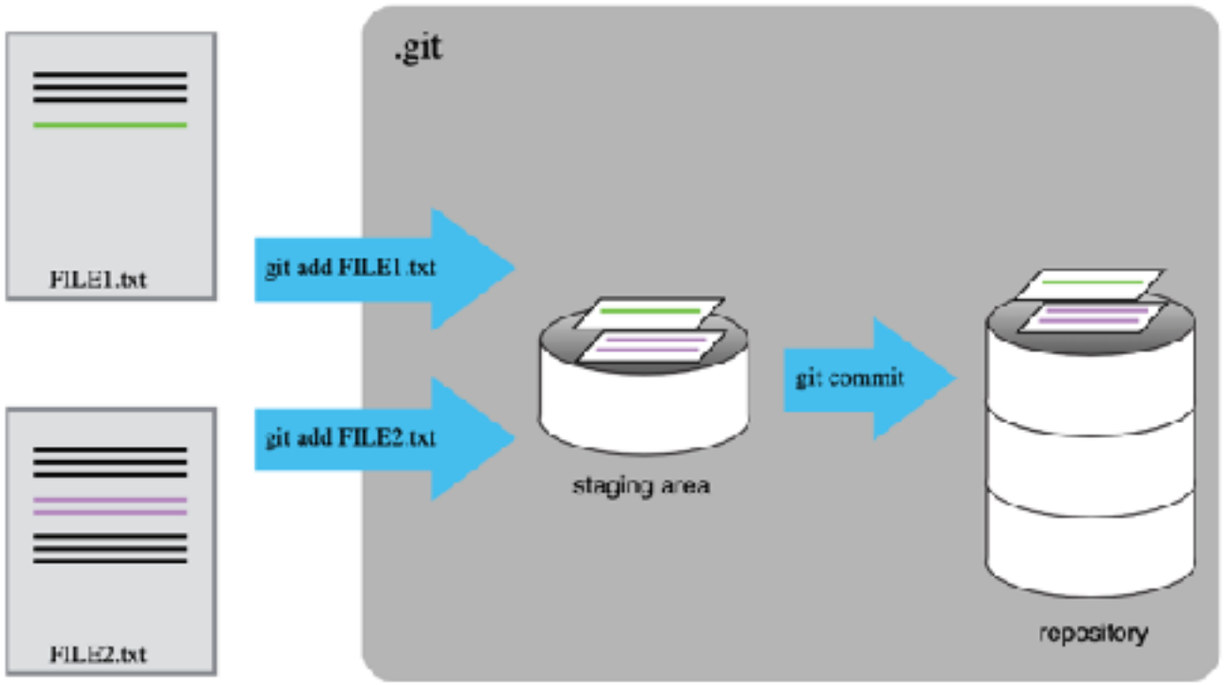
...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

Import code

Tracking changes

- Make a change to foo.txt.
- $ git add <files>
  - Now git tracks these files for changes.
- $ git commit -m '<message>'
  - Take a snapshot of the files you're tracking.
- $ git push
  - Save the snapshot to your remote repository (GitHub).



Connecting this repository to GitHub

- Commit all the files from your local repository:
  - $ git add *
  - $ git commit -m ‘<message>’
- Link your repository on GitHub:
  - $ git remote add origin <remote repository URL>
- Push your local changes to GitHub:
  - $ git push -u origin master
- After this initial push, we can just do:
  - $ git push
- Note: you will need to enter your GitHub password. To avoid this in the future, you can set up an SSH key.

.gitignore

- A list of files not to track, even if you ask git to track everything ($ git add *).
- Specific to each programming language.
- GitHub has many samples to choose from- much easier than making one from scratch.
- Suggestion: make a separate data directory and include it in your .gitignore file.

Adding a README

- GitHub automatically displays markdown README files on the main page of your repository.
- Great for project descriptions, dependencies, short examples, quickstart guides, etc.
- Here is a handy markdown cheatsheet.
- Let's add one now:
  - $ echo '# practice repository' >> README.md
- Add, commit, and push.

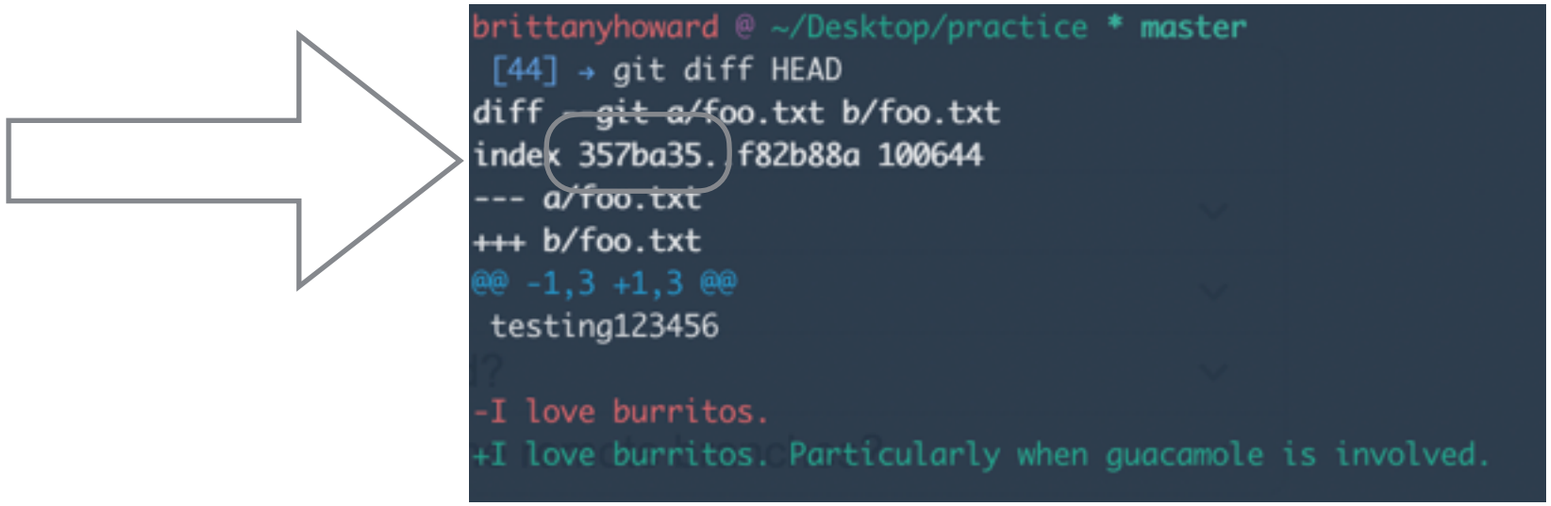
Explore the repository's history

- To see previous commits:
  - $ git log
- Or we can look on GitHub.

```
brittanyhoward @ ~/Desktop/practice * master
 [39] → git log
commit 94a3fae91091bf6bf6fc8a85f7fdbf0db467cc24 (HEAD -> master, origin/master)
Author: Brittany Howard <howardba@umich.edu>
Date:   Wed Jan 23 12:20:48 2019 -0800

    mmm burritos

commit 860e4f2ee56c9764c37fd33958bcadb6c1bb697d
Author: Brittany Howard <howardba@umich.edu>
Date:   Wed Jan 23 11:02:26 2019 -0800

    added gitignore

commit 79717790b774017e55fe0a1854d11c55fa2a5521
Author: Brittany Howard <howardba@umich.edu>
Date:   Wed Jan 23 10:56:36 2019 -0800

    added README

commit 8c7e4ab534fb6763737aed760c2576cd5ef91418
Author: Brittany Howard <howardba@umich.edu>
Date:   Wed Jan 23 10:51:59 2019 -0800

    first commit
```

git diff

- To see the changes that will be made when we commit, do either:
  - $ git diff HEAD
  - $ git diff HEAD <filename>
- HEAD indicates our most recent commit.
- HEAD ~n indicates our n^{th} most recent commit.
- We can also refer to a commit by its index.



Rewind -> checkout

- Make one more change to foo.txt. Add, commit, push.
- What if we made a mistake and want to revert to a previous version?
- We must choose a commit we want to rewind to ($ git log is helpful), then checkout that commit:
  - $ git checkout HEAD~<n> <filename>
  - $ git checkout <commit index> <filename>

Notes on checkout

- Make sure to include a filename, or you'll end up in detached HEAD mode -> for looking but not touching.
- If you want to switch back to HEAD again (like a 'redo' button), do:
  - $ git checkout — <filename>
- Checkout is good for your local repository. But if you pushed to a shared repository, do one of:
  - $ git revert HEAD~<n>
  - $ git revert <commit ID>
  - This makes a new commit that undoes your old one.
- Make sure to commit any changes you've added before trying to checkout.

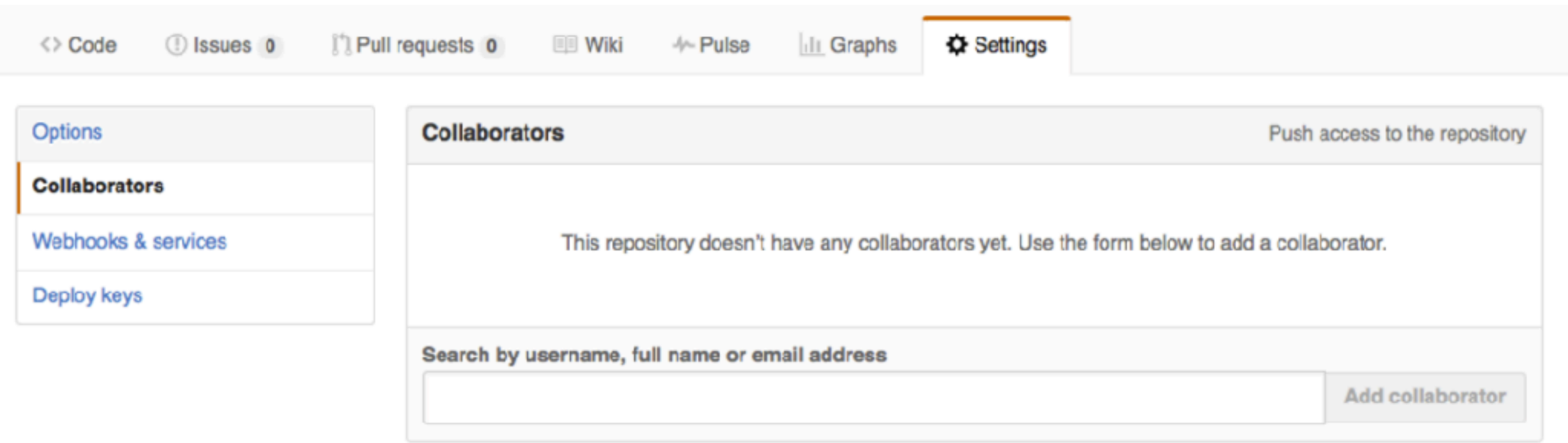
That’s all great for when I’m working alone. But what if I want to collaborate with a team?

The basic commands for collaborating with git

- $ git push
  - Copies commits from a local repository (your machine) to a remote repository (GitHub).
- $ git push
  - Copies commits from a remote repository (GitHub) to a local repository (your machine).
- Note: it is good practice to pull before making any new changes. That way you will be up-to-date and minimize conflicts when you try to push your changes.

Owners and collaborators

- Every repository has an owner. They can have as many collaborators as they want.
- Owners need to approve collaborators.
  - Your repository page > settings > collaborators > enter collaborators' usernames

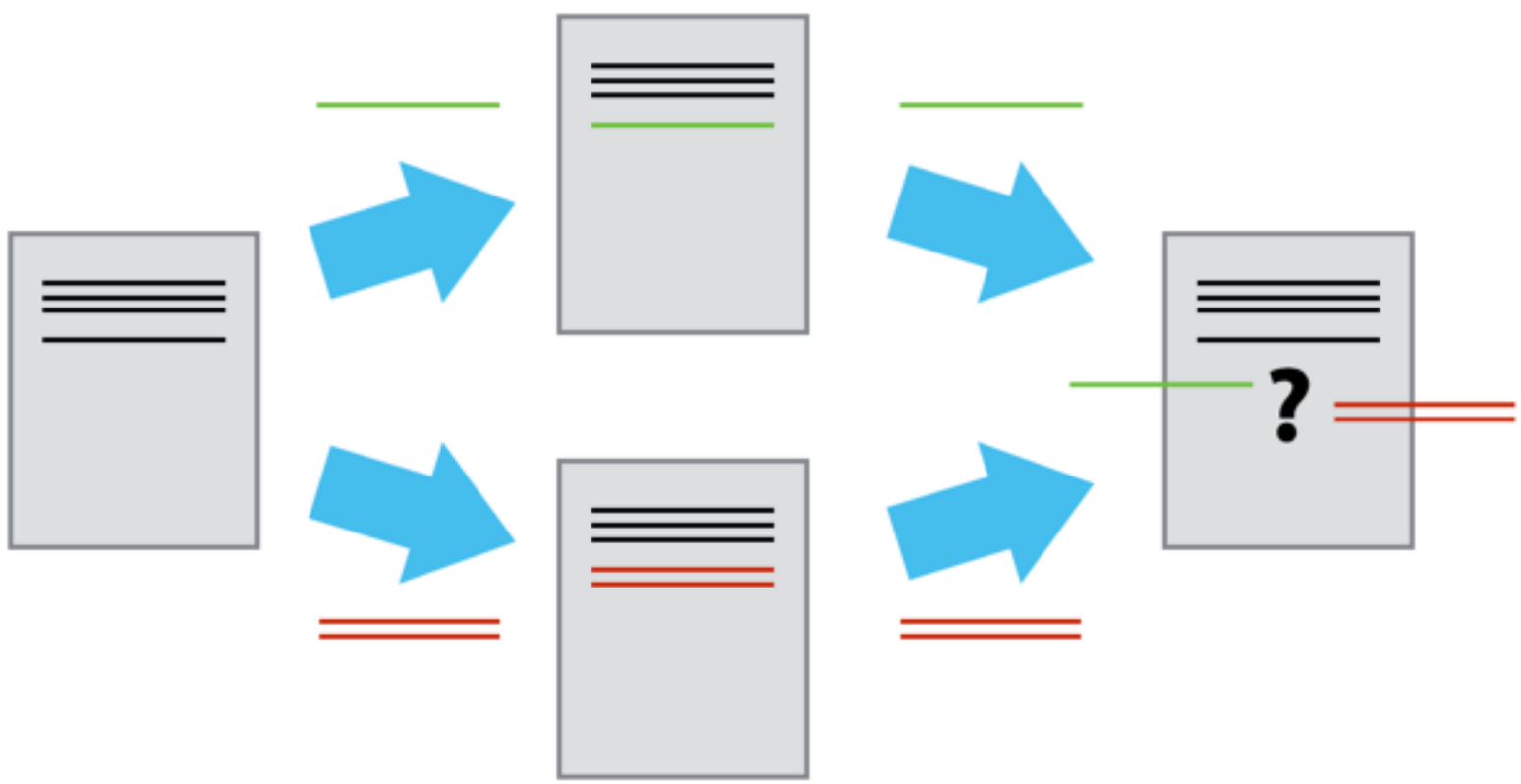


Cloning a repository

- Copies a remote repository to create a local repository with a remote called origin automatically set up.
- Let's clone my repository now:
  - $ git clone <repository url> <path to your local copy>
  - $ git clone <put in my real url> ~Desktop/
- Now you can make changes and commit and push them to my repository.
  - In hi.txt, replace mine with yours. Add, commit, push.

Conflicts

- If multiple people are making changes simultaneously, there are likely to be conflicts.
- Reduce conflicts by pulling often.



Resolving conflicts

- Someone has changed hi.txt, and so have I. We've changed the same line, and git can't resolve this conflict on its own.
- I can commit locally, but I can't push. I need to resolve the conflict first.
  - I need to pull the changes from GitHub, merge them with the copy I'm currently working on, and then push that copy.

Resolving conflicts

- See which file(s) are in conflict:
  - $ git status
- Open the conflicting file(s) with your favorite text editor.
- Conflicts will be marked like this:

```
If you have questions, please
<<<<<<< HEAD
open an issue
=======
ask your question in IRC.
>>>>>>> branch-a
```

- Delete the conflict markers.
- Make the changes you want.
- Add, commit, push.

Branches

- What is a branch? An environment where you can develop new features and then integrate them into your main project when they’re ready.
- Changes you make on a branch don’t affect your master branch.
- Best practice: everything on the master branch is deployable. All new features and fixes are done on separate branches.

Let’s make a branch

- Create the new branch and make it your working branch:
  - $ git checkout -b <name of your new branch>
- Push the branch to GitHub
  - $ git push -u origin <name of your new branch>
- To see all the branches we’ve created:
  - $ git branch
  - Or look on GitHub > Insights > Network
- Note: before switching branches, you must commit any changes you’ve made on your current branch.

Merging branches

- Make some changes in this branch. Add, commit, and push them.
- Now we want to merge this branch back into master:
  - $ git checkout master
  - $ git merge <name of your new branch>
  - $ git push
- There may be conflicts; resolve them as before.

Things to NOT push to GitHub

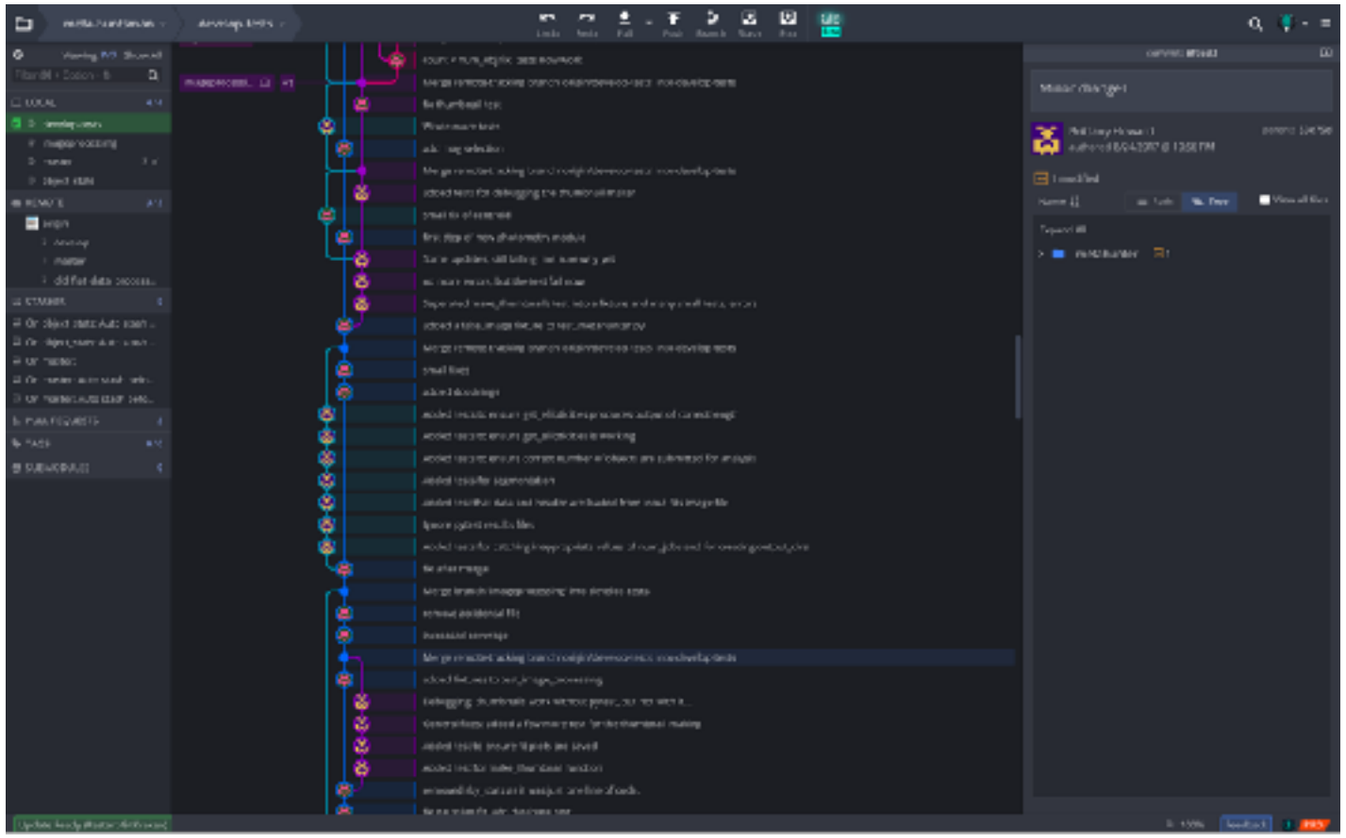
- Passwords
- The private part of your SSH keys
- AWS access keys
- API keys
- Credit card numbers
- PINs
- Etc.

Extra tools on GitHub

- Issues: assign issues to collaborators, comment on the status of these issues.
- Gists: a simple way to store and/or share code snippets.
- Projects: organize tasks, visualize your workflow, manage your team.
- GitHub Pages: generates pretty websites from your README file. Great for documentation.

GUIs for Git

- GitHub Desktop
- GitKraken
- Some IDEs have Git built into them.



BONUS SLIDES

Notes on remotes

- A remote is a copy of the repository that is hosted somewhere else that we can push to and pull from.

- There's no reason that you have to work with only one.

- Remember that the name you give to a remote only exists locally. It's an alias that you choose - whether 'origin', or 'upstream', or 'fred' - and not something intrinsic to the remote repository.

Cloning vs forking

- It's a subtle difference, and you should choose based on whether you need future updates to the code and whether you plan to contribute.
- Here is a helpful guide.