The Terminal, Git and GitHub

FLATIRON SCHOOL

First:
THE TERMINAL

LEARNING OBJECTIVES

// Utilize bash commands through a terminal interface

// Use the terminal to list, make, move and remove files and directories

// Use the terminal to navigate between files/directories and open Jupyter notebooks

Basic Commands

`$ pwd`

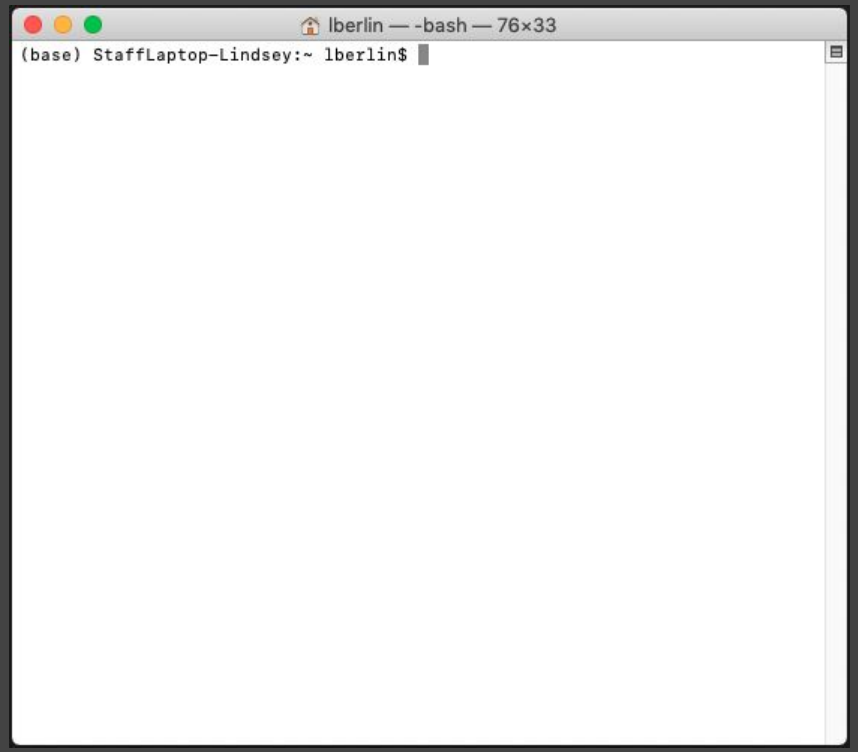
display the current working directory of the shell

`$ ls`

list the files and directories of the current directory

`$ cd`

change the directory to update the current working directory



Basic Commands

`$ touch`

create a new file

`$ mkdir`

create a new directory

`$ mv`

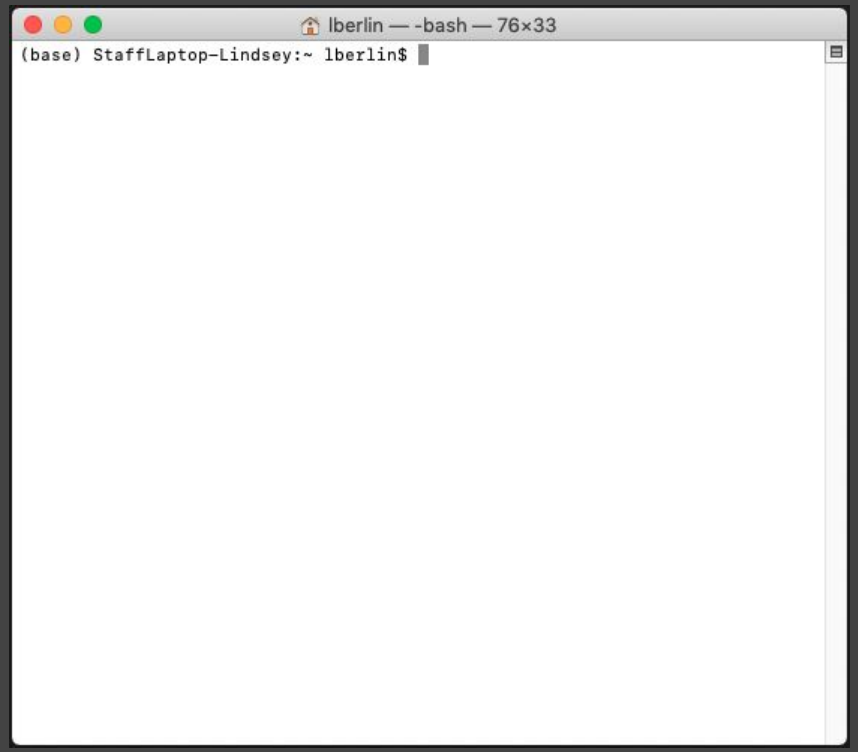
move a file from source to destination

`$ rm`

remove a file from the file system

`$ rmdir`

remove a directory from the file system



Special Directories

/

root, the top-level directory

~

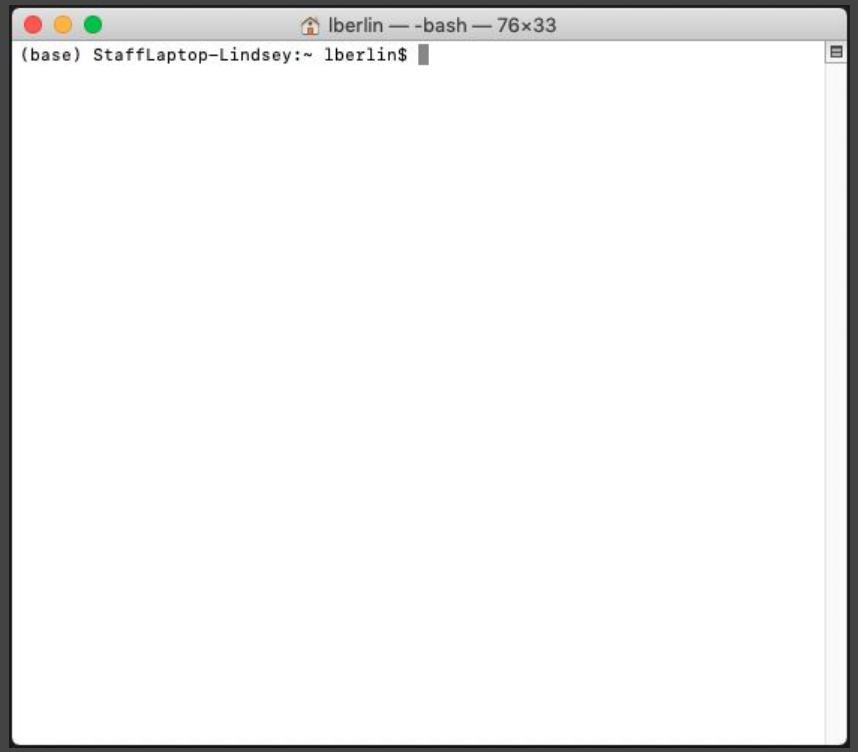
your home directory

.

the current directory

..

the parent directory



Additional Resources

Initial Learning Resources:

- OpenClassrooms' course on the command line
- MIT's Terminus command line game

Going Further:

- Unix Primer tutorial: Basic Commands in the Unix Shell
- Data Camp tutorial: 8 Useful Shell Commands for Data Science

And now,
GIT AND GITHUB

LEARNING OBJECTIVES

// **Fork** a repository on GitHub

// **Clone** a repository from GitHub

// Checkout **branches** to contain your work

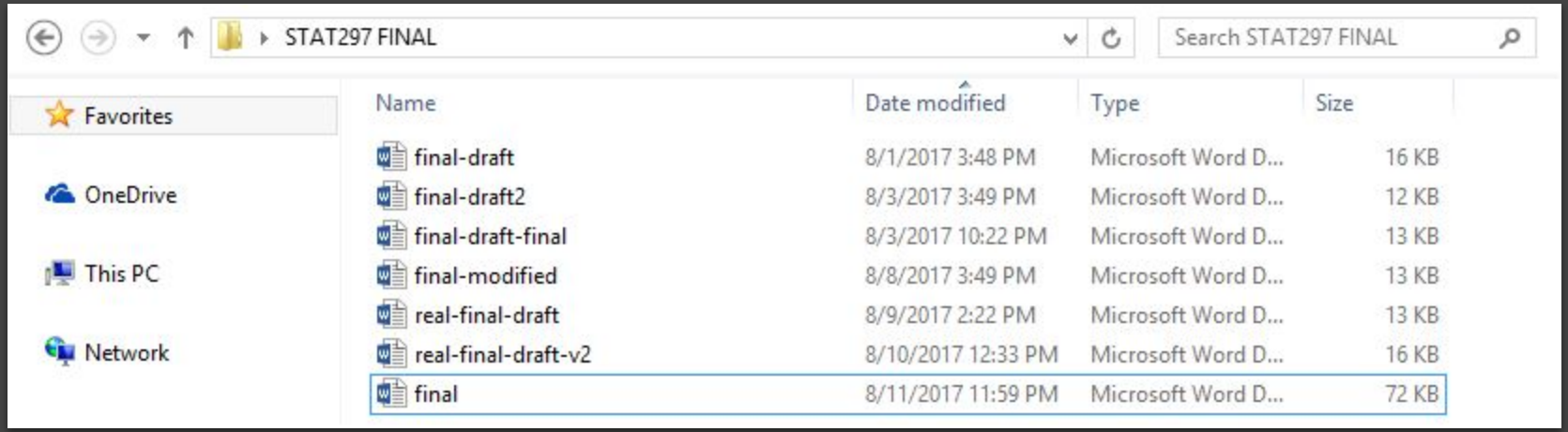
// **Add** updates to track for a commit

// **Commit** changes (with meaningful messages)

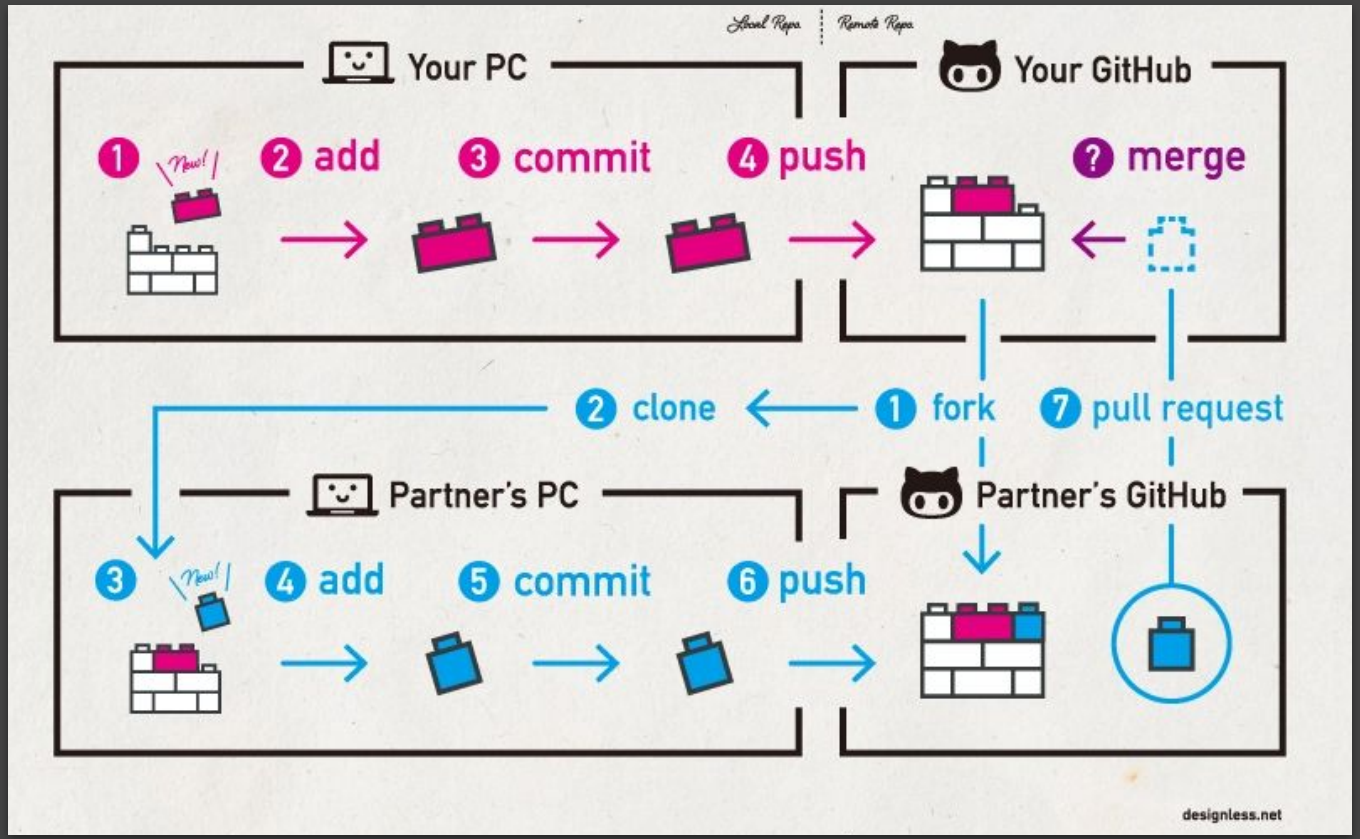
// **Push** local changes to a remote repository

// **Pull** remote changes to your local repository

How many of you have seen a folder like this?



Why bother?


Local Repo
Remote Repo
Your PC
Your GitHub
1 New!
2 add
3 commit
4 push
? merge
2 clone
1 fork
7 pull request
Partner's PC
Partner's GitHub
3 New!
4 add
5 commit
6 push
designless.net

Important Distinction!

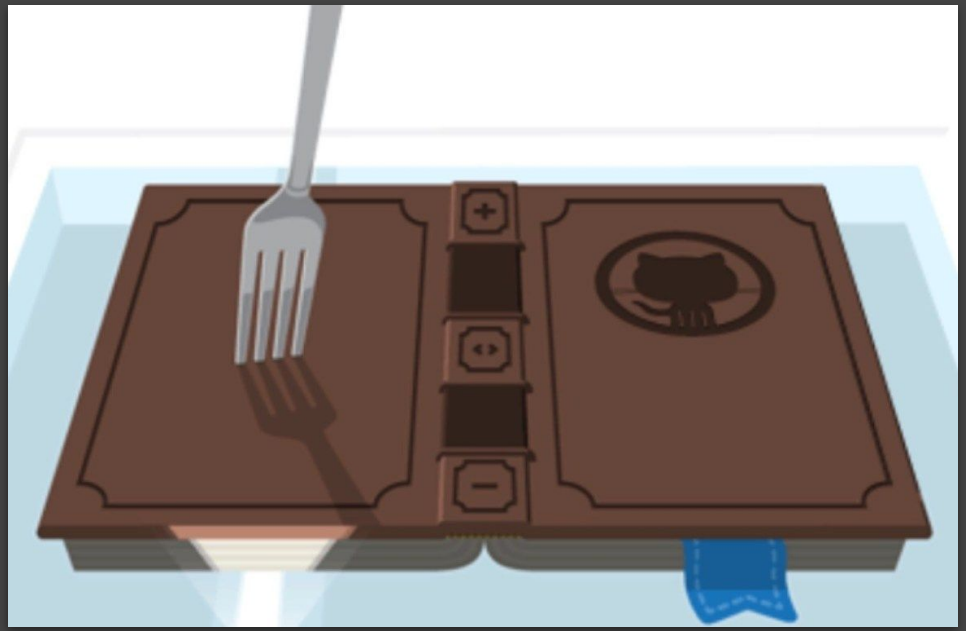
Let’s Try This Out

https://github.com/learn-co-curriculum/dsc-git-practice

Fork a repository

Forking creates your own personal copy over on your own personal GitHub.

- On a fork, you can freely experiment with changes without affecting the original repository you copied from.
- This is the best way to use someone else's repository as a starting point for your own projects!
- If you like, you can later submit those changes to the original repository in order to collaborate.

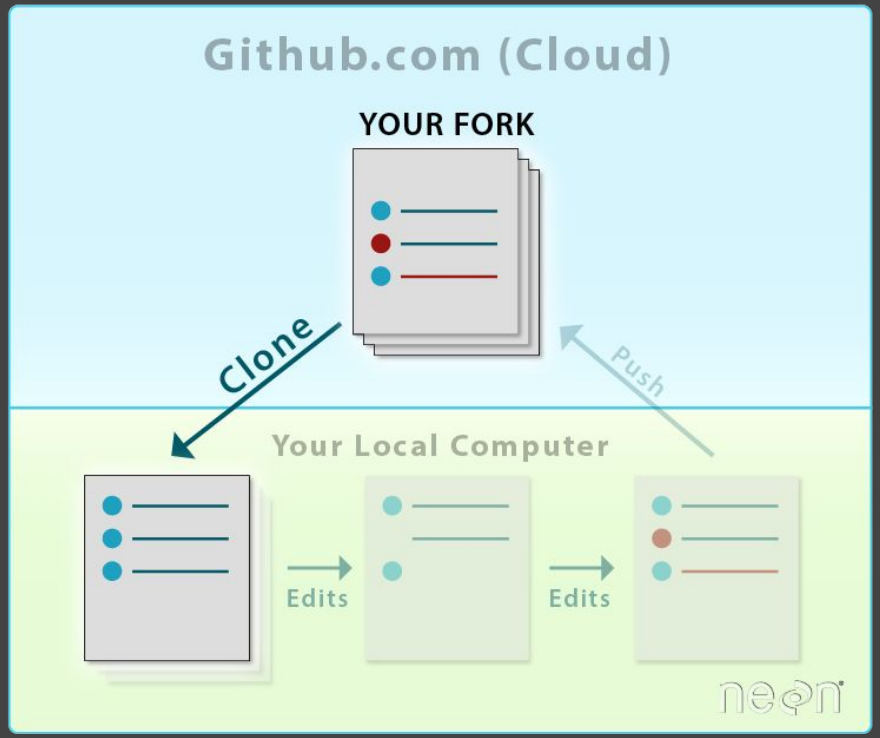


Clone a repository

Cloning makes a copy of an existing online repository on your local machine.

```
git clone [URL]
```

The difference between forking and cloning is that cloning moves from cloud to local, as opposed to forking which moves code from someone else's remote repository to your own remote repository - all in the cloud.



OR Initialize a Repository

Initializing turns a local directory into a local git repository.

```
git init
```

Here, you're creating a repository using ONLY git, without involving GitHub at all. You're working only on your local machine, not at all in the cloud.

Checkout Branches

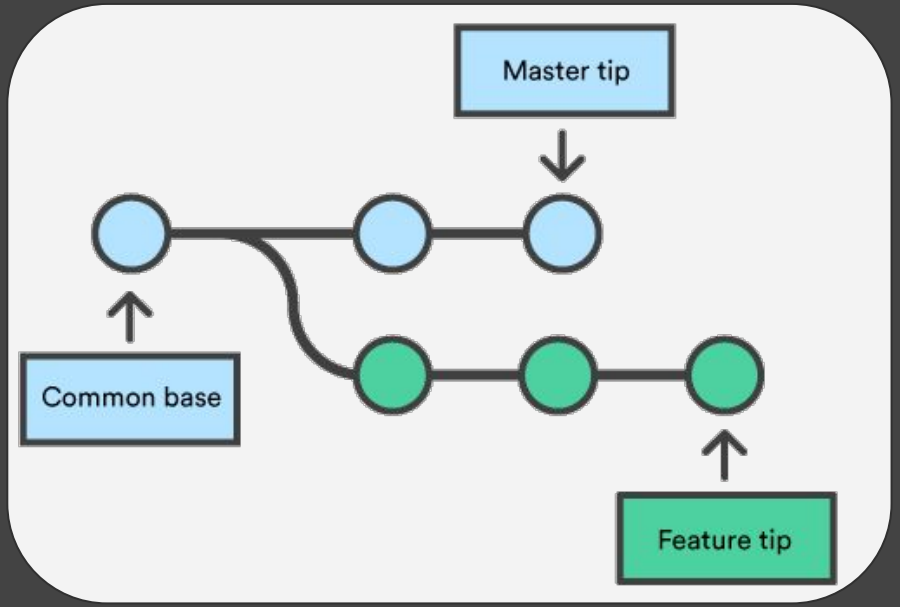
Branches allow you to do your own work based off a main ‘trunk’ of code, without disrupting other people working off that ‘trunk’.

To use a branch, you must both create the branch and **checkout** that branch.

```
git checkout -b [NEW-BRANCH-NAME]
```

This command both creates a new branch and checks out that branch, so any new work will be in that branch of code (two steps in one!).

Check your branches with `git branch`



Make Local Changes

This isn't a git command - just make local changes to the `README.md` file, or add a new file to the repository so we can see what it looks like to keep track of new changes.

Check Your Status

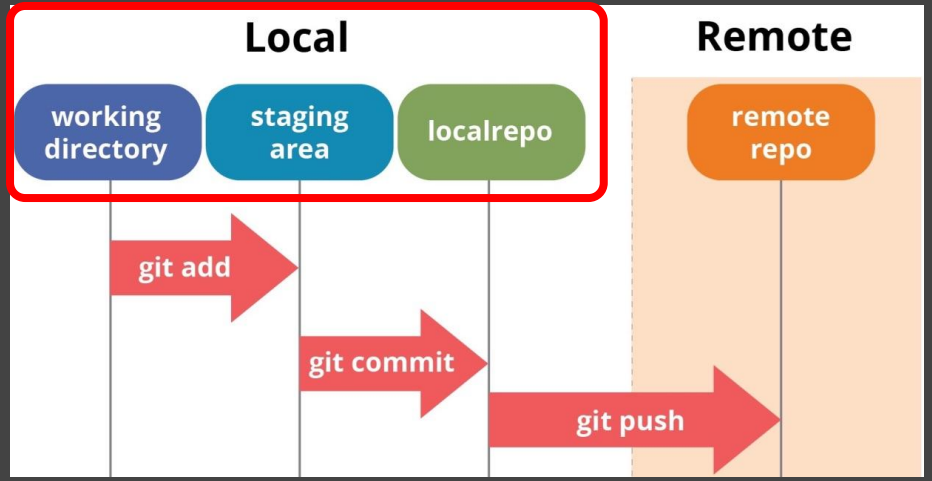
You made some changes, but what does that look like from Git's point of view? In other words, where are we in the process, on our local drive?

```
git status
```

Git differentiates between staged and unstaged files (as well as tracked and untracked files).

- Red represents unstaged files.
- Green represents staged files

what's our status?

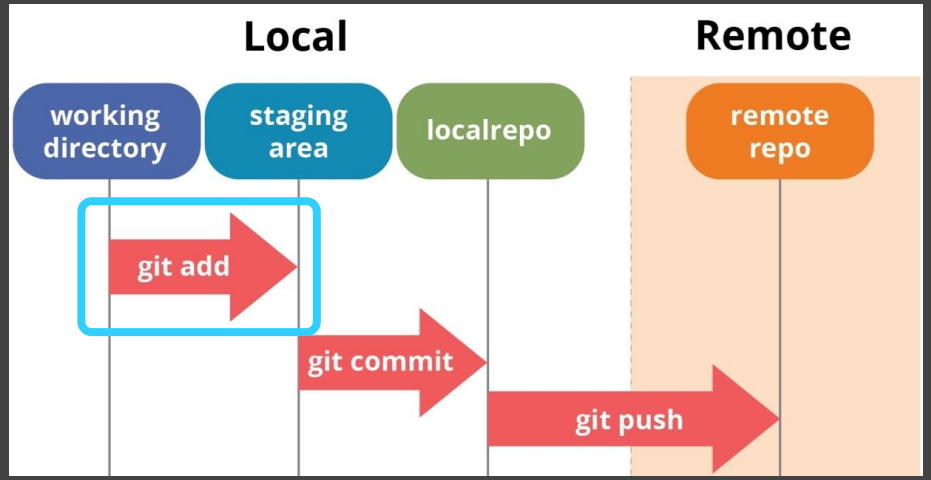


There are more steps in Git than you think you need...

Add Your Changes

Adding files tells Git which changes you'd like to stage, to eventually be committed to your local repository.

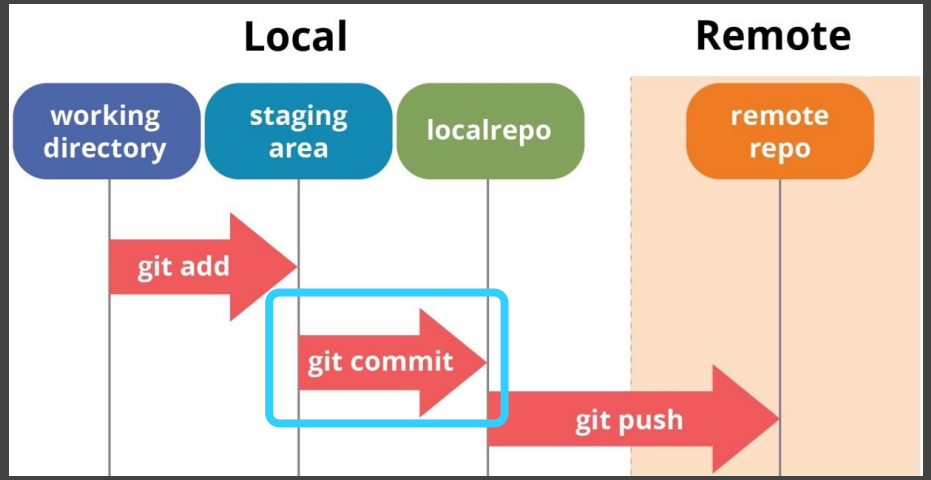
```
git add [FILE]
```



Commit Your Changes - With Meaningful Notes!

Committing tells Git that these changes shouldn't just be tracked, they're actually ready to be a part of your local repository.

```
git commit -m "[MEANINGFUL MESSAGE]"
```



Commit messages should be written in the present tense, and should finish the sentence: "If I apply these changes, they should…"

- "Add spell check feature"
- "Fix super awful bug written by Bryan"
- "Update gifs in README.md"

Don’t let this be you! Informative messages only!

Push Your Changes

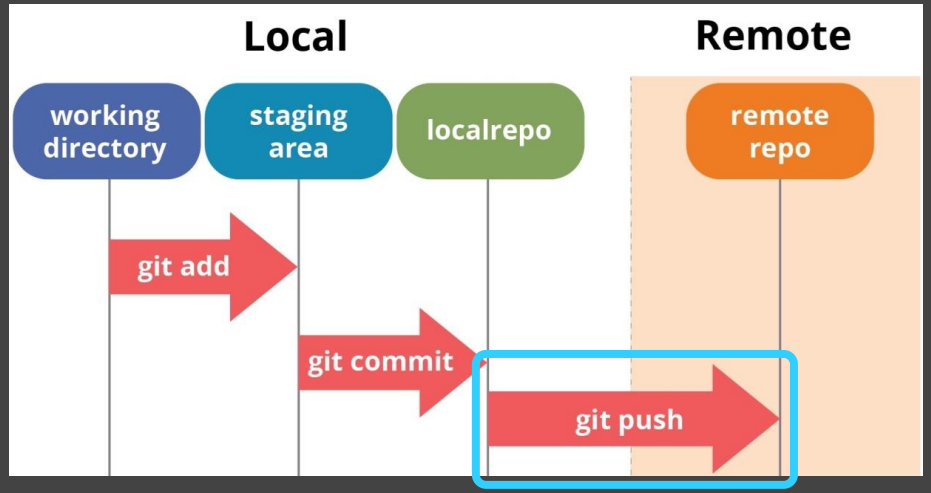
Pushing uploads your local repository to a remote repository (say, your GitHub)

`git push [REMOTE LOCATION] [BRANCH]`

(ex: `git push origin main`)

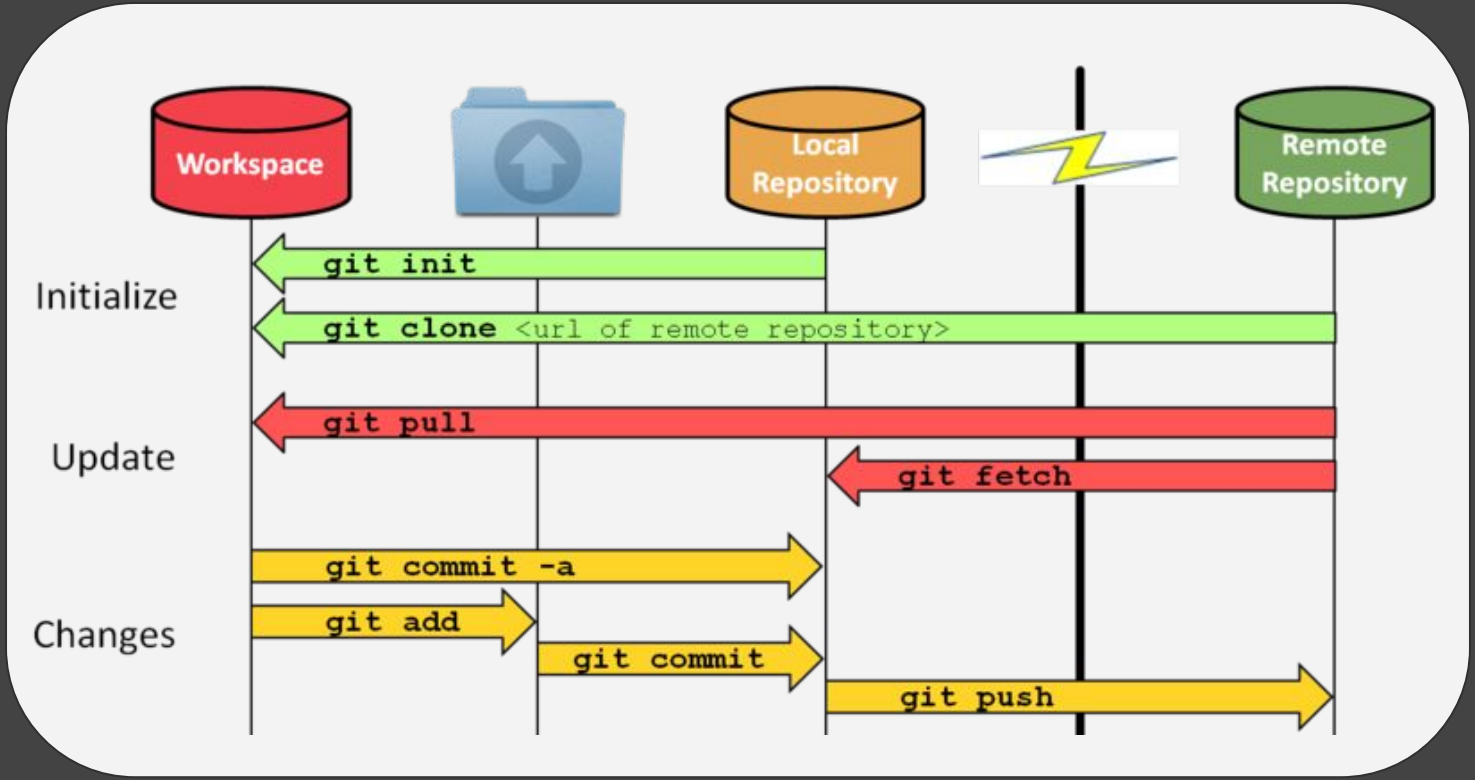
This is when you connect your local work to your remote repository - it also acts as you backing up your work to the cloud!

If you created a new branch, you should use your branch name to push - not the default!



Just a few more things, we’re almost there...

A Bigger Picture

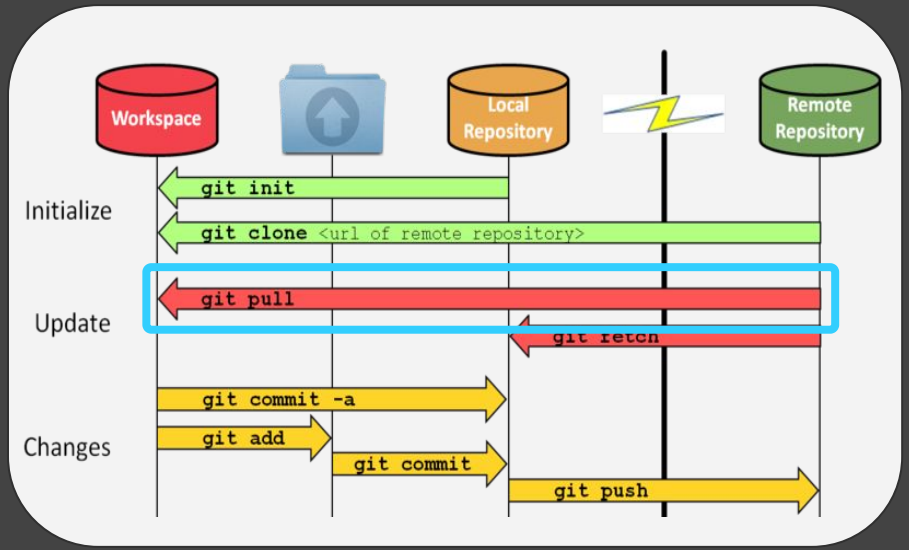


Pull Other Changes

Pulling brings remote changes down to your local workspace.

```
git pull [REMOTE LOCATION] [BRANCH]
```

If there are changes in the remote repository to which you're trying to push changes, you'll need to pull the changes down to your local machine before pushing your work back up.



Avoiding the Merge Headache

Merging allows you to bring changes together into one harmonious project . . .

HOW TO AVOID MERGE CONFLICTS:

- Plan ahead and communicate
- Work on different Jupyter notebooks
- Use your own branch

Additional Resources

- This blog post is a nicely laid out walkthrough of git
- Git has a whole open-source book on how to use git
- Atlassian has their own version of GitHub, but their tutorial on Git is solid
- Stumbled into a merge conflict? Learn how to resolve it in this blog post