“HOW DO I DO XYZ WITH GIT?!”

GITHUB + GIT 101

FIRST THING'S FIRST…GIT != GITHUB

GitHub is a website. Git is not.

Git is a revision control system + tool to manage your source code history.

GitHub is responsible for hosting Git repos.

There is no GitHub without Git but there is Git without GitHub!

BECAUSE GITHUB IS NOT GIT…

You need to link your GitHub to Git/your machine by generating a new SSH key and adding it to the SSH agent, if you've never used Git on your machine before.

But you've already done this ahead of time ;)

https://help.github.com/articles/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent/

FORKING A REPO VIA TERMINAL

Creating a fork is just lingo for copying a repository!

By forking a repo, you can experiment/fiddle/mess up big time/ make awesome changes all without messing up the original copy of the repository.

1) Go to https://github.com/savannahostrowski/learning-git

2) Click the Fork button in the top-right of the screen.

At this point, you have a fork of my repo in your GitHub…but unfortunately, GitHub != Git so you don't yet have your fork on your machine.

CLONING A REPO/FORK

To get your copy of the repo (fork) on your machine, you're going to need to clone it.

1) Go find your fork of my repo on your GitHub.

2) Click **Clone or download**

3) Click the clipboard with an arrow to copy the link.

CLONING THE REPO VIA TERMINAL

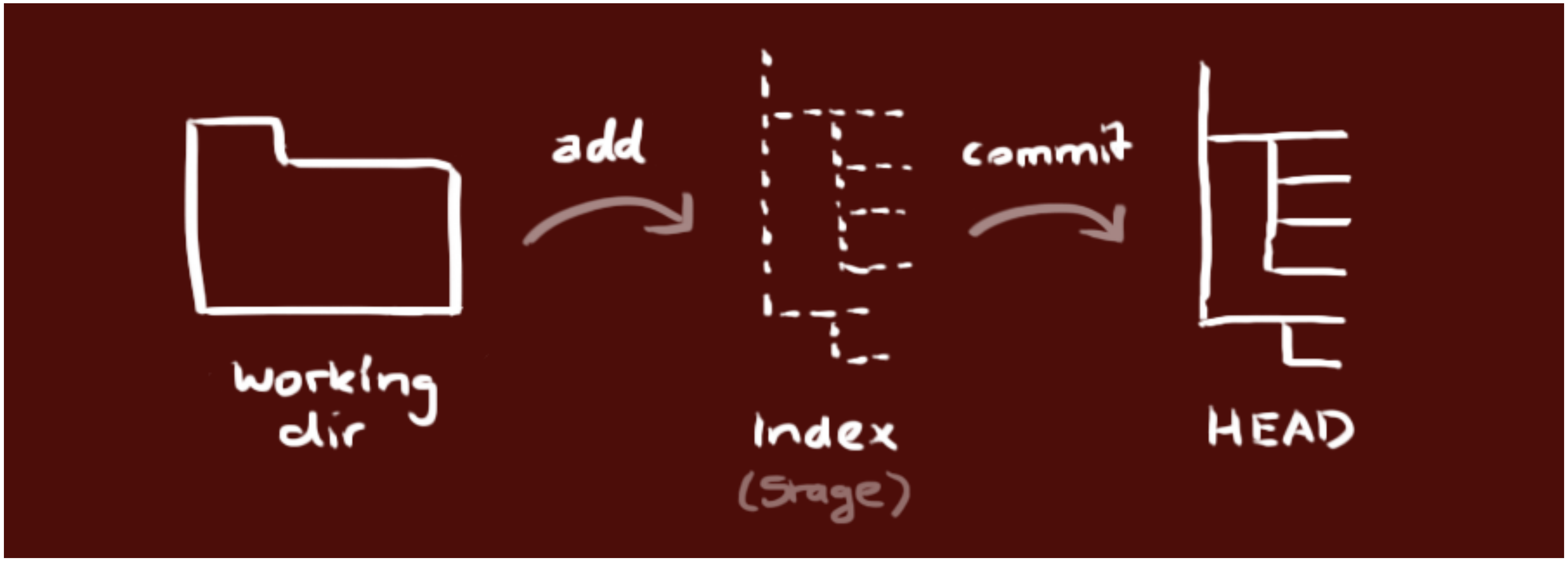
Alright, you're ready to clone your fork and get a copy on your machine.

1) Navigate to the directory you want to place the forked repo in.
    1) Open terminal + **cd** to the directory
2) Type **git clone git@github.com:<YOUR_USERNAME_HERE>/learning-git.git** into the terminal
3) Enter password, if prompted.
4) Type **ls** into the terminal to see the list of folders/directories in that directory. You should now see the **learning-git** repo!

WHAT'S IN YOUR LOCAL REPOSITORY?

In your local repository, there are three trees:

1) Working Directory - Holds actual files
2) Index - Acts as a staging area for changed files
3) HEAD - Points to the last commit you made



GIT BRANCHES

Cool! You're ready to make some changes/add some features!

Let's create a branch.

1) **git checkout -b <branch_name>**
    1) **checkout** to checkout the branch
    2) **-b** to create a new branch
2) You are now on your new branch. Double-check your branch by typing **git branch** (your current branch will be the one with a asterisk next to it in a different colour.
3) If you want to switch branches, you can type **git checkout <branch_name>** without the **-b**
4) If you want to delete a branch, you can type **git branch -D <branch_name>**
    1) **-D** is for delete ;)

GETTING NEW REPO BRANCHES

You can also get also get new branches from upstream if you **git fetch**

MAKING CHANGES + SEEING WORKING TREE STATUS

1) Open the README or create a new file using your favorite text editor, and save your changes if needed.

2) Type **git status** in the terminal to see the status of the working tree.

    1) You will see unstaged, changed files in red.

MOVING FILES FROM THE WORKING TREE TO INDEX

You can now propose your changes by adding it to the Index by using **git add <filename>** (either a single file, a directory, or multiple files) or if you want to add all of them then use **git add .**

COMMIT FILE CHANGES TO THE HEAD

To commit your file changes, use **git commit -m '<commit_message>'** .

You can also combine this step and the previous by using **git commit -a -m '<commit_message>'**

Best practices for commit messages

1) Use imperative mood - "Add tag-based filtering to DAG runs"

2) Less than 72 characters

    1) This number seems weird but the reason it's 72 characters is because **git log** adds a padding of 4 blank spaces when displaying the commit message. To keep it perfectly center and make it read well on an 80-column terminal we want to keep 4 more blank spaces on the right side (fun fact!)

3) Explain what and why vs. how (we don't need to know implementation details here!)

Remember, at this point once you've committed your changes, the files you've commit are committed to HEAD but not yet in your remote repository on GitHub. Because again, Git is NOT GitHub.

VERIFYING YOUR COMMIT + SEEING LATEST COMMITS

Alright, you're now ready to push up your changes to the local repo. Before doing this, you can double-check that you have committed the appropriate files.

Using **git status,** you can verify that all the files you commit have left staging and have made their way to HEAD from Index. If you've done this properly, your changed files will have disappeared from Working Directory and Staging.

To verify that you've actually committed to the HEAD, you can type **git log.** This is a record of all changes made to your current branch. It will include all changes made on your base branch (dev, master etc.) and all changes you've made on your branch! Check to see that your commit is the first one in the log. You can navigate this log with the up and down arrow keys. When done, type **q** to quit.

PUSHING YOUR CHANGES TO THE REMOTE REPOSITORY

This is where magic happens, my friends!

Type **git push origin <branch_name>**.

You are now ready to open a pull request!

OPENING A PULL REQUEST

1) Go to your fork of the repo on GitHub

2) Click on the Pull Requests tab or click on the yellow bar with your branch name on the main page.

3) Click the green **New Pull Request** button.

4) Click **Compare across forks**.

5) Set the base fork to **upstream/repo_name** and **base branch** to the **master/dev/whatever branch**.

6) Set the head fork to your **fork/repo_name** and the **compare** branch to **your branch name**.

At this point, you should see your commit message(s) appear followed by changes files (green = added, red = deleted). Click **Create Pull Request** and then fill in the title and PR description on the next screen. Then, click **Create Pull Request** again!

RECOMMENDATIONS

Although we've only used a couple Git commands here, it's super easy for things to get scary super quickly.

Here are best practices:

1) Keep your master branch (dev, master, etc.) clean. Don't commit directly to that branch.

2) Create branches for each feature, bug, task, etc.

3) Make sure that each new branch is created from a fresh copy of your master branch.

4) If you're working on some feature for a long time, remember to **git rebase** or **git pull** before pushing up your changes.

    1) **git rebase** will pop off your changes, pull all new changes from upstream onto the branch, and then put your changes back on (typically resulting in less merge conflicts)

    2) **git pull** will just try and smash upstream changes directly onto your branch

5) Make commits incremental and frequently (there's nothing worse than losing your code + you can always roll back to older versions of your code).

WHAT IF I MADE A MISTAKE?!

99.9% of mistakes made in Git are resolvable!

You can amend commit messages, undo commits, remove accidentally staged files, squash commits, resolve merge conflicts etc.

StackOverflow is the real MVP here.

WHAT IF I ACCIDENTALLY COMMITTED TO MY MASTER BRANCH?!

git reset –soft HEAD~1

- **git reset** is all about moving HEAD
- **–soft** moves HEAD (and only HEAD)
- **HEAD~1** refers to how many commits back you want to move head (could be any number)

IN SUMMARY, OUR GENERAL WORKFLOW LOOKS LIKE THIS

1) **git pull** on your master to make sure you have the latest code on your machine

2) **git checkout -b <descriptive_branch_name>** to create a new branch

3) **git status** to see changed files

4) **git add <file_1> <file_2>…** OR **git add .** OR **git commit -a -m…** to add changed files to staging for commit

5) **git commit -m '<message_here>'** to commit your changes to your branch

6) **git push <remote_name> <branch_name>** to push your branch up to GitHub

7) Hit up your GitHub + open a PR from your **fork + branch** to **upstream + desired branch.**

HERE ARE SOME AWESOME RESOURCES + EXERCISES:

- https://services.github.com/on-demand/intro-to-github/
- http://rogerdudler.github.io/git-guide/
- http://git-school.github.io/visualizing-git/
- https://services.github.com/on-demand/downloads/es_ES/github-git-cheat-sheet.pdf
- https://www.codecademy.com/learn/learn-git

FIN.