Git Intro

Goals

- Describe why Git is useful
- Describe the workflow for using Git locally
- Define what a VCS is
- Initialize an empty git repository, and explain what the command ***git init*** does
- Remove a git repository

What is Git?

If you google “what is git” you will probably see the definition for “an unpleasant or contemptible person.” Thankfully, Git is much better than that. According to [the Git documentation](https://git-scm.com/ <https://git-scm.com/>):

“Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.”

In plain English, Git is a tool that allows developers to track versions of their code over time.

It does this by creating “snapshots” of the current state of the code base whenever you tell it to.

Put simply, Git is essential when collaborating with other developers to ensure that previous “snapshots” of the code can be revisited if necessary.

This is helpful when revisiting breaking changes or undoing changes.

When you’re learning about tools like Git, you’ll often see the acronym VCS. This stands for **Version Control System**

Git is a VCS because it lets you create different versions of your code and easily swap back and forth between different versions.

While Git is a hugely popular VCS, it’s not the only one. Subversion is another example.

Installing Git

Before we can get started with anything Git related, we need to make sure you have Git installed.

In your terminal, type in `git --version`; if you do not see an error, you are good to go.

If you are seeing any errors, you may need to install Git on your computer.

If you’re on Linux, try running ***sudo apt-get install git*** or

If you’re on Windows - https://git-scm.com/download <https://git-scm.com/download>

Getting started with Git

Once you have Git installed, you need to "initialize" a repository with Git before you can start using it

run the command ***git init*** inside of a folder to do this!

This creates a ***.git*** folder which is what allows you to start using git in that folder

Initializing a repository in the wrong place

If you accidentally initialize a repository in the wrong directory, you can just remove the ***.git*** folder using ***rm -rf .git***.

You **don't** want to make your Desktop or Home folder a git repository!

Local workflow

- Three areas
  - Working Directory
  - Staging Area (added, but not committed)
  - Repository (added and committed)

Git status

- In order to see where our files are in the local workflow we run the command ***git status***
- Untracked files will be in red text
- Tracked but uncommited files will be in green text
- After commiting our working directory will be clean and no longer show the commited files

Working Directory

- The working directory holds files that we are working on but have not yet saved to git
- As files are edited, git sees that they have been modified but their changes are not recorded
- Lets imagine we just created a new repository
- If we run the command ***git status***, it would look something like this

```
HEAD git_playground $ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        app.js
        index.html
        main.css

nothing added to commit but untracked files present (use "git add" to track)
HEAD git_playground $
```

<_images/untracked_files.jpg>

- All files that have been modified are shown in red text
- These files are untracked until we move them to the staging area with the command ***git add***

Git add

- Git add is used to save our modified files so that they will be included in the next commit
- This allows us to choose what modified changes we want to save and which changes we want to ignore
- The command ***git add*** followed by the name of the file moves that file to the staging area
- To move all modified files to the staging area at once we use ***git add .***

This is what ***git status*** would look like after adding 'app.js' and 'index.html'

```
HEAD git_playground $ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   app.js
        new file:   index.html

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        main.css

HEAD git_playground $
```

<_images/tracked_files.jpg>

'main.css' will not be included in the next commit unless it is moved to staging

Staging Area

- If we move a modified file to the staging area, we are preparing to commit the state of the file at that point
- If we continue working on that file we have to add again or the new changes won't be tracked
- The staging area is where we save the state of the work we are preparing to include with the next commit
- Files that are staged are shown in green text

Git commit

- Once we are satisfied with the work of a file in our staging area we commit it to the local repository
- We need include a commit message that sumarizes the changes that were made in that commit
- The message is connected to the commit with the ***-m*** flag
  - ***git commit -m "summary of this commit"***
- A commit is still local until we push it to a remote repository (more on this later)

After making the commit will see a message similar to this

```
[HEAD git_playground $ git commit -m "initial commit"
[master (root-commit) 011c34e] initial commit
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 app.js
 create mode 100644 index.html
master git_playground $
```

<_images/committed_files.jpg>

Committing Workflow

- ***git status*** - What files have been added / modified?
- ***git diff*** - Among those files, what has been changed?
- ***git add <NAME_OF_FILE>*** - Add NAME_OF_FILE to staging area
- ***git commit -m*** - Commit the file with a message!
- Optional: ***git log*** - See a log of all your commits

.gitconfig Settings

If you take a look at ***git log*** you may not see any information for the author and email.

Here's how you change it:

```
git config --global user.name "YOUR NAME"
git config --global user.email "YOUR EMAIL"
```

If you also find it annoying to press ***q*** every time in ***git log***, you can change this as well

```
git config --global --replace-all core.pager cat
```

These global configuration settings live in a file called ***.gitconfig*** which typically lives in your home directory. Try running ***cat ~/.gitconfig*** to see all of your settings!

Recap and next steps

Recap

Why Git?

- We need a nice way to keep track of changes to code
- We want a way to revert back to previous versions of code

Coming up

- We still have a single point of failure
- If our computer dies, we lose everything!
- Let's make sure our code exists in another repository
- One that is not *local*, but *remote*
- That's exactly what GitHub is for!