1. Fork
2. Clone from fork
3. Set up remotes

Fetch upstream
   a. `git fetch upstream`
Checkout upstream/main
   a. `git checkout upstream/main`
   b. If your working tree is dirty (`git status` shows red/green), do you need the files?
      i. Yes: (, then and ( ) files
      ii. `git reset --hard`

Checkout a new branch
   a. `git checkout -b NEW-BRANCH-NAME`
   b. The branch name should be named something based on the work you're doing
Do work (write code, create art files)
Save the work
   a. Add files to staging
      i. `git add FILENAME.EXTENSION`
      ii. Repeat as needed for every file that will be committed
   b. Commit
      i. `git commit -m "COMMIT MESSAGE HERE"`
   c. as needed
Push to origin new-branch
   a. `git push origin NEW-BRANCH-NAME`
   b. as needed
Open/Modify Pull Request against "upstream/main"
   a. as needed
   b. If accepted, merge

Git is a "version control system" that allows you to capture "snapshots" of code at a certain point in time.

The main benefit of using git is it allows for easy (once you get used to it) code sharing, not only between other people, but between multiple computers that you may use, as well! Git is used to back-up and compare different "versions" of the same code base.

.

Let's get started with the words you'll need to know:

A repo is just a regular folder containing all the code (or files/folders), plus a special folder, ".git".

Remotes are the places where your code lives (usually online), like GitHub or BitBucket.

This refers to a change in code (a file) that will be preserved in time. The point where a commit is made can be "jumped" back into at any time. Every commit gets assigned a "hash" - which is just a mathematical way of verifying a unique "state" (combination of all text, along with the time and author). Example of a commit hash: "c41ed97a986448d6fc7522fe008bea123df422c0"

Head is the commit where your "code is at." It's a snapshot of your project's code.

Many new users to git are confused by branching. While a commit is a snapshot of code at a certain time, a branch allows for simultaneous development of the same code base without cross-pollination. Many would ask, "why would I want to have multiple branches?"

Checking out code is "jumping into" a snapshot of code, including commit hashes or branches.

Merging combines multiple branches. It creates a special commit that will resolve "conflicts" between branches, where the same code was edited in both branches.

Pushing is an operation performed against remotes. It syncs your machine => servers.

Pulling is like pushing, but the to/from is reversed. It takes the code from a server and syncs it to your local machine. This will also set your HEAD to the most recent commit on that branch.

Fetching is like pulling, but it doesn't set your HEAD like pulling does.

https://codeberg.org

Install Homebrew and OS X Command Line Tools:

```
$ /usr/bin/ruby -e "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

```
$ brew install git
$ brew install git-lfs
```

```
$ sudo apt update && sudo apt install git git-lfs
```

Follow this guide to install :
https://gitforwindows.org/
You'll need to use Git Bash for the exercises in this guide.

Set your name and email so git "knows" who you are:

```
$ git config --global user.name "YOUR NAME GOES HERE"
$ git config --global user.email YOUREMAIL@GOES.HERE
```

The following only needs to be run one time on your machine:

```
$ git lfs install
```

Start out by making a new directory to store your repositories (let's keep things organized):

```
$ mkdir ~/git
$ cd ~/git
```

Check for a valid output from `ssh-add -L` . If you see an error, follow this section, otherwise, skip it.

(or the computer restarted):

```
$ eval `ssh-agent -s`
$ ssh-keygen -f ~/.ssh/id_rsa -t rsa -N ''
$ ssh-add
$ ssh-add -L # copy the output of this and paste as a new SSH key in
Github => Settings => SSH and GPG Keys (name the key whatever you
like, usually based on the name of a computer)
```

Add the following to the bottom of your ~/.gitconfig:

```
[alias]
      s = status -s
      lg = log --oneline --decorate --all --graph
      d = diff --stat
      tags = for-each-ref --sort=taggerdate --format
'%(refname)%(taggerdate)' refs/tags
```

These are " " that will make common git operations much easier.

Okay, let's all go through this, together. This is all outlined in a summary at the top of this document. Please refer to that after you've gone through this a few times.

1. the "git-workshop" repository:
   a. https://codeberg.org/singlerider/git-workshop-25-jan-202
2. the forked repository. Note how it's no longer under the "singlerider" user:

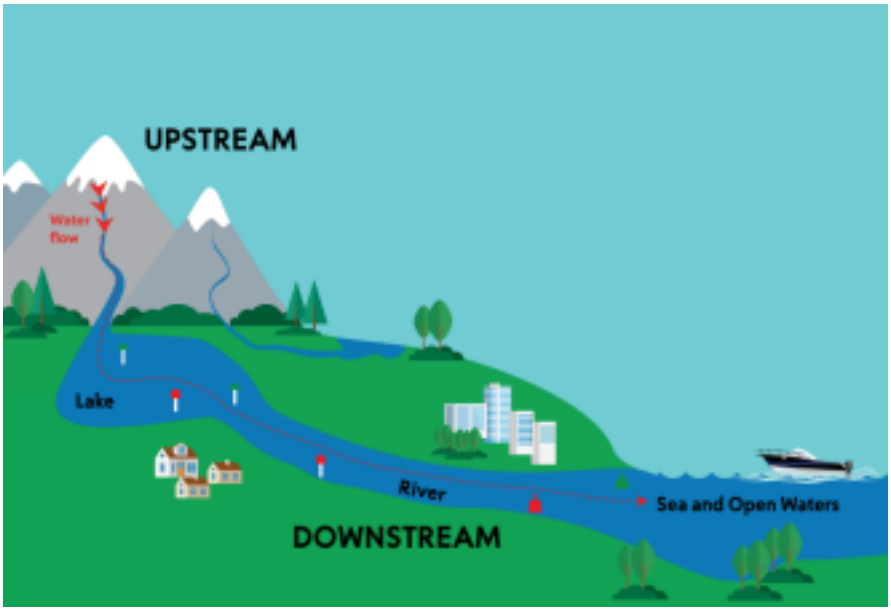
```
$ git clone
https://codeberg.org/$USERNAME/git-workshop-25-jan-2024.git

$ cd git-workshop-25-jan-2024
```

3. :
   a. Note that the URL is just copied and pasted as demonstrated in the screenshot above:

```
$ git remote add upstream
https://codeberg.org/singlerider/git-workshop-25-jan-2024.git
```

"Upstream" is always the code that everyone else shares. Like a river, it flows "downstream." Your (the fork with your username) will always be called "origin." "origin" is always "downstream" of "upstream!" Always! You can technically name it whatever you want, but this is one of git's "best practices." It standardizes things a bit so everyone can communicate effectively, no matter which team they're on.



We've got it all numbered and ready to go. We'll all walk through the steps to merge some “code" back into "upstream" using the Common Workflow at the top of the page. Save these steps for the future, when you'll need them!