Git Workshop

Quick Reference

Initial Setup of the Repo

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

Common Workflow

1. Fetch upstream
    a. `git fetch upstream`
2. 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: **Stage** (**GOTO 3**, then **GOTO 5a)** and **commit** (**GOTO 5b**) files
        ii. `git reset --hard`
        iii. GOTO 2
3. 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
4. Do work (write code, create art files)
5. 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. **GOTO 4** as needed
6. Push to origin new-branch
    a. `git push origin NEW-BRANCH-NAME`
    b. **GOTO 4** as needed
7. Open/Modify Pull Request against "upstream/main"
    a. **GOTO 4** as needed
    b. If accepted, merge
8. GOTO 1

Introduction

Why use git?

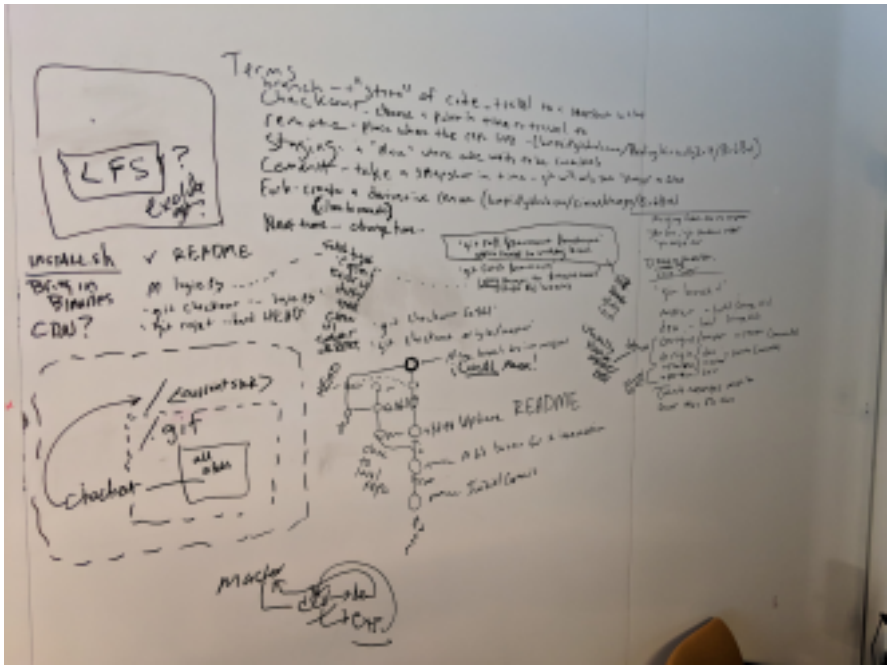
What is git, in the first place?

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

So what?

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.

I'm scared



I know; It's okay. It's not as bad as it seems.

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

Terms

Repository

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

Remote

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

Commit

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

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

Branch

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?"

Checkout

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

Merge

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

Push

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

Pull

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.

Fetch

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

Setup

Codeberg Account

https://codeberg.org

Git Installation

Mac

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
```

Linux

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

Windows

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

First-time configuration

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
```

Create a Home for Projects

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

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

SSH (optional)

Check for a valid output from `ssh-add -L` . If you see an error, follow this section, otherwise, skip it. **If you're on Windows, you'll need to run this first command included in the list below with Git Bash every time Git Bash is reopened** (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)
```

Customization (optional)

.gitconfig

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 "**aliases**" that will make common git operations much easier.

Practical Usage Examples

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. **Fork** the "git-workshop" repository:
    a. https://codeberg.org/singlerider/git-workshop-25-jan-2024
2. **Clone** the forked repository. Note how it's no longer under the "singlerider" user:

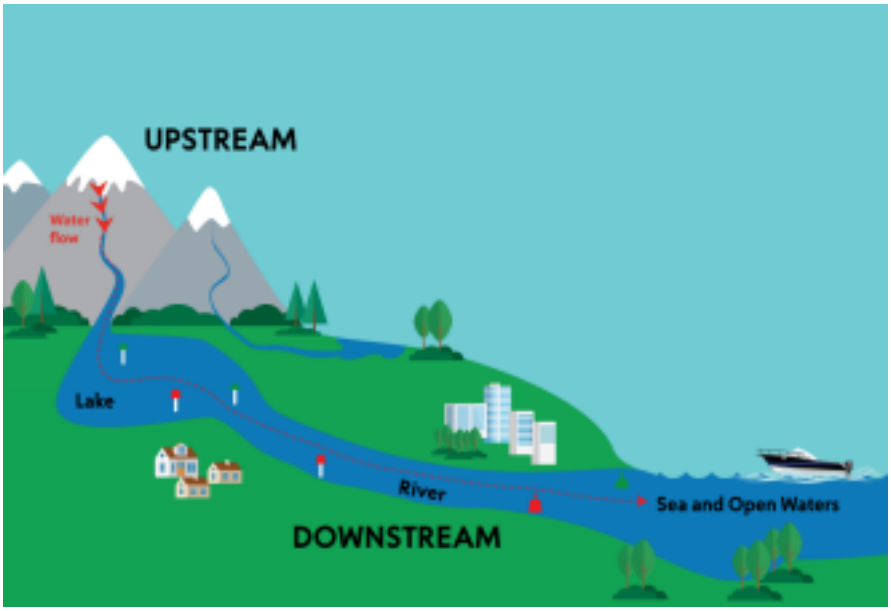
```
$ git clone
https://codeberg.org/$USERNAME/git-workshop-25-jan-2024.git

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

3. **Set up remotes**:
    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 **remote** (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.



Let's Merge a Pull Request

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!