

# Intro To Git & GitHub

---



## Students will be able to:

1. Describe the purpose of version control
2. Understand the differences between Git and GitHub
3. Fork or clone repositories
4. Create a repository
5. Use these commands

- `git init`
- `git clone`
- `git add`
- `git status`
- `git commit`
- `git branch / git checkout -b`
- `git checkout`
- `git merge`
- `git push`
- `git pull (git fetch + git merge)`
- `git log`



When you spend all your time  
on feature branches

---

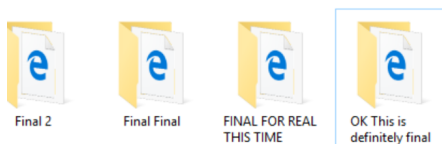
Git Jokes - git with it 🤖

- <https://github.com/EugeneKay/git-jokes/blob/lulz/Jokes.txt>

---

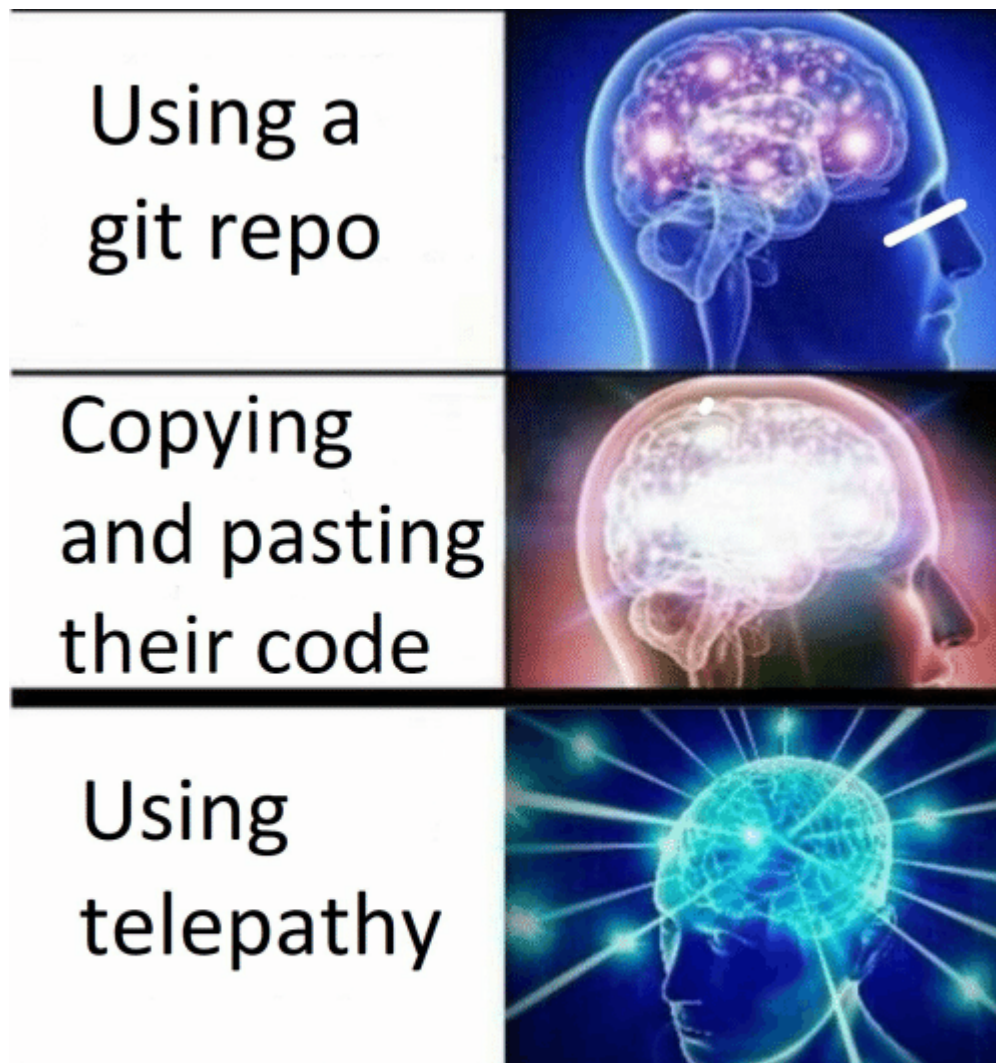
## What Is Version Control?

- Part Time machine (think sci-fi, multiple timelines), part Google doc (multiple collaborators)
- Simultaneous collaboration, avoid conflicts
- Developers working in teams are continually writing new code and changing existing code. Version control systems helps us keep track of our code changes over time as well as help us prevent concurrent work from conflicting when working with other developers! Using modern day version control systems, one developer on the team can be working on a new feature while another developer is fixing an unrelated bug at the same time.
- Time Machine: If we find a bug in our code, we can simply compare earlier versions of the code to help fix the mistake. Rollbacks.



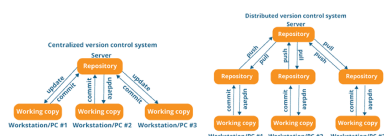
## Benefits of version control

- Helps with troubleshooting! Allows us to have a *complete* history of EVERY FILE - EVERY SINGLE CHANGE made by sometimes many individuals over the years! Each change keeps track of the creation/deletion of files, edits to the contents of files, the person who made the changes, the date/time, and written notes about the purpose of each change (commit messages).
- Improves the development process! Branching and merging! Because of these features, teammates can work on the same project at the same time even while working on completely different tasks. Branching is even beneficial for an individual working on their own because it allows them to work on independent streams of changes.



## Version control with group projects

### Overview of Centralized vs Distributed Version Control Systems



In a centralized version control system (Subversion, Clearcase, Perforce, etc), you have a database on one server or system. That's where the project resides. You can download a snapshot of part of the latest of the project. You pull the files that you need, but you never have a full copy of your project locally. You can then start working on that project and check in incremental changes of some sort to the codebase on the server. You have to operate through that central server, though, so if the network is down, you will not be able to check in their code. The entire database lives on that central server, so if you lose that central server and don't have a back up of it, then that project is gone! A few people who checked out snapshots may have some of the later snapshots of the project, but that's it.

In distributed systems (Git, Mercurial, etc), you clone the whole project instead. You take every bit of the project that's on the server and copy it to your local hard drive. You can then start working on the project locally and you don't have to be online for most operations. The only operations for which you need an internet connection are the ones for synchronizing (push & pull). Because you can work locally, operations tend to be quicker since there's no network latency. If the central server goes down, everyone has a copy of the database. Everybody's clone in some state is a full backup of the project. Moreover, you can push and pull directly to peers or to the central server.



mercurial



---

## Git & GitHub

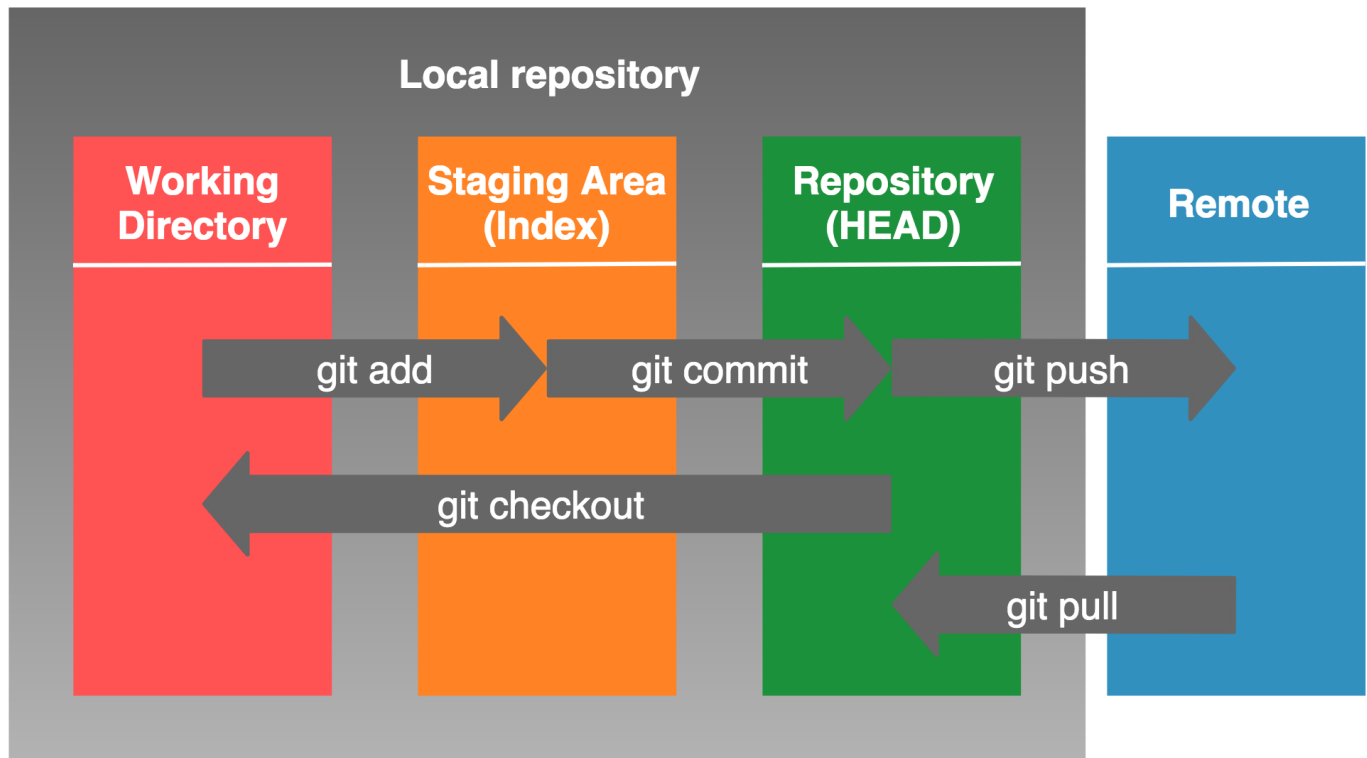
[Git](#) is a distributed version control system used for tracking changes in source code. Git has been around since 2005. Git is [free and open source](#)!

[GitHub](#) is a source code hosting service that hosts git projects. GitHub has been around since 2008 and was written in Ruby, ECMAScript, Go, and C. Some other examples of popular source code facilities include [Bitbucket](#), [GitLab](#), and [SourceForge](#).



---

## Git Stages



The first area is the working area, which is also known as the working tree. The second area is the staging area, which is also known as the cache or the index. Then we have the local repository and remote repository.

The working area contains the files that are not handled by Git. They're just in your local directory and can also be referred to as untracked files. The working area is like your scratch space; it's where you can add new code, modify code, or delete code, etc.

The staging area - whatever's in the staging area are what files are going to be included as part of your next commit. It's how Git knows what's going to change between the latest commit and the next one.

[https://en.wikipedia.org/wiki/Staging\\_area](https://en.wikipedia.org/wiki/Staging_area)

Then, the repository are the files and code changes that Git knows about and has saved into its database. The repository contains all of your commits and a commit is just a snapshot of what your working and staging area look like at the time of the commit. Those live in your `.git` directory.

---

## Git Commands Cheatsheet

- `git init` initializes a git repository

### Saving Changes

- `git add <filename>` adds a file or changes in a file to a repository
- `git add -A` adds everything in current directory (files and changes) to a repository
- `git commit -m <message>` saves changes you've made to the repository

### Reverting Changes

- `git reset <Log Number>` resets git repo to specific commit

- `git reset --hard <Log Number>` reset git repo, and current directory to specific commit
- `git commit --amend` Adds changes to previous commit
  - `git commit --amend -m "New message"` changes your previous commit message

## Working with Remotes

- `git remote add <remote_name> <url>` connects repo to a remote url (usually github)
- `git remote rm <remote_name>` removes a previously added remote
- `git remote -v` lists all of your remotes
- `git push <remote_name> <branch>` pushes changes to a remote git repo (usually github)
- `git fetch <remote_name> <branch>` fetches change from a remote, but does not merge into local repo
- `git pull <remote_name> <branch>` pulls and merges changes from a remote git repo (usually github)
- `git clone <url>` copy's a repo from github

## Working with Branches

- `git branch` lists different branches
- `git branch <new_branch_name>` creates a new branch
- `git checkout <branch_name>` moves you to the branch specified
- `git checkout -b <new_branch_name>` creates a new branch, and moves you to new branch
- `git merge <branch_name>` merges the specified branch into the working branch





Origin



Master

```
>git merge
```



## Helpful Commands

- `git help` lists possible git commands
- `git status` shows changes that have not been committed
- `git log` shows commit history
- `git diff` show changes between commits, commit and working tree, etc
- `git config -l` show configs
- `git config --global user.name "John Doe"` sets a name that will be attached to commits
- `git config --global user.email johndoe@example.com` sets an email that will be attached to commits

---

## Baisc Git Workflow

--

**Step 1:** Create a new directory `mkdir new_directory` *Note: A directory is not the same as a repository*

**Step 2:** Move into your new directory `cd git_practice`

**Step 3:** Initialize a new repository `git init`



**Step 4:** Write some code and Add changes to staging `git add .`

**Step 5:** Commit changes to repo `git commit -m "My changes"`

**Step 6:** Repeat **steps 4 and 5**

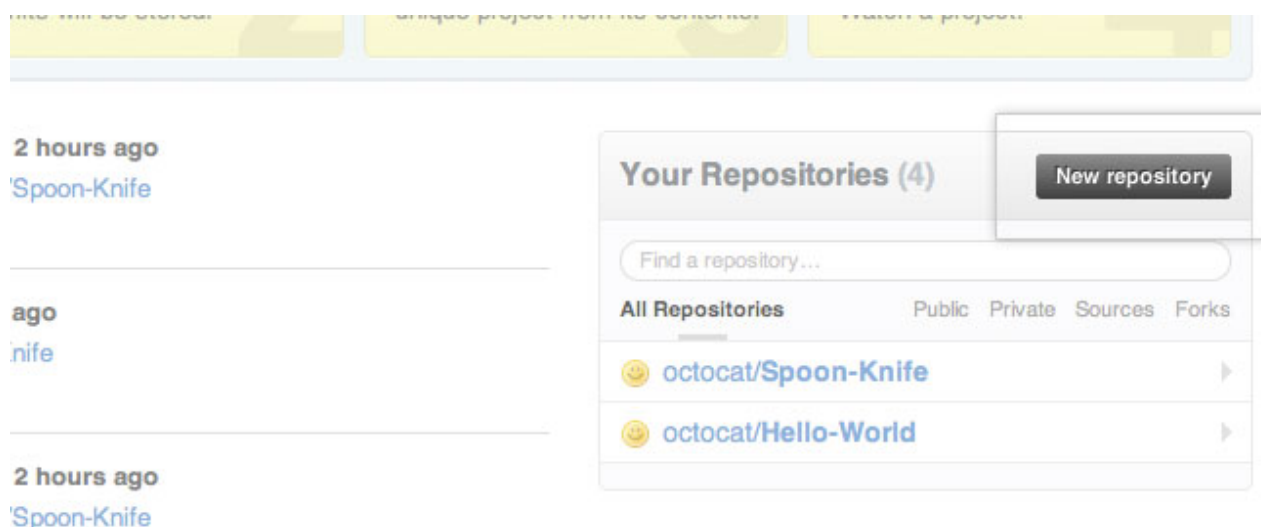
**Step 7:** Create a repo on Github

---

## Working with Github



### How to Create a Repo

1. Click "New Repo"




2. Fill out the information on this page. When you're done, click "Create Repository."


**Owner** **Repository name**

**PRIVATE**   **github** /

Great repository names are short and memorable. Need inspiration? How about **flaming-tyrion**.

**Description** (optional)

☐  **Public**  
Anyone can see this repository. You choose who can commit.

☒  **Private**  
You choose who can see and commit to this repository.

☐ **Initialize this repository with a README**  
This will allow you to `git clone` the repository immediately.

Add .gitignore: **None**

**Create repository**

3. Congratulations! You have successfully created a repository!

**Step 8:** Connect your Github Repo to your local repo `git remote add origin git@github.com:user_name/my_repo.git`

**Step 9:** Push your local repo to Github `git push origin master`

**Step 10:** Continue to add, commit, and push changes `git add . git commit -m "new message" git push origin master`

---

## Git Branching and Merge Conflicts

### Creating a Branch

There are two ways to create a branch in git.

**Option 1:** `git branch <new_branch>` creates a new branch `git checkout <new_branch>` moves you to the new branch

**Option 2:** `git checkout -b <new_branch>` creates a new branch and moves you to it

### Moving Between Branches

To move between branches: `git checkout <branch_I_want_to_go_to>` moves you to specified branch.

To list available branches: `git branch` will list all branches.

### Merging Branches

We can combine two branches together by using `git merge`

Let's say we have a branch **master** and a branch **feature**

In order to combine the two (from the master branch), we run...

```
git merge feature
```

---

## Merge Conflicts

Git is smart, but not that smart. Merge conflicts occur when the changes between 2 branches overlap, and git is not sure which version you want.

When you are trying to merge and a conflict arises, your terminal will tell you, and it will look something like this...

```
Auto-merging test.rb
CONFLICT (content): Merge conflict in test.rb
Automatic merge failed; fix conflicts and then commit the result.
```

The conflicts will also appear in your files with special tags.

```
puts "Hi"  
puts "More things"
```

The first section of the code is from our master branch and the second is the code we are trying to merge in.

Here you will have to manually select the code you want.

```
puts "adkljfdklsajf"  
puts "Hi"  
puts "More things"
```

After you select the code you want to keep you will have to add and commit these changes.

```
git add .  
git commit -m "Fixed conflict"
```

## How to Create a Pull Request on Github

*Before you can open a pull request, you must create a branch in your local repository, commit to it, and push the branch to a repository or fork on GitHub.*

1. Visit the repository you pushed to
2. Click "Compare and Review" in the repository
3. You'll land right onto the compare page. *(You can click Edit at the top to pick a new branch to merge in, using the Head Branch dropdown)*
4. Select the target branch your branch should be merged to, using the Base Branch dropdown
5. Review your proposed changes.
6. Click "Click to create a pull request" for this comparison
7. Enter a title and description for your pull request
8. Click Send pull request

## Resources

- [Official Documentation:](#)
- [Try Git Tutorial:](#)
- [Git Hug game:](#)
- [Other Resources:](#)
- [Setting up SSH Keys:](#)
- [General Github Help:](#)

- [Version Control Back in the Day Discussion](#)
- [Merge vs Rebase](#)
- [Inner workings of git](#)
- [Interview with creator of Git](#)