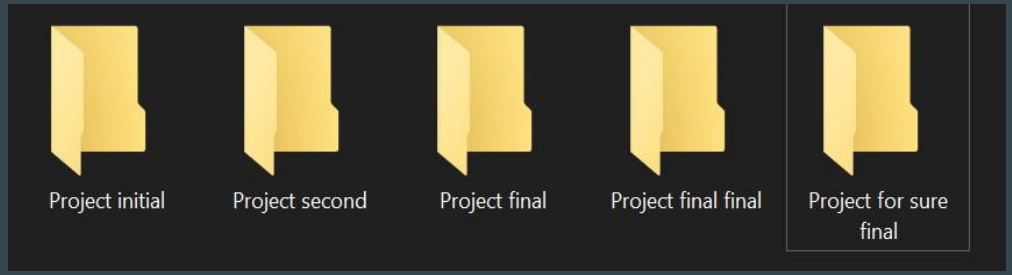
Git & GitHub Crash Course

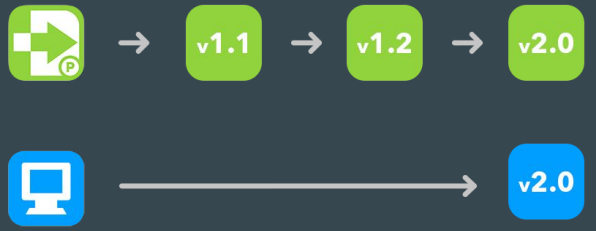
UW Code Force

Steven Simko & Meharban Singh

What is Git?

- Version Control System

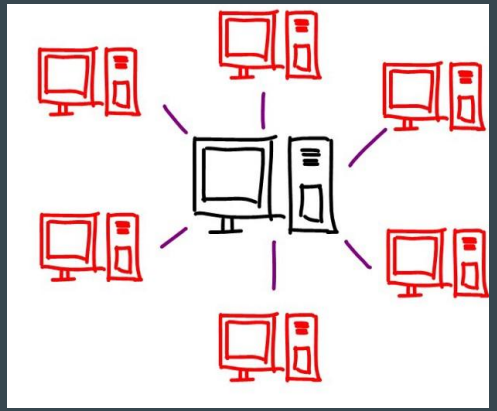




https://www.kioskproapp.com/images/rebranding-version-control.png

- Distributed VCS
  - Prevents Single point of Failure

Note: Git isn't the only VCS, although it is the best (that's what most people think so you better agree).



https://i.ytimg.com/vi/YS-QvfCZWvc/maxresdefault.jpg

Why use Git?

- Git allows multiple developers to work on the same project together!
- Easy to track what changes were made to a project, who made them, when they were made and why they were made.
- Allows you to rollback your code to a previous working version.
- Synchronize your code across devices (as long as you push your changes often!)
- Work on more than one version at a time.
    - Release a game to users but work on a new version without messing with the older published version.
    - A concept called “branches” are used to represent different versions.

Git Initial set-up

1. Download git from https://git-scm.com/downloads and open Git Bash.

2. You need to tell git who you are so it remembers the code you are saving. To provide an email and username,

```
git config --global user.name "Your Name"

git config --global user.email "youremail@example.com"
```

Git terminology

Repository

- Local and remote

Commit

Push

Pull

Fetch

Branch

- Master

Merge

Conflict

README.md

https://git-scm.com/docs/gitglossary

Some commands you will use everyday

- Add a file to a commit

  `git add .`

- Commit

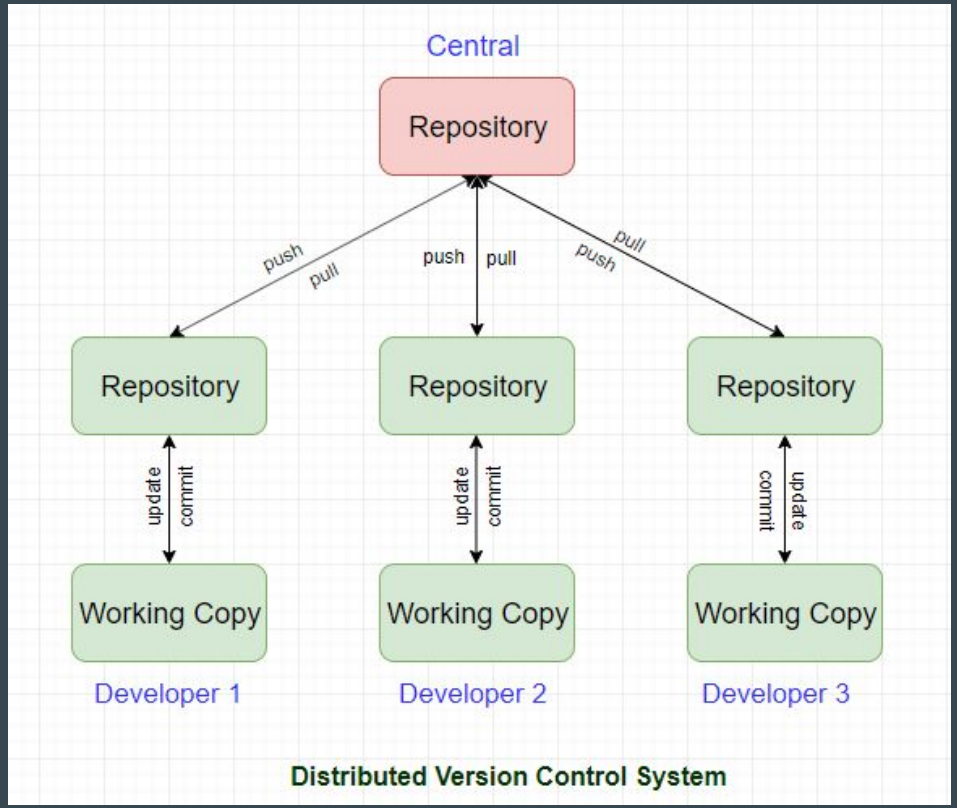
  `git commit -m 'message'`

- Push commits to repository

  `git push`

- Pull changes from repository

  `git pull`



https://ostechnix.com/wp-content/uploads/2019/12/Distributed-version-control-system.png

Some commands you will use everyday cont.

- Check status of a commit

  `git status`

- Fetch branch updates

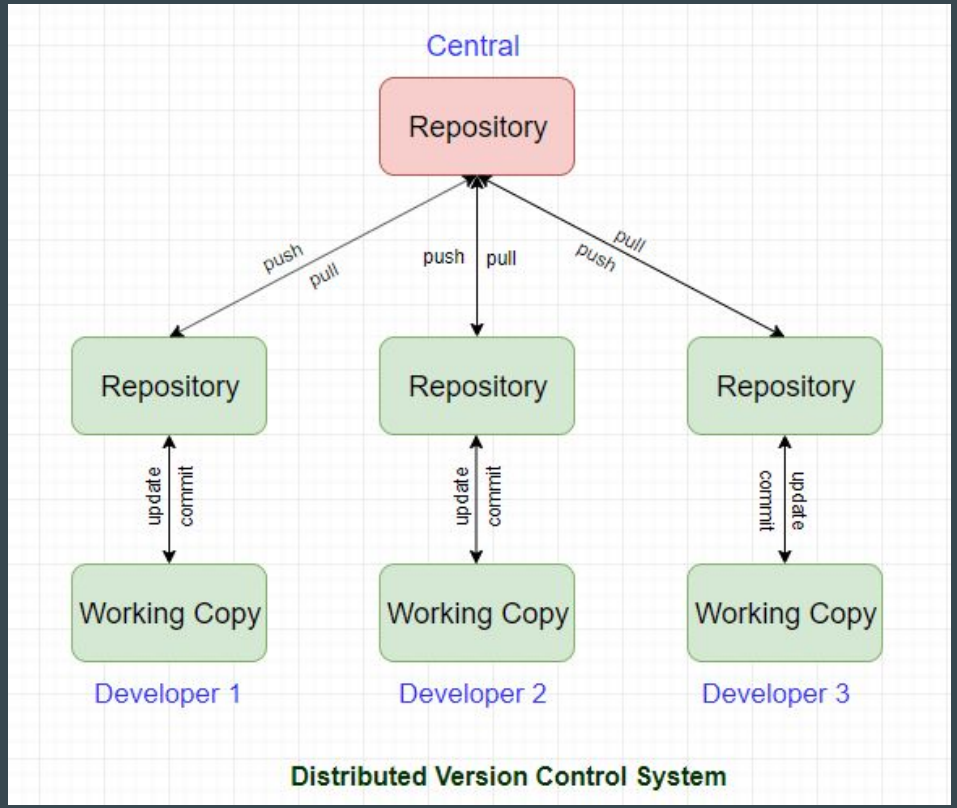
  `git fetch`

- Change branches

  `git checkout 'branch'`

- Merge branches

  `git merge 'branch to merge with'`



https://ostechnix.com/wp-content/uploads/2019/12/Distributed-version-control-system.png

What is a Commit?

- Code that is ready to be pushed to a repository (staged).
  - Say you make some changes to `HelloWorld.java`
  - You can add this file to a commit by typing `git add HelloWorld.java`
  - Add a message to go along with this change with `git commit -m "Renamed string text to message"`
  - Now your changes, along with your message are ready to be pushed to your branch `git push`
- What if I made a mistake in my commit?
  - `git reset --soft HEAD~1` will remove the commit, but keep the changed files
  - `git reset --hard HEAD~1` will remove the commit, and reset the modified files to the last push

How to make a Commit

- This is best described with an example. In this example, we will work with 3 files. `index.html`, `style.css` and `server.js`
- We have made changes to all 3 files, and need to commit the changes.
- `index.html` and `style.css` have similar changes made, so we can commit these together.
  - `git add index.html`
  - `git add style.css`
  - `git commit -m "Colour scheme changes"`
- This will create a commit with both files, with the specified message. We can then make another commit for the `server.js` file.

How to make a Commit

- Like the previous files, we will need to add it with the following commands
  - `git add server.js`
  - `git commit -m “Added database dependencies”`
- We can then push both these commits with `git push`
  - Note that you can technically add all your files to one commit, but it’s best practice to add specific messages declaring what you have changed in each file.

Pushing changes to remote

Once you have all your commits on the local, you can also push them to the remote server by running:

```
git push
```

When you are pushing for the first time, you might need to specify what remote server you are pushing to. So first associate a new remote to the repo:

```
git remote add origin https://github.com/Your-Name/repo.git
```

And then permanently set this remote as the default repo to push/pull from:

```
git push --set-upstream origin master
```

Pulling changes from remote

If another developer pushed the code on the remote server, your local repo is a few commits behind the one on remote. To update your local repo, you need to pull those changes from the remote to your local.

```
git pull
```

This would update your local repo to match the remote repo. This is also called as merging of the branch in your local with with branch on the remote repo.

Merge Conflicts

Most of the times when you merge two branches (push or pull changes), its flawless and you do not have to make any adjustments in the code.

However, sometimes when pushing or pulling changes, git cannot decide how to merge the code in some places.

This is called a merge conflict. This happens when you push/pull some changes to/from another repo but the two repos had some changes in the same areas of the code.

Merge Conflicts: Example

Say a developer A pulls the code from the remote repo onto his local, so now he has the most updated version of the code. Now he starts making some changes on an if condition on line 21 in a file.

At the same time, a developer B makes some changes on his local on the same if condition on line 21 and pushes the code on the remote.

Now the remote has some extra changes that dev A is unaware about. When he will try to push the changes from his local to the remote repo, GitHub won't know how to merge the two lines since both of them change the if condition and GitHub won't know which one to keep. This is a merge conflict.

Example cont: solving the conflict

In such a situation, dev A needs to `fetch` the changes from the remote while not affecting the changes he made. If he pulls the changes, they merge automatically - which would give an error since we have a merge conflict.

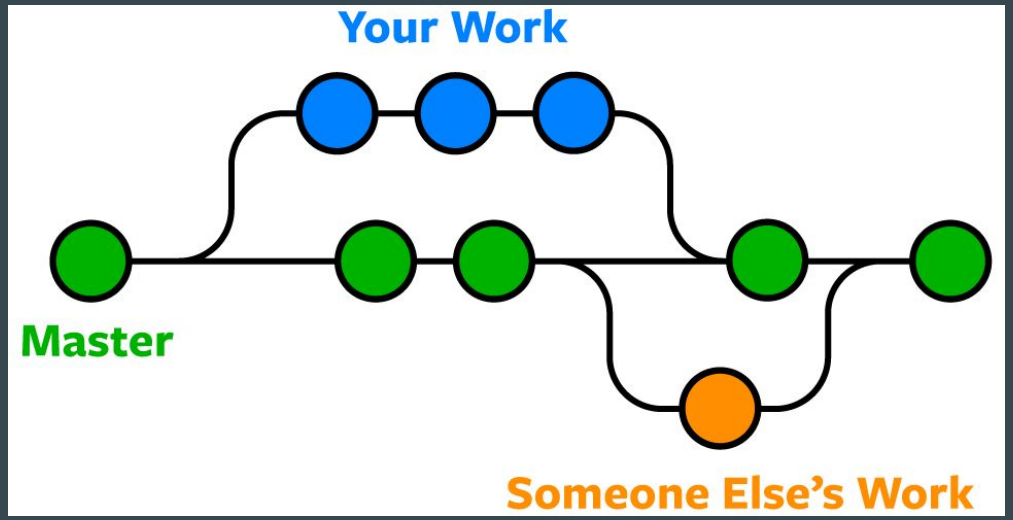
So, dev A could `git fetch` the changes from the remote which would place both changes together so dev A can decide which one to keep ... or to get rid of both of them.

Once the conflict is resolved, dev A can commit and push the changes to the remote again.

```
<<<<<<< HEAD
if(i > 20) {
=======
if(i >= 20) {
>>>>>>> remote
```

What is a Branch?

- As stated earlier, a branch is a different “version” of your project.
- Everyone should work on their own (up to date) version!



https://www.nobledesktop.com/learn/git/git-branches

The Workflow

1) Create repository for your project, add collaborators.
2) Initialize a git directory (*git init)* push an existing project. If the project was first created on GitHub, clone the project to a local directory.
3) Create branches for different features or developers.
    a) I.e we might make a branch for an authentication feature
    b) Or Steven might have his own personal branch, Meharban might have his own branch etc.
4) Write your feature. Make sure to rebase (merge) with the master branch often!
    a) This ensures that any changes that have been pushed to master while you've been working on your feature are accounted for! Keep your branch up to date, otherwise it will be a pain to deal with later!
5) Let your lead programmer know when you're ready to merge your feature! Don't mindlessly merge into master!
6) Repeat steps 4-5 :)

README.md

- A markdown file (.md) that is used to list any details/documentation about the project.
  - Markdown is like HTML but much faster to write.
  - Has special formatting rules instead of the HTML tags.
    - Example:
      `<h1>Heading</h1>` in HTML
      `# Heading` in markdown
  - Markdown cheatsheet: https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet
- Shows up as the first readable file in any project on GitHub.
- Good place to explain what the project is, what it does and who made it.
- Example README.md file for uw-codeforce: https://github.com/Meharban-Singh/parcel-tracking-system/blob/master/README.md

[Hint: Click `Raw` and you can see the document without the markdown formatting. Feel free to use the file as a template.]