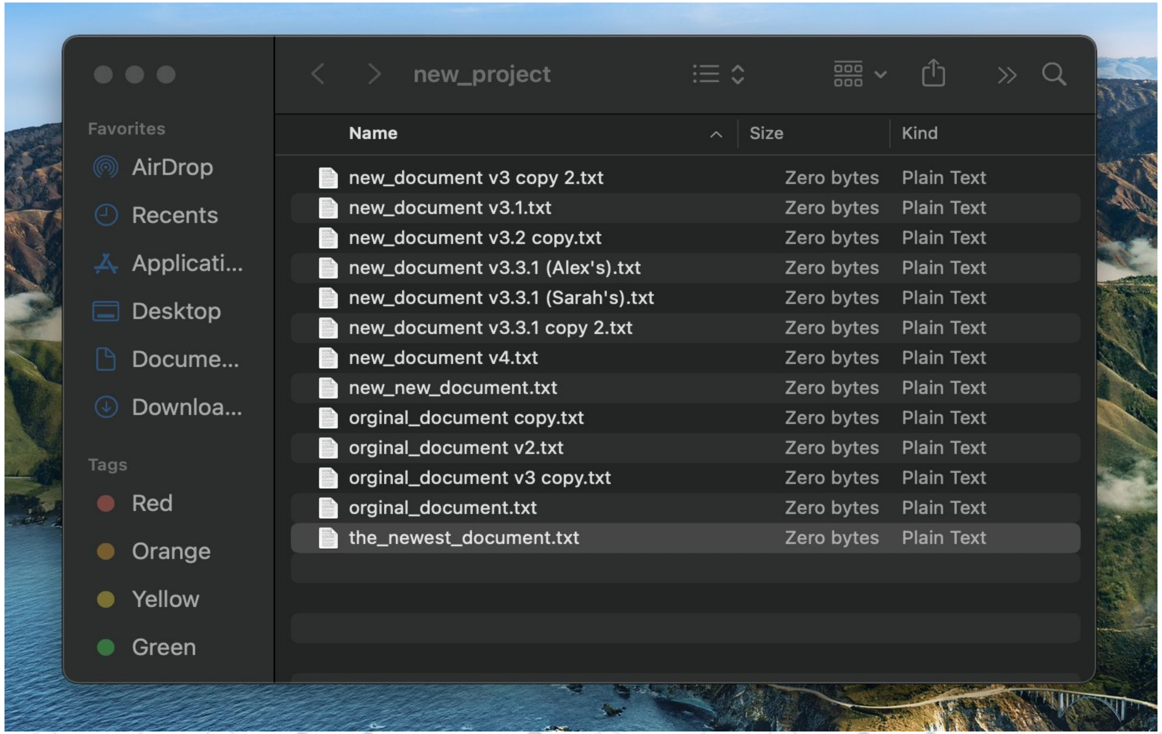
Backing Up & Sharing Code With Git

Roots Course By Anni Yan

What is Version Control?



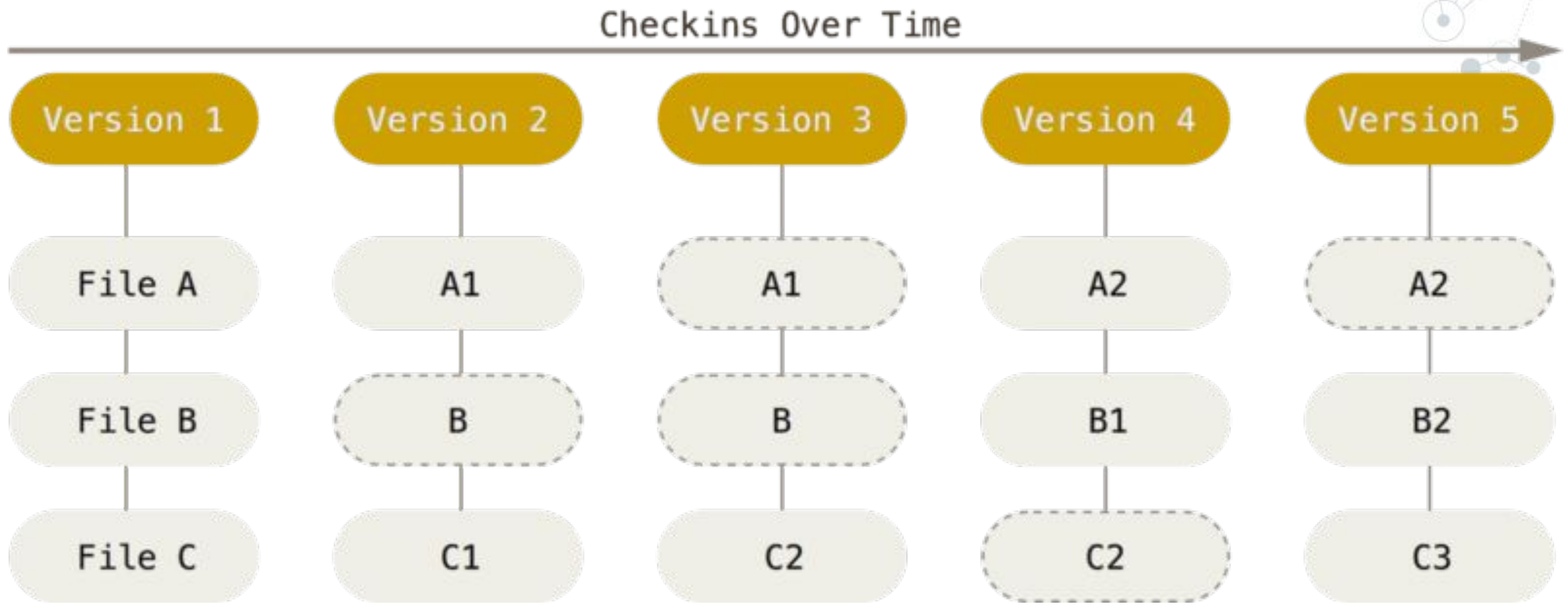
Name	Size	Kind
new_document v3 copy 2.txt	Zero bytes	Plain Text
new_document v3.1.txt	Zero bytes	Plain Text
new_document v3.2 copy.txt	Zero bytes	Plain Text
new_document v3.3.1 (Alex's).txt	Zero bytes	Plain Text
new_document v3.3.1 (Sarah's).txt	Zero bytes	Plain Text
new_document v3.3.1 copy 2.txt	Zero bytes	Plain Text
new_document v4.txt	Zero bytes	Plain Text
new_new_document.txt	Zero bytes	Plain Text
orginal_document copy.txt	Zero bytes	Plain Text
orginal_document v2.txt	Zero bytes	Plain Text
orginal_document v3 copy.txt	Zero bytes	Plain Text
orginal_document.txt	Zero bytes	Plain Text
the_newest_document.txt	Zero bytes	Plain Text

What is Version Control?

It is the practice to document and organize code in our project. We can:

- keep track of file changes
- view change history
- work on multiple versions of the project at the same time
- And much more...

What is Git?


Checkins Over Time
Version 1
Version 2
Version 3
Version 4
Version 5
File A
File B
File C
A1
B
C1
A1
B
C2
A2
B1
C2
A2
B2
C3

Install Git

- Git != Github/Gitlab
- Edstem Self-paced course
- Use `git --version` to check if it’s installed and what version you have

Create a git project

- Create a directory:

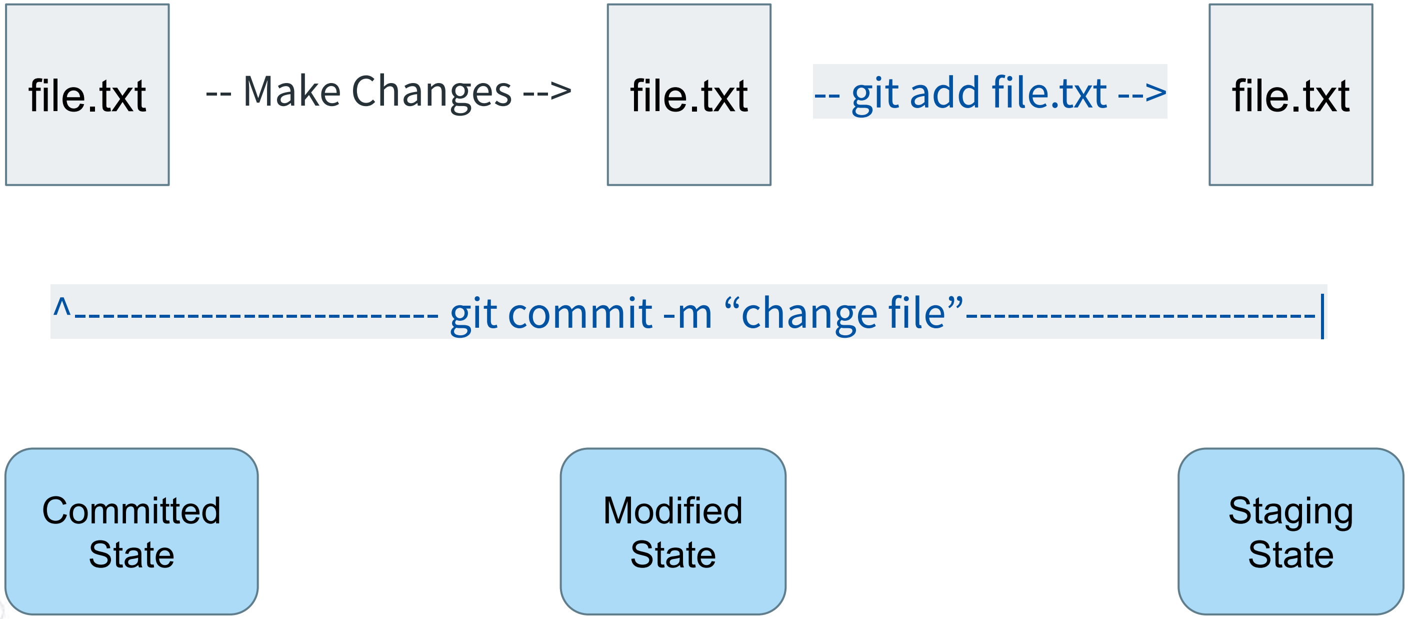
```
$ mkdir project
```

- Use **git init** to turn it into a git project:

```
$ cd project
```

```
$ git init
```

Git Commit Flow



Three states in Git

- **Modified**: you modified your file but has not committed to your git history
- **Staged**: you marked the modified files to go into git history
- **Committed**: the staged files are now in git history

Staging files

- Add a file:

```
$ git add filename.file
```

- Add a directory:

```
$ git add path/to/a/directory
```

- Add all:

```
$ git add .
```

Commit files

- Commit with a message:

```
$ git commit -m "describe what you did"
```

- Add and commit at the same time:

```
$ git commit -am "describe what you did"
```

- Replace the last commit:

```
$ git commit --amend
```

Git branches

- See branches:

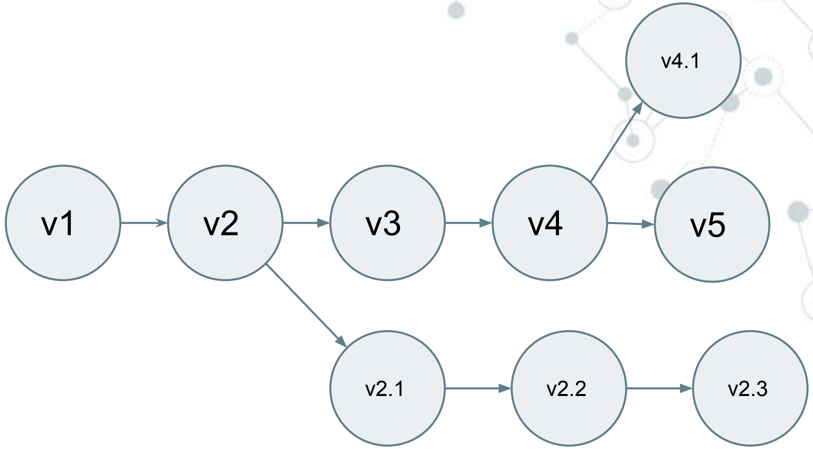
```
$ git branch
```

- Create a new branch:

```
$ git branch <branch-name>
```

- Checkout a branch:

```
$ git switch <branch-name>
```



Combine branches

- Combine a branch into main:

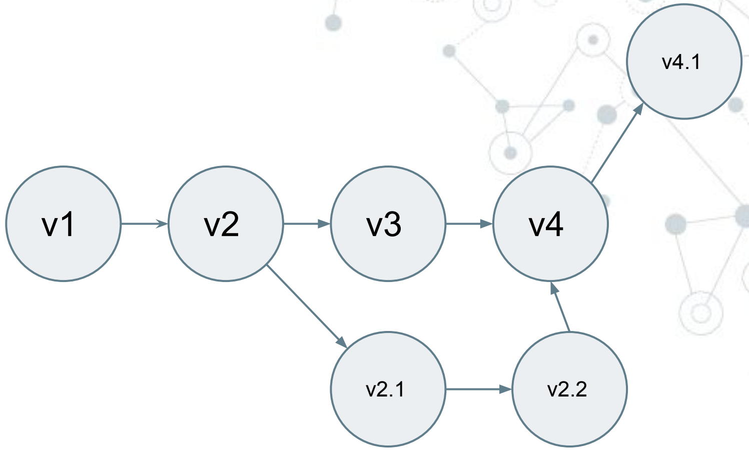
```
$ git switch main
```

```
$ git merge <branch-name>
```

```
$ git rebase <branch-name>
```

- Delete a branch:

```
$ git branch -d <branch-name>
```

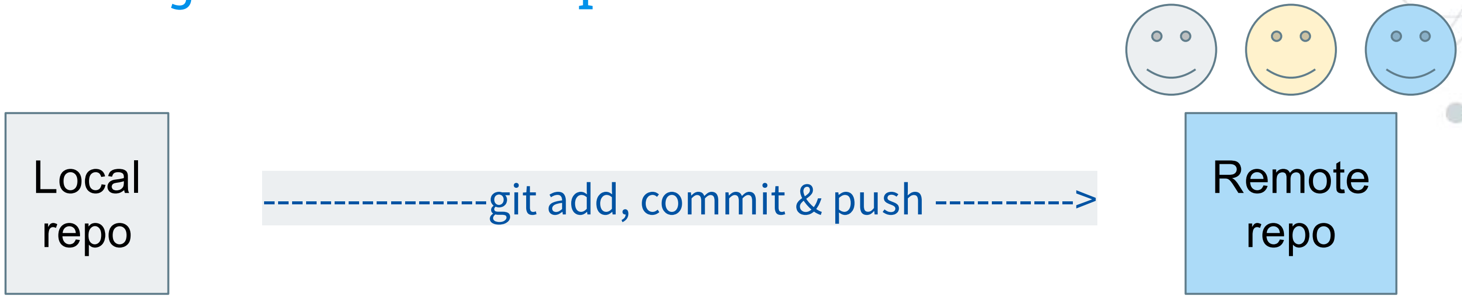


Let’s talk remotely!

What is remote repository?

- **Remote repository(repo)** means that your project lives somewhere online and you can share it with people.
- If you don’t have a Github or Gitlab account, now it’s your time to sign up.

Working with remote repo



First time:

^--------------- git clone https://gitlab.oit.duke.edu/project--------------------|

Rest of the time:

^-- git pull---|

Config user

- Check configuration:

```
$ git config --list
```

- Set global username and email:

```
$ git config --global user.name "John Doe"
```

```
$ git config --global user.email “johndoe@example.com”
```

Create a remote repo

- Add remote repo:

```
$ git remote add origin https://gitlab.oit.duke.edu/project
```

- See remote shortname and address:

```
$ git remote -v
origin   https://gitlab.oit.duke.edu/project (fetch)
origin   https://gitlab.oit.duke.edu/project (push)
```

Download an existing remote repo

- Clone a remote repo:

```
$ git clone https://gitlab.oit.duke.edu/project
Cloning into 'project'...
remote: Reusing existing pack: 1857, done.
remote: Total 1857 (delta 0), reused 0 (delta 0)
Receiving objects: 100% (1857/1857), 374.35 KiB | 268.00 KiB/s, done.
Resolving deltas: 100% (772/772), done.
Checking connectivity... done.
```

Update remote repo

- Pull from a remote repo:

```
$ git pull
```

- Push to a remote repo:

```
$ git push
```

Resolving conflicts

- Try to push:

```
$ git push
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.
```

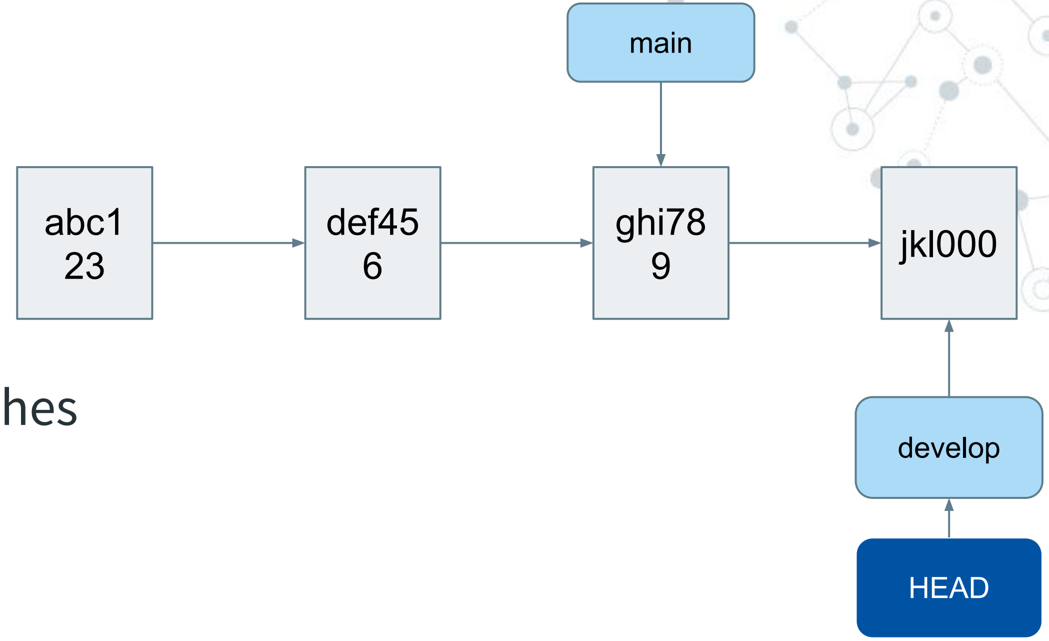
Resolving conflicts

◎ What it looks like:

```
<<<<<<<< HEAD:index.html
<div id="footer">x is 1</div>
========
<div id="footer">x is "red"</div>
>>>>>>>> master:index.html
```

Working together

1. Commit often
2. Create a **main** branch
3. Work on **feature** branches
4. Merge into **main**
5. Create **issues** in Gitlab
6. Ask your team leads



Git best practices

- See past commit history:

```
$ git log
```

- Ignore files, use .gitignore:

```
$ open .gitignore
```

- Create a README.md file:

```
$ touch README.md
```

Resources

- [Git documentation]
- [Git Beginner Tutorial]
- [Colab: install Git]
- [Github install Git]