



Git Introduction



Did you finish pre-class work?



Students, drag the icon!



Pear Deck Interactive Slide
Do not remove this bar

Git Journey



Table of Contents



- ▶ What is version control?
- ▶ What is Git?
- ▶ How to create a Git repository?
- ▶ Basic Git commands
- ▶ Git workflow

What do you know about Git? »

Let's discuss about Git



What is Git?



Git is an open source distributed
version control system





1

What's Version Control?



What's Version Control?



Version Control Systems

What comes to your mind when you hear this?





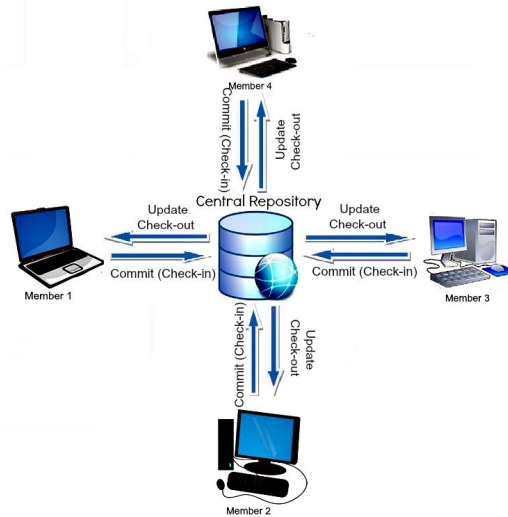
What's Version Control?

- Track changes on text files / source files for you
- Unlimited Undo / Redo
- Time Travel
- Collaborative development environment
- Compare and Blame
 - ◆ What changed
 - ◆ When it changed
 - ◆ Why it changed
 - ◆ Who changed it

Version Control Systems

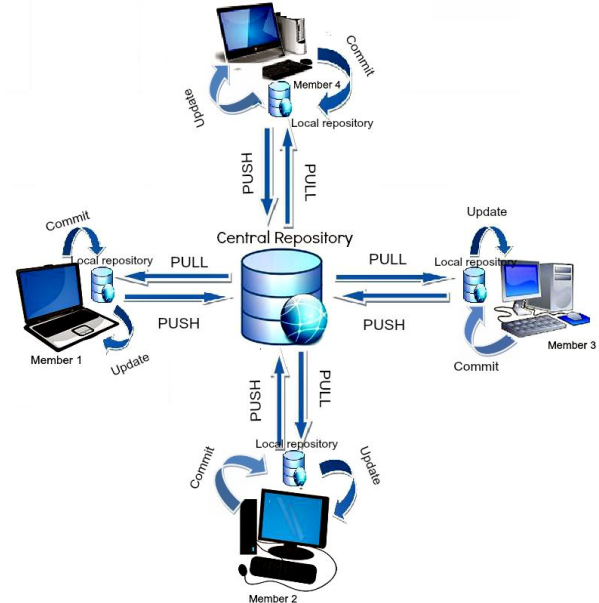
Centralized

A centralized VCS requires a server to perform any operation on your project. We need to connect to a server to download all our files to start to work. If this server goes down, we can't work.



Distributed

But, a distributed VCS doesn't need a centralized server in the same sense. Each user has a complete copy of the entire project. Most operations are performed against this local copy. We can work offline.





What's Version Control?



Version Control Systems (VCS)

- **Tracks** and **records** changes to files over time
- Can track any type of file, but most commonly used for code
- Contains extra information such as date, author, and a message explaining the change



What's Version Control?



Benefits of Version Control Systems (VCS)

- Can **retrieve** previous version of files at any time
- Retrieve files that were accidentally deleted
- Can be used **locally**, or **collaboratively** with others



2

What is Git?



What is Git?

- **Git** is a software
- Content Tracker
- Distributed Version Control System (VCS)
- Linus Torvalds

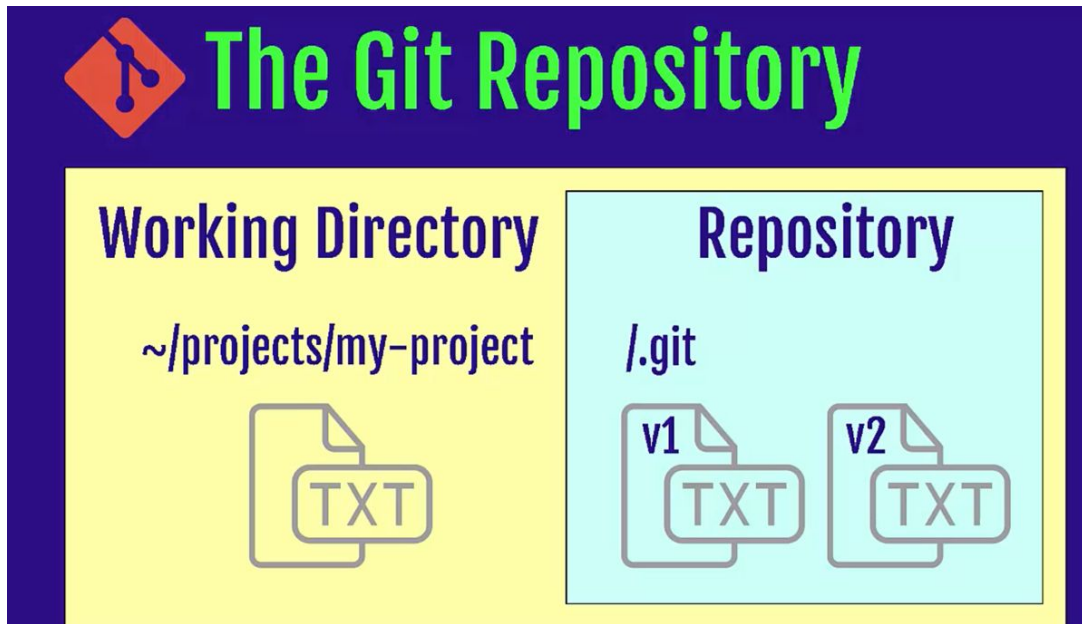




Git Repository

What is a repository

- A directory or storage space where your projects can live.
- Local Repository
- Remote Repository





Git Repository

→ Let's check if you have git in your computer

```
git --version
```

→ git needs your identity to mark/label changes / editor

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

```
git config --global user.email "Your Email"
```

```
git config --global core.editor "vim"
```

```
git config --list
```




Git Repository

→ to create a new local repo

```
git init
```

→ to see the commands

```
git help
```

→ to see the status of your repo

```
git status
```



Git Repository

- to create a new remote repo and connect it with your local repo (after you create a remote repo on Github/Bitbucket etc.)

```
git clone address
```



3

Workflow



Workflow

Working Directory

Where you work.
Create new files,
edit files delete
files etc.



Staging Area (Index)

Before taking a
snapshot, you're
taking the files to
a stage. Ready
files to be
committed.



Repository

Committed
snapshots of your
project will be
stored here with
a full version
history.





File Stages

Committed

Unmodified changes from the last commit snapshot

Modified

Changes made to files since last commit snapshot

Staged

Changes marked to be added into the next commit snapshot



Stage modified files & commit changes

Create a new file



Working Directory

resume.txt
untracked file



Staging Area (Index)



Repository



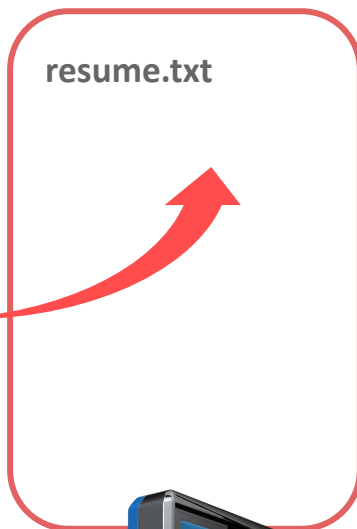


Track/stage a file

Working Directory



Staging Area (Index)



Repository





Stage files options

→ stage one file

```
git add filename
```

→ stage all files (new, modified)

```
git add .
```

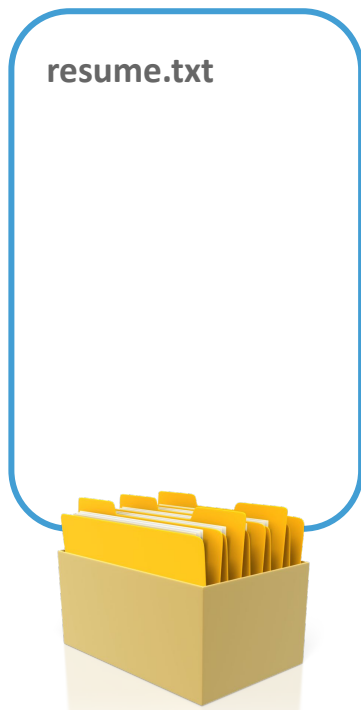
→ stage modified and deleted files only

```
git add -u
```

Commit



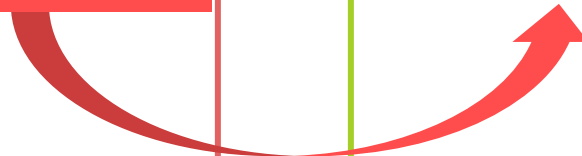
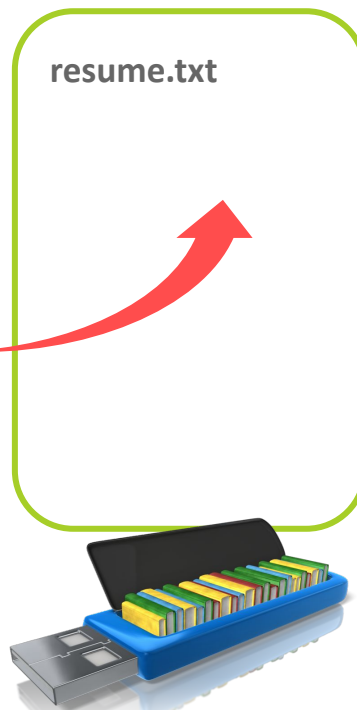
Working Directory



Staging Area (Index)



Repository



Commit



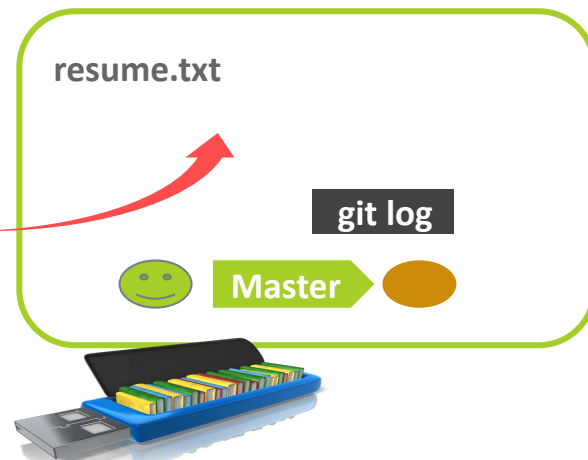
Working Directory



Staging Area (Index)



Repository



The `git commit` command captures a snapshot of the project's currently staged changes. Committed snapshots can be thought of as “safe” versions of a project—Git will never change them unless you explicitly ask it to.



Commit

- Commit the files on the stage

```
git commit -m "message"
```

- Add and commit all tracked files

```
git commit -am "message"
```

- amend commit message

```
git commit --ammend
```

Remove from stage

Working Directory



Staging Area (Index)



Repository



*`git rm` without `--cached` will simply remove the file from both index and working directory. `$$ git rm --cached` # exact opposite of `git add`. *`git restore --staged <file>` will again move the file from the staged to the unstaged area from either the index or another commit. This command does not update your branch. The command can also be used to restore files in the index from another commit.

Checkout from Repo



Working Directory

Staging Area (Index)

Repository

resume.txt
new_file.txt

resume.txt
new-file.txt

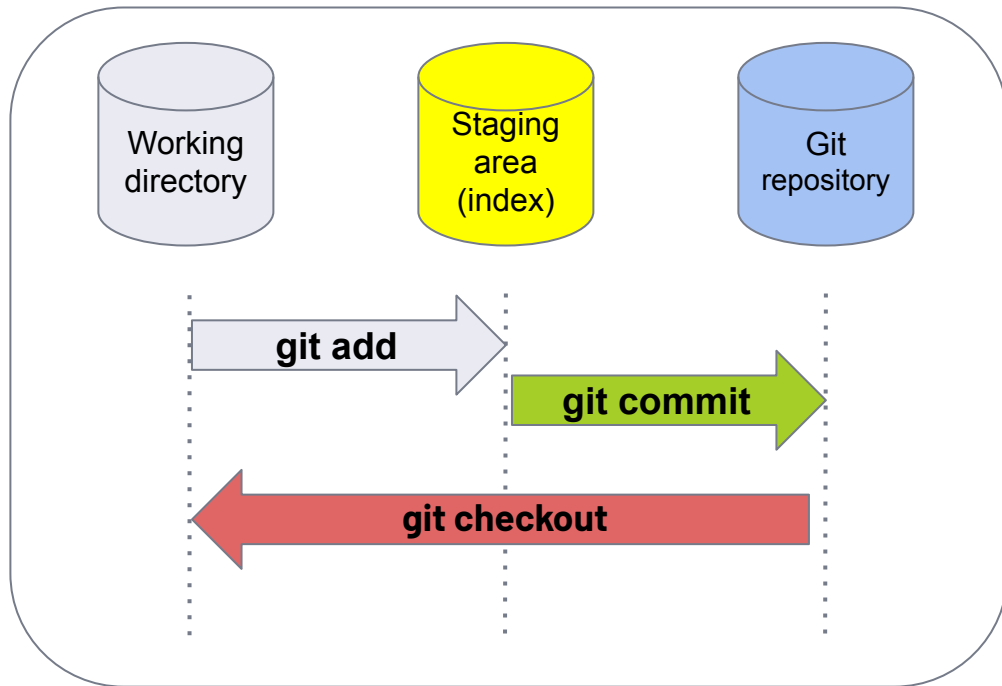
resume.txt
new-file.txt

git checkout

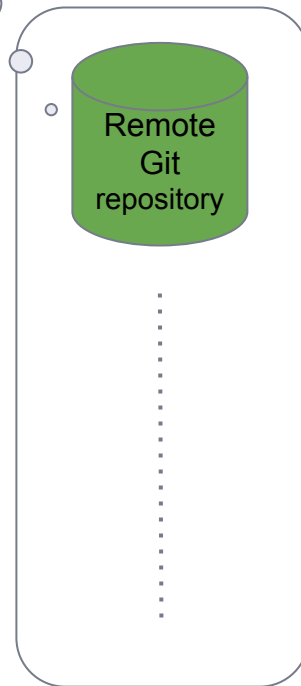




Local Machine



GitHub





New project

- Create a repo
- Create a new file/edit file etc.
- Stage/Track your changes
- Commit changes

```
git init
```

```
git add .
```

```
git commit -m "message"
```




Task-1

- Create a new repo under **project-3** folder
- Create a file named **file1.txt**
- Change the file
- Stage the file
- Commit the file to your repo





Task-1 Solution

→ Create a new repo under **project-3** folder

```
git init
```

→ Create a file named **file1.txt**

```
touch file1.txt
```

→ Change the file

```
vim file1.txt
```

→ Stage the file

```
git add .
```

→ Commit the file to your repo

```
git commit -m "message"
```

Task-2



- Create a file named **file2.txt**
- Edit **file2.txt**
- Stage
- Delete the file **file1.txt**
- Rename **file2.txt** >> **file3.txt**
- Stage **file3.txt**
- Unstage **file3.txt**
- Stage **file3.txt** again
- Commit the file to your repo
- Change the message of the commit
- Switch back to your first commit in **Task-1**



Students, write your response!



Task-2 Solution

- Create a file named **file2.txt**
- Edit **file2.txt**
- Stage
- Delete the file **file1.txt**
- Rename **file2.txt** >> **file3.txt**
- Stage **file3.txt**

```
touch file2.txt
```

```
vim file2.txt
```

```
git add .
```

```
rm file1.txt
```

```
mv file2.txt file3.txt
```

```
git add .
```



Task-2 Solution Cntd.

→ Unstage **file3.txt**

```
git rm --cached file3.txt
```

→ Stage **file3.txt** again

```
git add .
```

→ Commit the file to your repo

```
git commit -m "message"
```

→ Change the message of the commit

```
git commit --amend -m "new message"
```

→ Switch back to your first commit in [Task-1](#)

```
git log
```

```
git checkout "first commit ID"
```

Summary

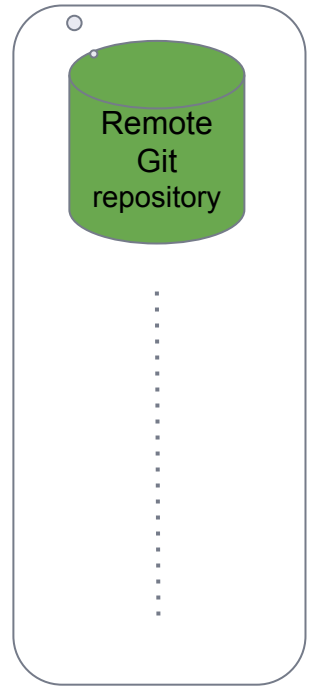
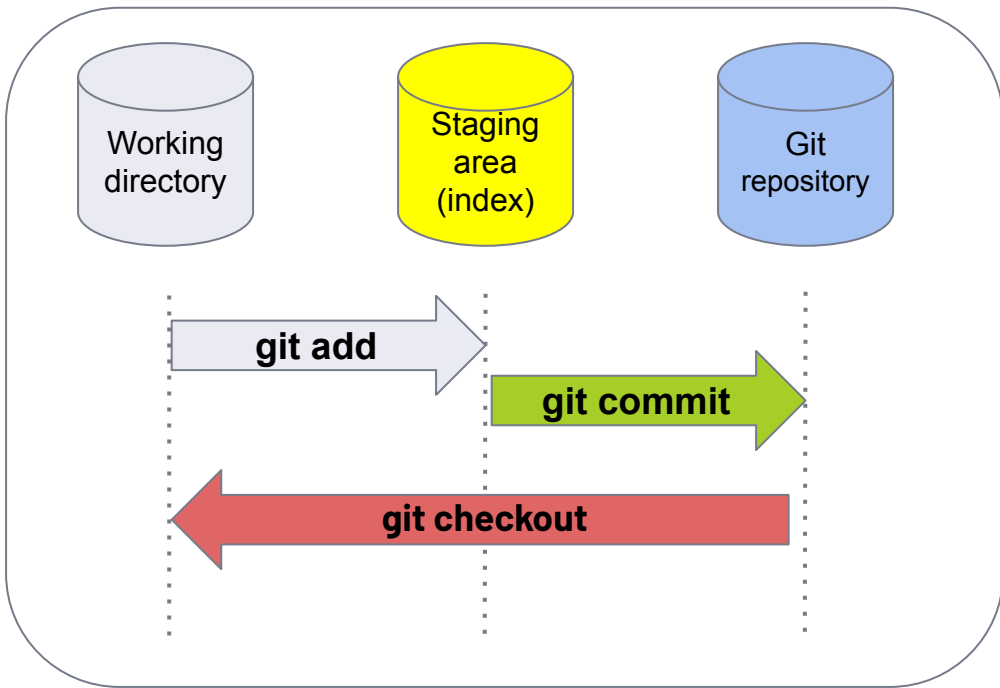


Summary

will talk about next session

GitHub

Local Machine



git init
git status
git add .
git commit -m "abc"
git log
git checkout



THANKS!

Any questions?

You can find me at:

- ▶ sumod@clarusway.com
- ▶ Slack - @sumod

