Team XCEED Welcomes All of YOU!!

xceed@nitj.ac.in

Dr. B R Ambedkar National Institute of Technology

Introduction to Git & GitHub

Dr. D. Harimurugan

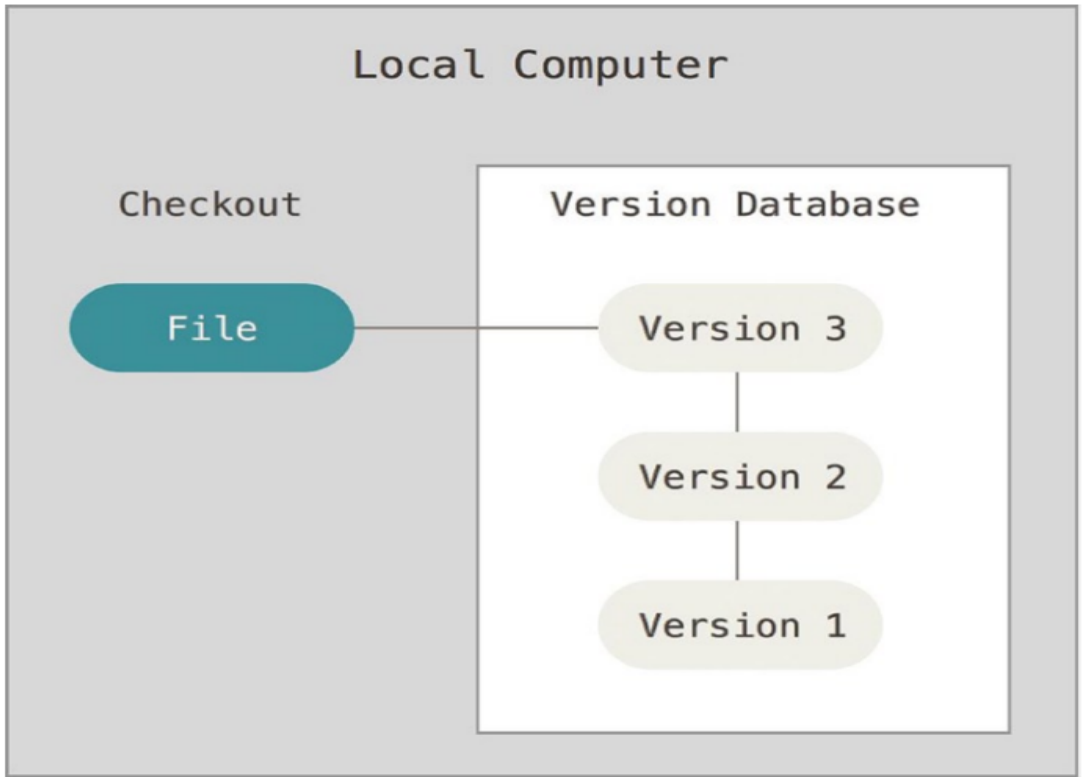
Course Contents

1. Introduction to Version Control
2. Getting Started with Git
3. Basic Git Operations
4. GitHub
5. Working with GitHub
6. Advanced Features

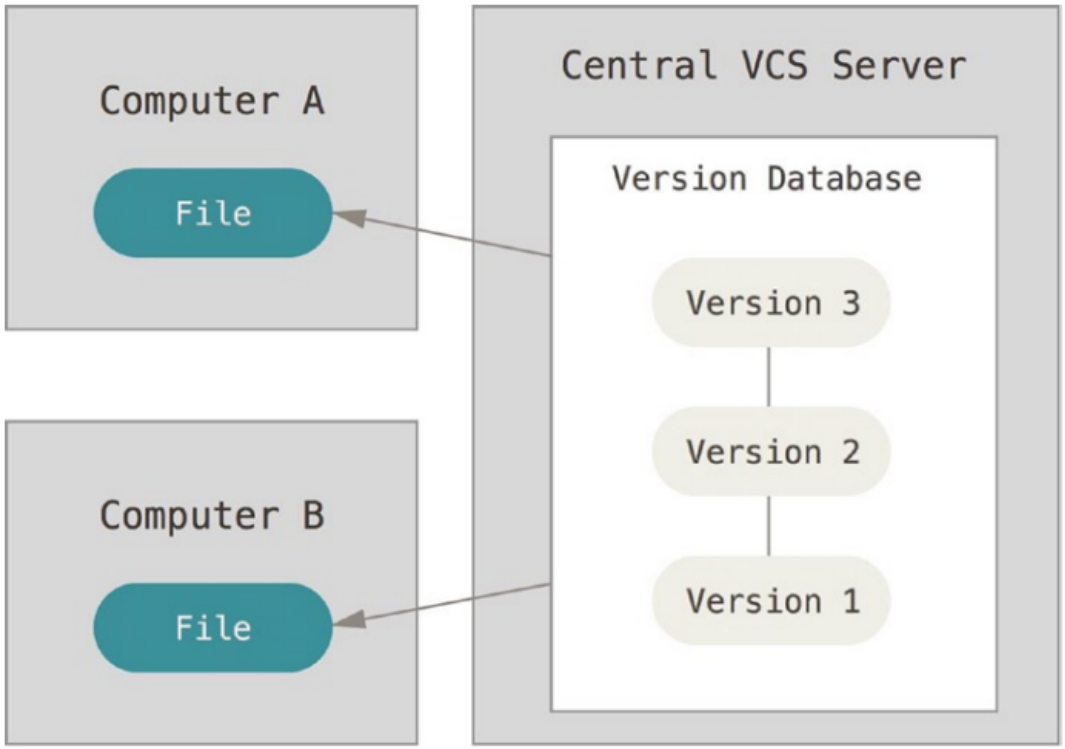
What is Version Control?

- A time machine for your code
- Track changes and history
- Collaborate without conflicts
- Real-world scenarios:

Types of Version Control Systems: Local Version Control Systems

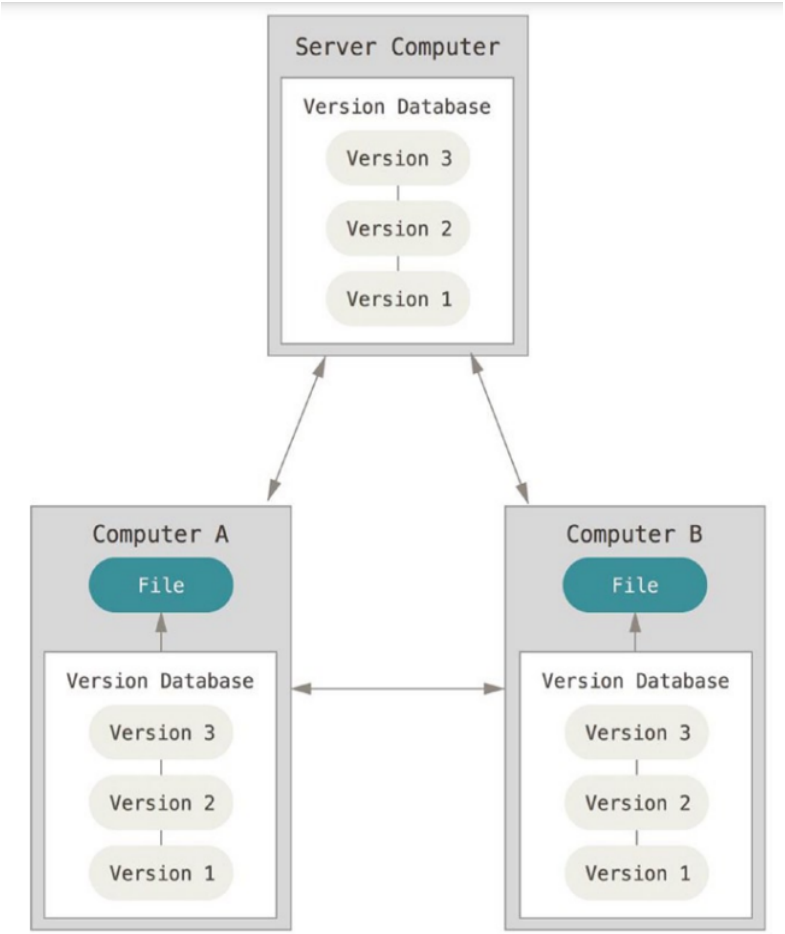


Types of VCS: Centralised Version Control Systems



- Single central server
- Network access required
- Limited offline work
- Simple but less flexible

Types of VCS: Distributed Version Control Systems



- Full local repository
- Work offline
- Multiple remotes
- More flexible workflow

Installation & Setup

Download and Install

- Windows: git-scm.com/download/win
- Mac: `brew install git`
- Linux: `sudo apt install git`

Essential Configuration

```
git config --global user.name "Your Name"
git config --global user.email "you@example.com"
git config --global core.editor "code --wait"
git config --list
```

Expected output in terminal

user.name=Your Name
user.email=you@example.com
core.editor=code −−wait

Short History of Git

- Git began with a bit of creative destruction and fiery controversy. Git was built in roughly 5 days!
- In 2002, the Linux kernel project began using a proprietary DVCS called BitKeeper
- In 2005, the relationship between the community that developed the Linux kernel and the commercial company that developed BitKeeper broke down, and the tool's free-of-charge status was revoked
- This prompted the Linux development community (and in particular Linus Torvalds, the creator of Linux) to develop their tool based on some of the lessons they learned while using BitKeeper

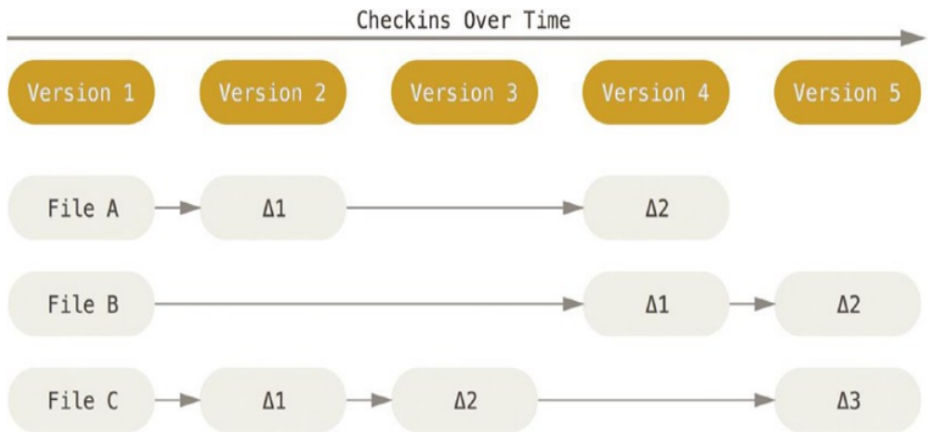
Features of New System: Git

- Speed
- Simple design
- Strong support for non-linear development (thousands of parallel branches)
- Fully distributed

How git is different?

Snapshots! Not Differences!

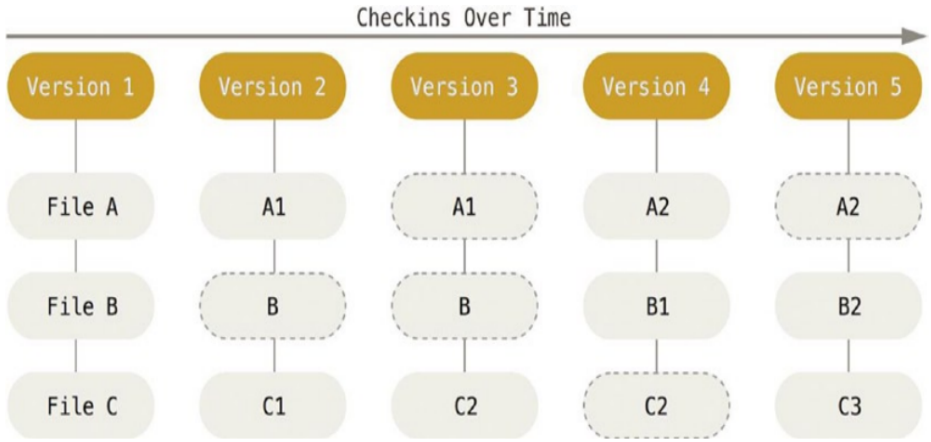
- The major difference between Git and any other VCS (Subversion and friends included) is the way Git thinks about its data.
- Conceptually, most other systems store information as a list of file-based changes
- These systems think of the information they keep as a set of files and the changes made to each file over time.



How git is different?

Snapshots of miniature filesystem

- Every time you commit or save the state of your project in Git, it basically takes a picture of what all your files look like at that moment and stores a reference to that snapshot.
- To be efficient, if files have not changed, Git doesn't store the file again, just a link to the previous identical file it has already stored
- Git thinks about its data more like a stream of snapshots.



How git is different?

Nearly every operation is local!

- Most operations in Git only need local files and resources to operate—generally, no information is needed from another computer on your network
- You have the entire history of the project right there on your local disk,and most operations seem almost instantaneous.
- To browse the history of the project, Git doesn't need to go out to the server to get the history and display it for you—it simply reads it directly from your local database
- If you get on an aeroplane or a train and want to do a little work, you can happily commit until you get to a network connection to upload

Git has Integrity

No loss of information: SHA-1 Hashing!

- SHA-1 (Secure Hash Algorithm 1) is a cryptographic hash function that Git uses to uniquely identify objects
- It produces a 40-character hexadecimal hash from an input.
- Git stores its content in blobs (Binary Large Object)
- A SHA-1 hash looks something like this: 24b9da6552252987aa493b52f8696cd6d3b00373
- You will see these hash values all over the place in Git because it uses them so much.
- In fact, Git stores everything in its database not by filename but by the hash value of its contents.

Basic Commands - Part 1

Task-1: Initialize Repository

- Create a new folder and navigate to this folder in VS Code and initialise using the code given below

```
git init
```

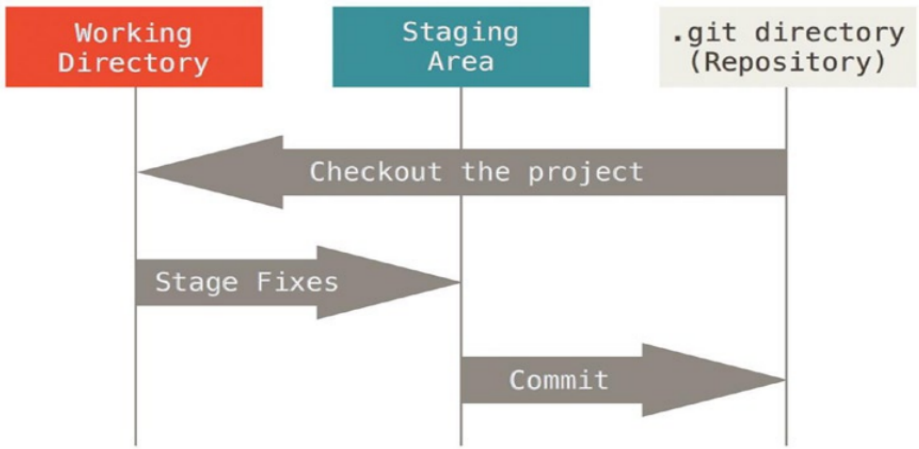
- The above command will create a git folder in your new folder (go to your folder and check its hidden folder)

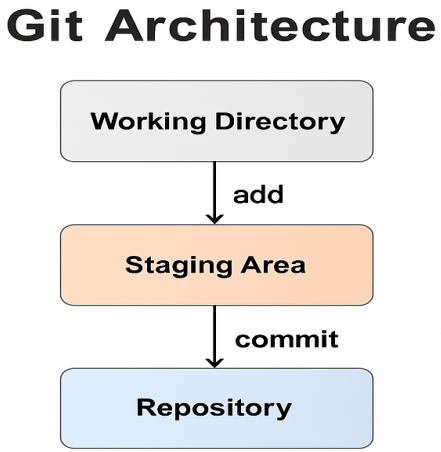
Check Status & History

For checking Current state: git status
For viewing history: git log

Three stages of git

Git uses **three-stage system** to manage changes, allowing you to control what gets saved in your project history





- **Working Directory:** Where you make changes (edit, create, or delete files)
- **Staging Area:** A "preview" zone where you select changes to save permanently
- **Repository:** Where finalised changes are permanently stored as commits.

First git operations

Task-2: Lets make a first commit

- Create a new file in the folder where we initialised the git.
- It can be any file (image, .txt or .html file). I am naming the file as hari.txt

```
git status
```

- The file you have added will be shown in red color, indicating git is not tracking the changes in it.

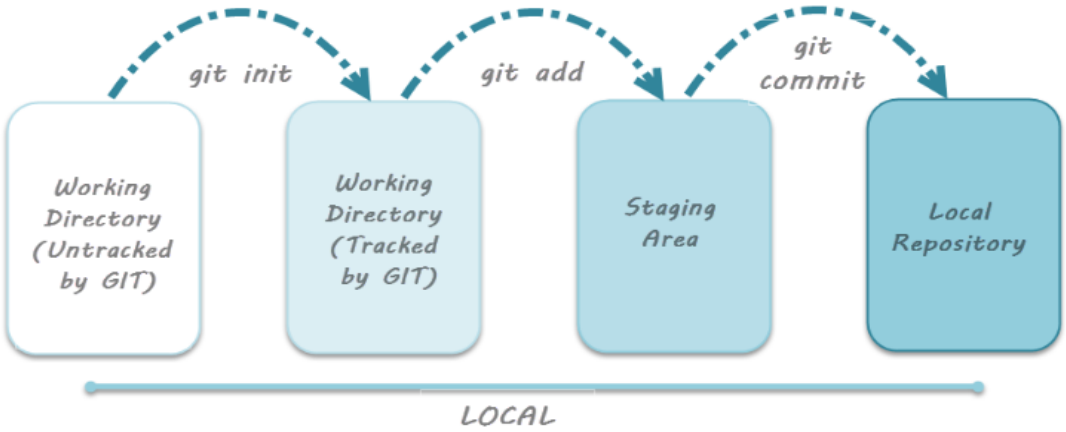
```
git add hari.txt
```

- Check the status, now file is stagged and ready to be committed.

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

Congratulations! You have made a first commit! Check the log.

Quick Summary

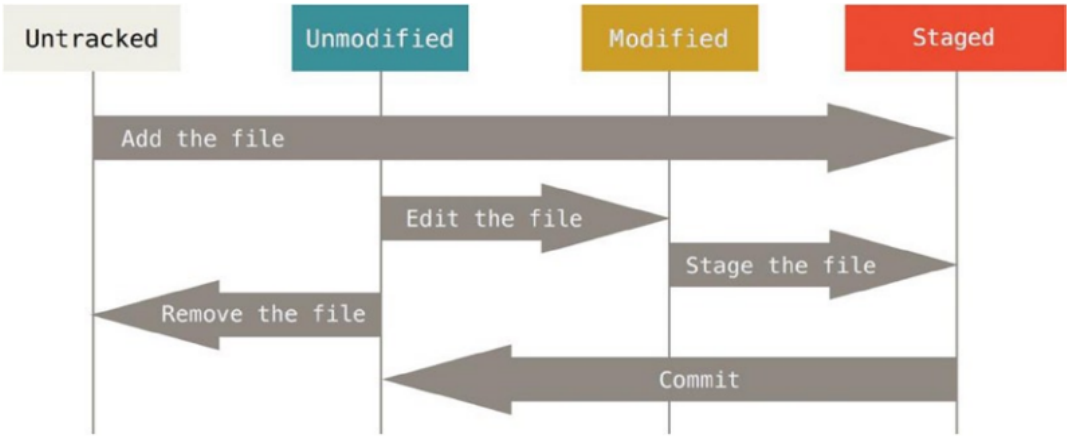


Task-3: Make your second commit

- Add some content to the file you have already added and commit it.
- view the log! Don't forget to add a commit message

Life cycle of git files

- **Untracked:** File is new and not tracked by Git.
- **Tracked:** File has been added to Git, which includes several sub-states:
  - **Unmodified:** No changes since last commit.
  - **Modified:** Changes made but not staged.
  - **Staged:** Changes added to the staging area, ready to be committed.
- once the commit is made, all files are changed to unmodified



Unstage and Untrack

Unstage and untrack

- Let's say you want to remove a file from staged state but want to keep the changes in the file

git restore −−staged $< filename >$

- To untrack a file in Git (remove it from version control while keeping it in the working directory)

git rm −−cached $< filename >$

Comparison of files

Comparison between the stages

- Compare the changes between "Working directory" Vs "Staging area"

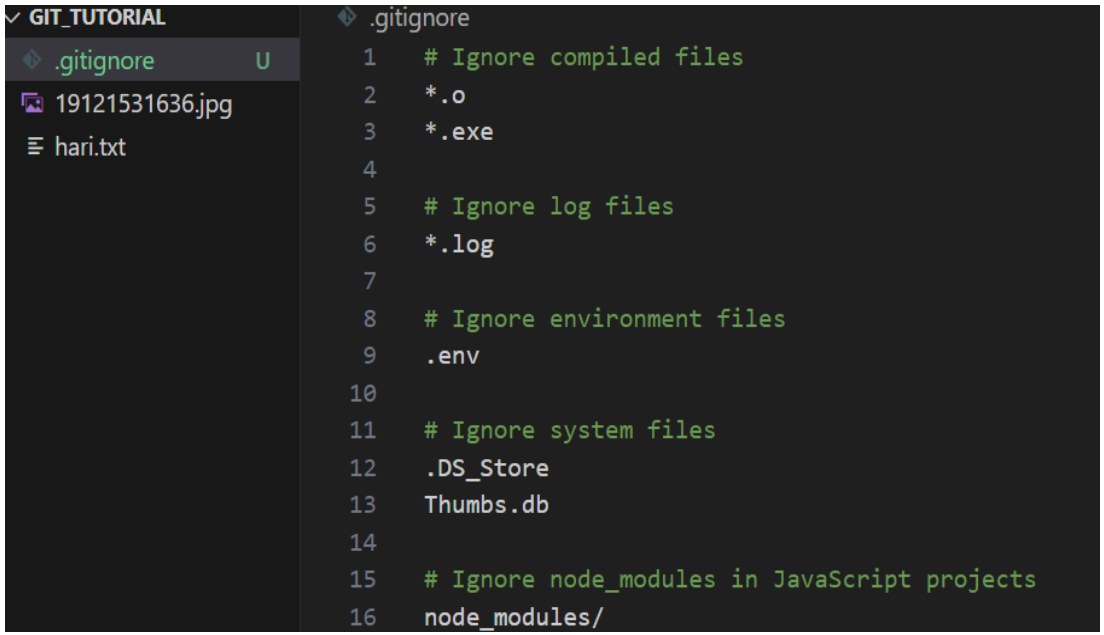
```
git diff
```

- Compare the changes between "staging area" Vs "last commit"

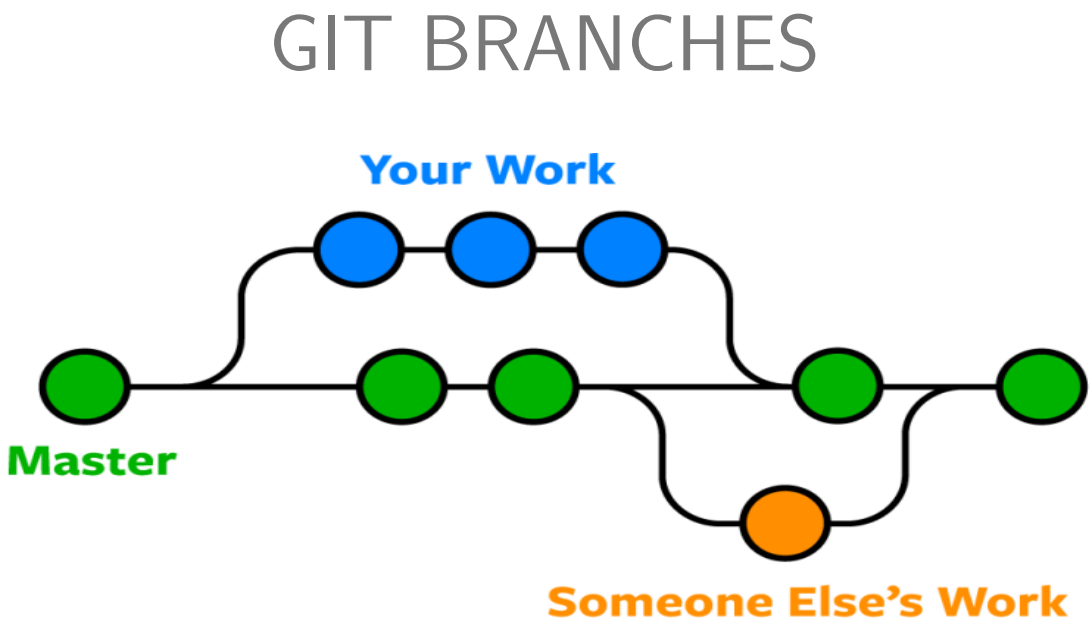
```
git diff --staged
```

GitIgnore

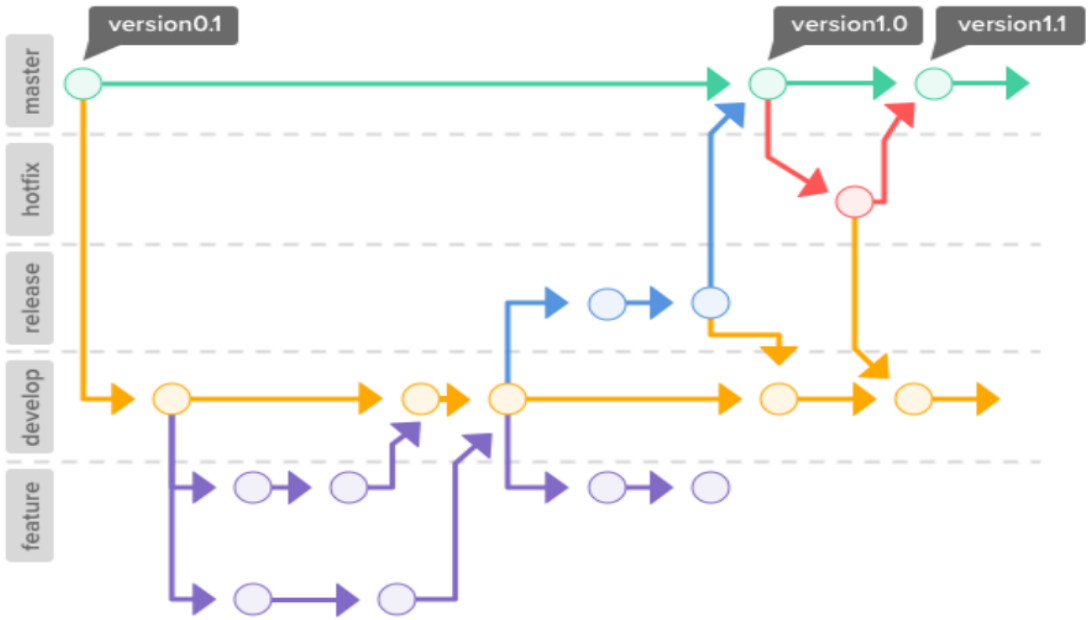
- Add a file with the name .gitignore in your repo
- A .gitignore file tells Git which files or directories to ignore in a repository. This means Git will not track, stage, or commit those files.
- Sample .gitignore file



git Branches



git Branches



Git Branches

- A branch in Git is like a separate path of development. It allows you to work on new features, bug fixes, or experiments without affecting the main code (usually the main or master branch)
  - **Work Independently** – Develop features without affecting the main code.
  - **Collaboration** – Multiple developers can work on different features.
  - **Bug Fixes & Testing** – Test new ideas without breaking the project.
  - **Safe & Organized Code** – Keep your main branch clean and stable.

Check Current Branch

- To show all branches in your repository.

```
git branch
```

- The * marks your current branch.

Creating a new branch

Create new branch and switch to it

- To create a new branch in your repository

```
git branch feature-branch
```

- Above command will create a new branch, but you are still in the master branch
- Switch to the new branch using the following command

```
git checkout feature-branch
```

- There is a shortcut to create and switch to a new branch in a single line

```
git checkout -b feature-branch
```

Creating a new branch

Some interesting points about branching

- Git does not allow duplicate branch names in the same repository.
- If you need a similar branch, modify the name or delete the old one first.
- You can create a new branch from any existing branch, not just from main.
  - If you create a new branch while on **main**, the new branch will start from the latest **main** commit.
  - If you create a new branch while on **feature-A**, the new branch will start from the latest commit of **feature-A**.
  - This allows you to chain feature branches without merging everything into main first.
  - **The new branch will inherit all commits from the branch you are on**

Task 4: Make some commits in a new branch

- Add some additional lines to the file in the feature branch you have created
- Stage and commit the changes.
- Add a new file and commit in this branch

Merging with master branch

- After making commits in the feature branch, the following steps are to be followed

Merging with master branch

- Go to master branch

```
git checkout master
```

- Merge the commits from the feature-branch to the master branch

```
git merge feature-branch
```

- Delete the feature-branch after merging if required

```
git branch -d feature-branch
```

The commits you have made in feature branch will be shown in the log in the master branch after successful merging

Commit Graphs

Command-Line Visualizations

- Visual representation of commits.
- Merges show integration of different branches.

```
git log --graph
```

- Compact representation

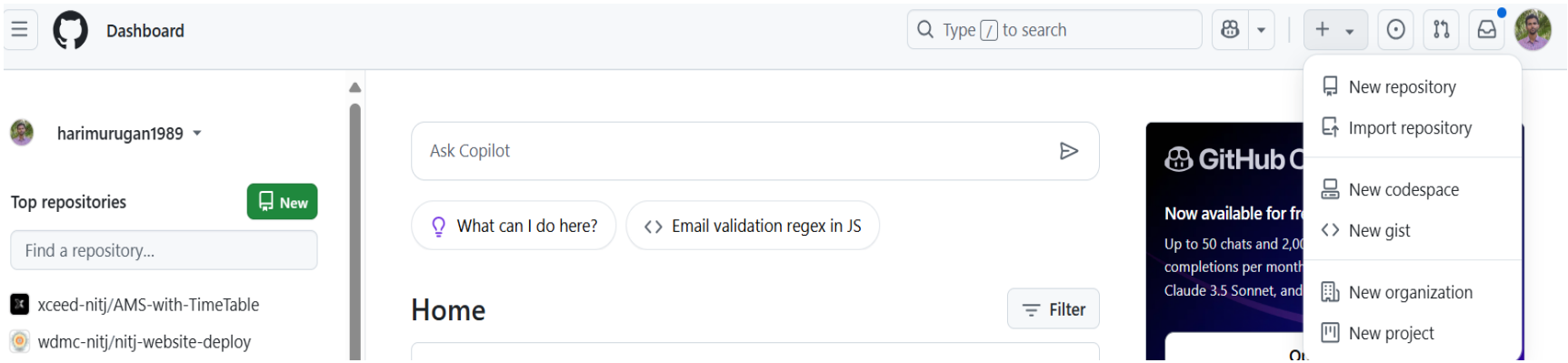
```
git log --oneline --decorate --graph --all
```

Creating your first repo in gitHub

- GitHub is a platform that hosts remote Git repositories in the cloud.
- It lets you
  - Store code online
  - Collaborate with others
  - Track changes visually
  - Share projects

Task-5: Creating a repo on GitHub

In GitHub, click on + (at the top) to create a new repository



Creating your first repo in gitHub

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?
Import a repository.

Required fields are marked with an asterisk (*).

Owner * harimurugan1989 / **Repository name *** FE_demo_repo

FE_demo_repo is available.

Great repository names are short and memorable. Need inspiration? How about **reimagined-spork** ?

Description (optional)

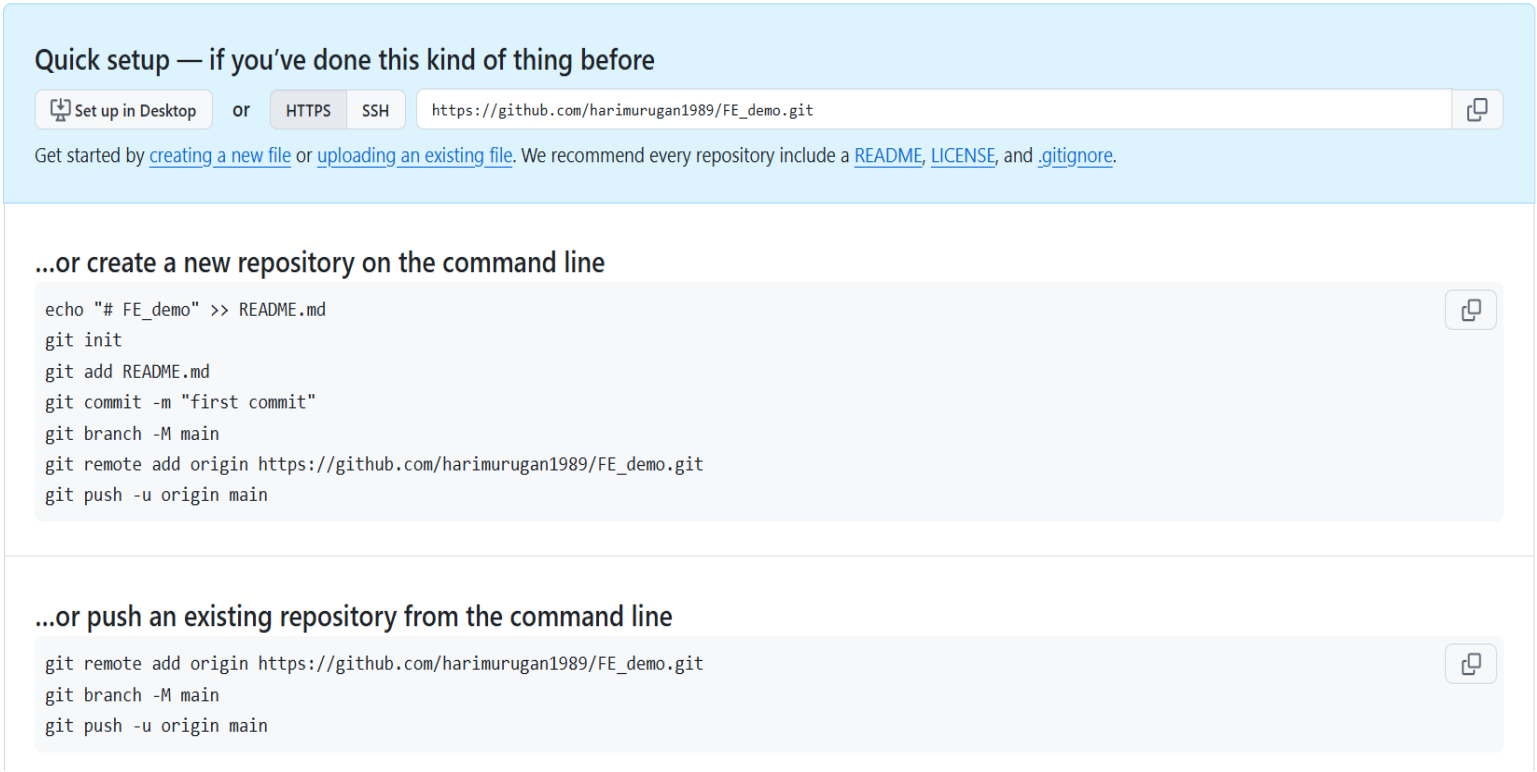
Sample description

- Public
  Anyone on the internet can see this repository. You choose who can commit.
- Private
  You choose who can see and commit to this repository.

Initialize this repository with:

- [ ] Add a README file
  This is where you can write a long description for your project. Learn more about READMEs.

Creating your first repo in gitHub



Quick setup — if you've done this kind of thing before

Set up in Desktop or HTTPS SSH https://github.com/harimurugan1989/FE_demo.git

Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.

...or create a new repository on the command line

```
echo "# FE_demo" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/harimurugan1989/FE_demo.git
git push -u origin main
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/harimurugan1989/FE_demo.git
git branch -M main
git push -u origin main
```

Clone a repo from GitHub

```
git clone https://github.com/harimurugan1989/FE_demo.git
```

Remote Operations

Basic Remote Commands

- To upload the committed changes to the remote repo

```
git push origin main
```

- If pushing a new branch for the first time:

```
git push --set-upstream origin feature-branch
```

- To pull changes from remote

```
git pull origin main
```

List of Dangerous Git Commands

Command	Explanation
`git reset --hard`	Deletes all uncommitted changes permanently.
`git clean -df`	Removes untracked files and directories.
`git push --force`	Overwrites remote history, erasing commits.
`git rebase -i HEAD N`	Modifies commit history, causing potential loss.
`git branch -D <branch>`	Permanently deletes a branch, including unmerged changes.

GitHub Workflow

This workflow ensures collaborative and structured development in open-source projects.

1. Fork repository (on GitHub)
2. Clone your fork
3. Create feature branch
4. Make changes
5. Push changes
6. Create Pull Request
7. Review & merge

You don't have to be great to start, but you have to start to be great!!