# 5-Days Workshop on Git and GitHub

## Day 1 - Introduction to Git

19th February 2022

By,

**Microsoft Learn Student Ambassador Community**

## Code of Conduct

- Be respectful of different viewpoints and experiences.
- Please mute your mic during the entire session unless requested to unmute.
- If you feel any doubts in the middle of session, feel free to raise your hand or drop your doubts on the chatbox.
- It's not mandatory but if possible turn your camera at the end of session when requested to take some snapshots of this workshop.

## Guides to Help You

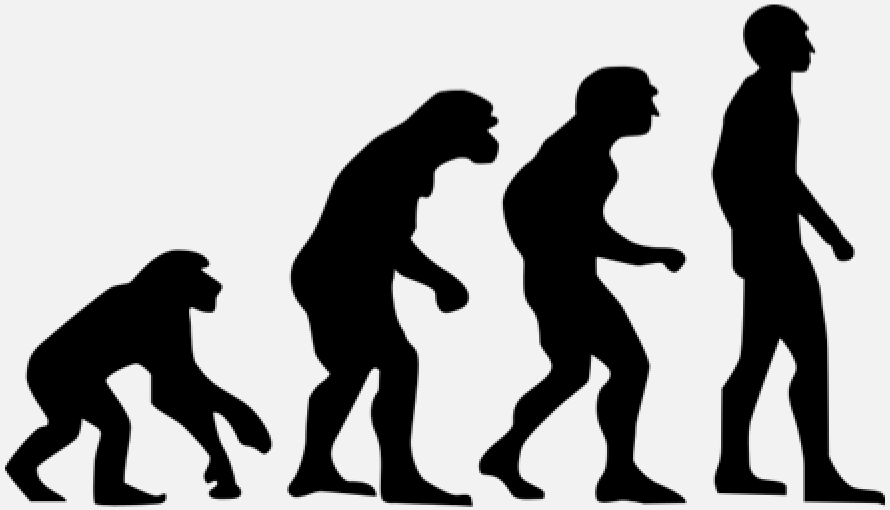
- The workshop will be conducted every evening from 5:30 pm to 7:15 pm.
- Each day you will receive a Daily Check-In Form after 30 minutes of starting of workshop.
- Also, each day you will be given a Challenge that you must complete within 7 days to be eligible for the perks.
- All of our links are live i.e. you can click on link in the screen to visit the URL. Try out this link.
- Be sure to keep checking our repo for awesome resources.

## About Me …

- a **Nepali**,
- a Undergrad Computer Engineering student,
- a **Microsoft Learn Student Ambassador**,
- a tech enthusiast,
- an open source admirer,
- and I love Open-Sans font.

## What will we do today?

- Introduction to VCS,
- Introduction to Git,
- Basic Git commands,
- Git tags,
- Undoing changes,
- Rollbacks,
- **Lab:** Playing with Git in local



## What is Version & VCS ?

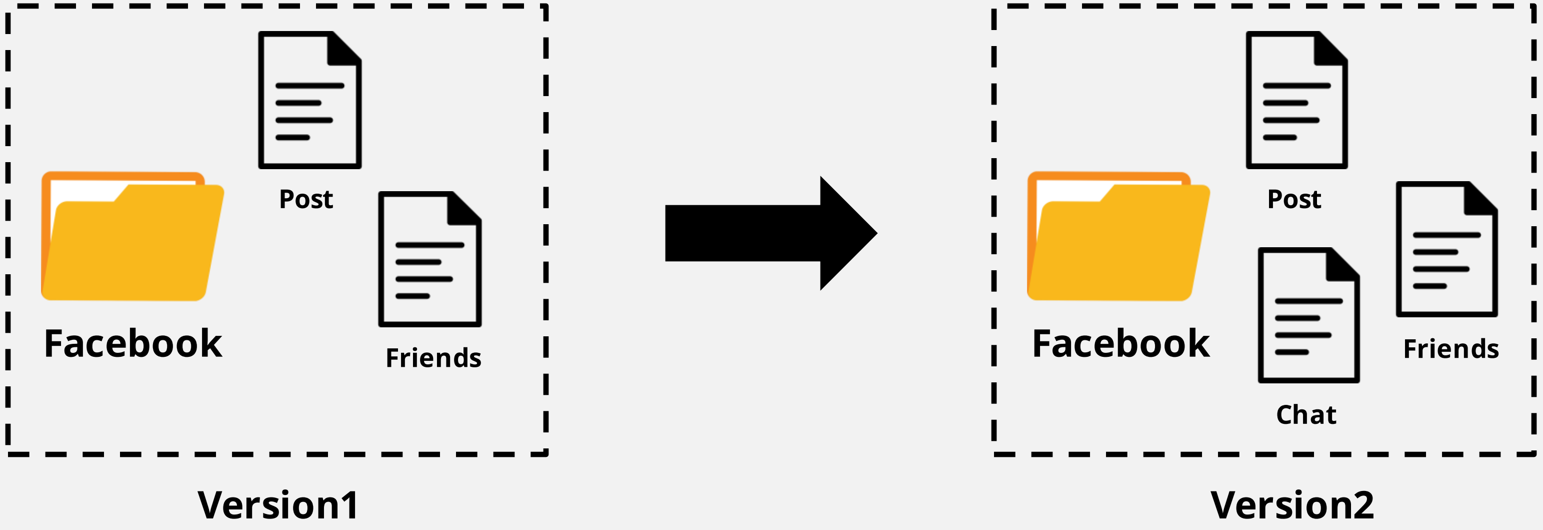
## Let's visualize VERSIONING

Version 1 (Initial Version)

Version 2 (Final Version)

# About VERSION

- Version is defined as **the state of project** you are working on.
- To be noted that slight change in a project changes the version of the entire project.

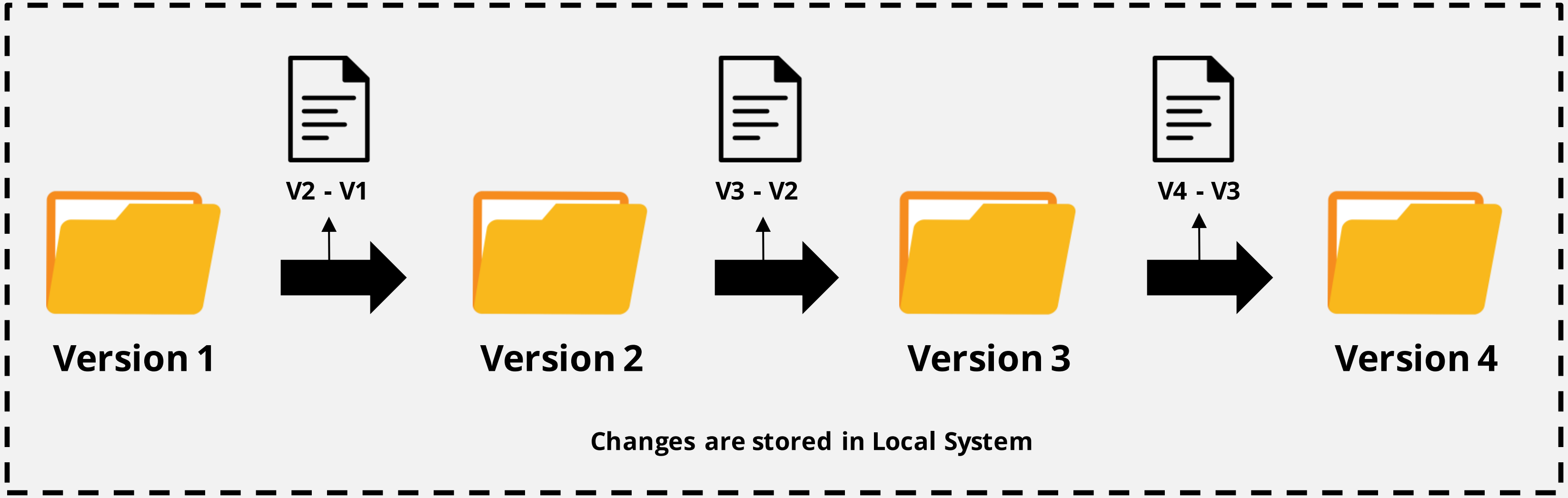


## What is VCS ?

- VCS or **Version Control System** manages the entire process of version and **keeps the track of the changes made**.
- We need VCS to;
    - **track** the changes made,
    - **revert** back to any versions when needed,
    - storage **efficient** and keeps project clean,
    - **compare** the changes made between any two versions,
    - to know **information** like who, when, why made the changes,
    - to get all these benefits with **minimal effort**.

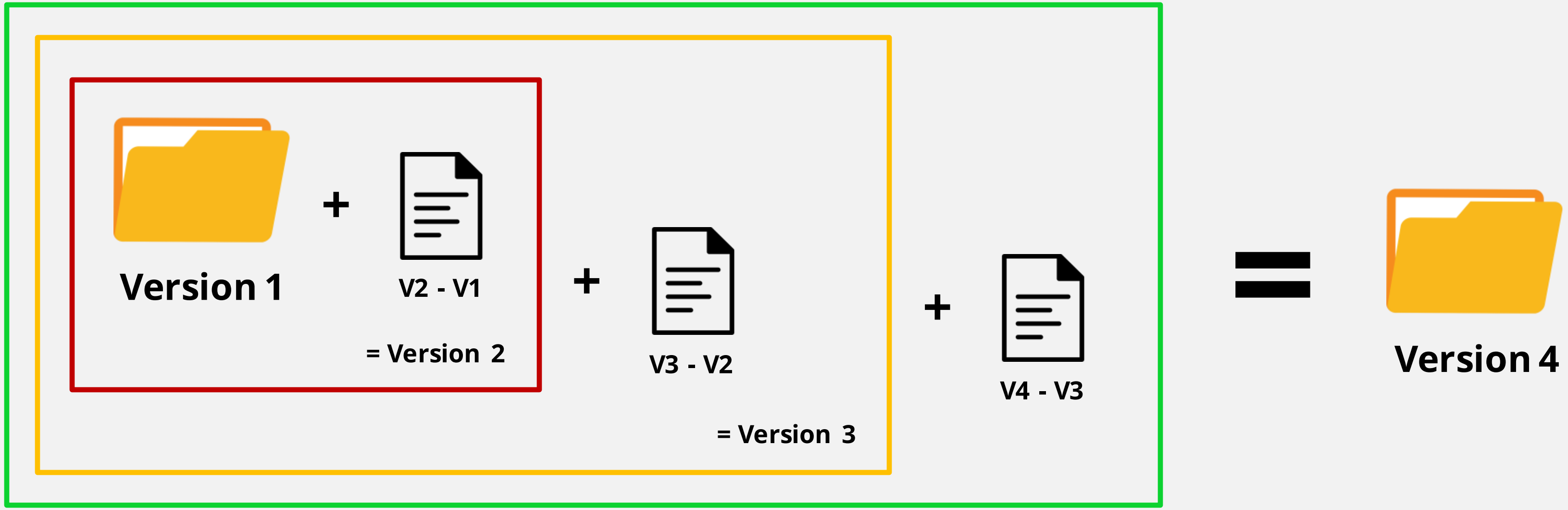
## Types of VCS

- **Localized VCS;**
  - Changes are stored in Local database,
  - It can use the changes history to recreate any version.



## Types of VCS

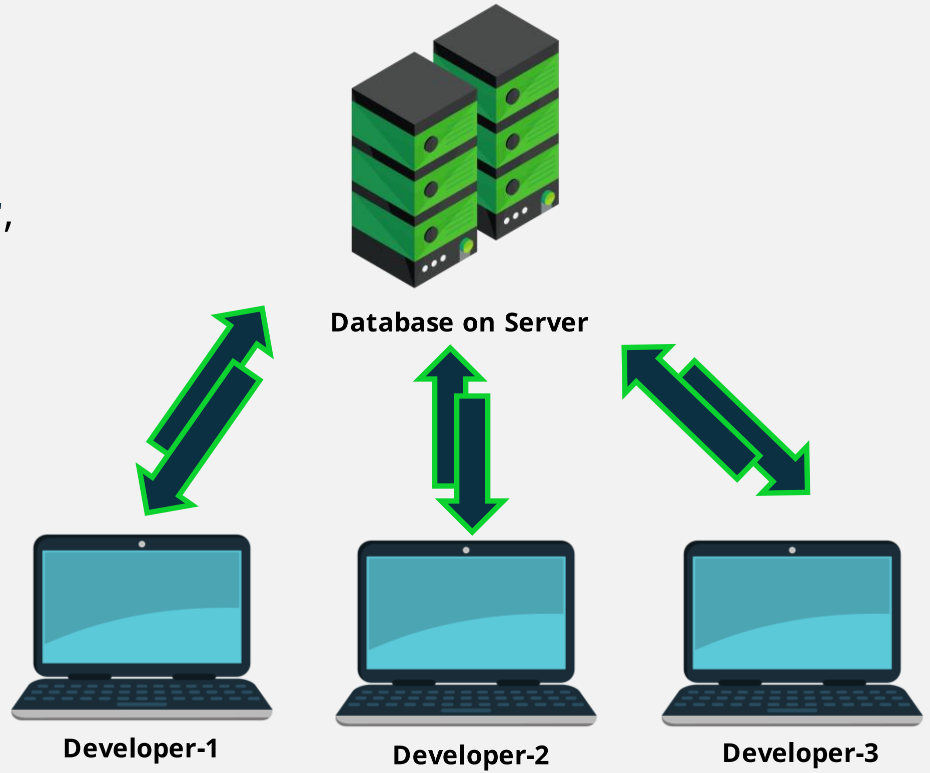
- So, to **re-create Version4** what we do is;



This is an example of **Delta-Based VCS** while there are many other methods of implementing VCS but it is very simple to understand and implement.

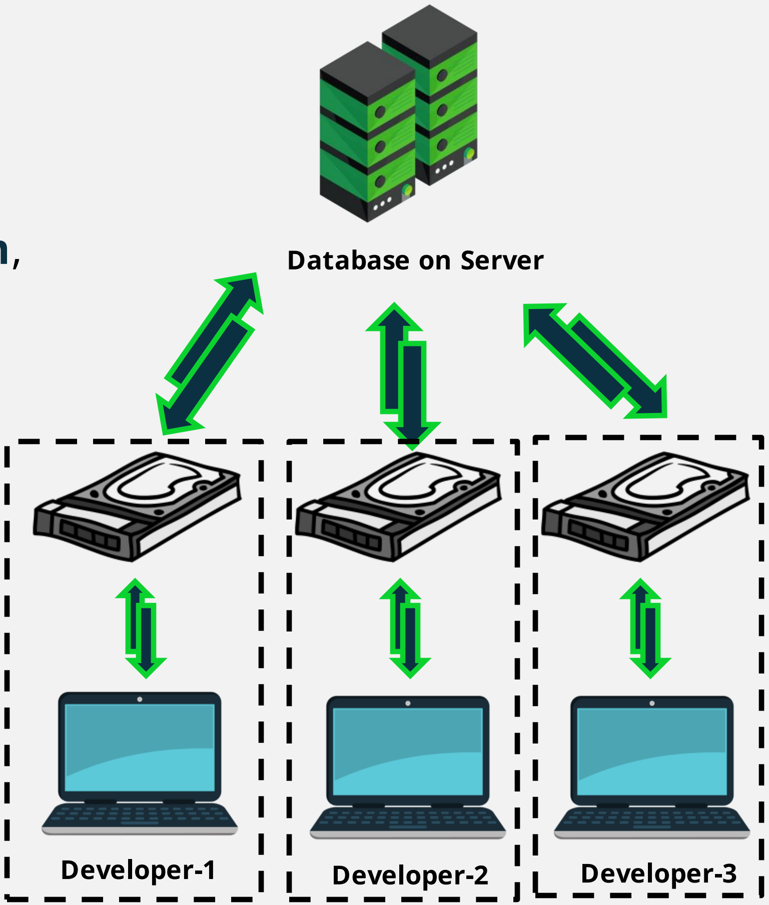
## Types of VCS

- Centralized VCS;
  - Changes are **stored in a server**,
  - Collaboration becomes easier,
  - Depends on network latency,
  - **Single point of failure**.



## Types of VCS

- Distributed VCS;
- Changes are **stored also in local system**,
- All benefits of Centralized VCS,
- Doesn't depends on network latency,
- Better bug prevention,
- **Easy to restore**.



## Check-In Alert

Proceed to the link, bit.ly/mlsa-git-checkIn with today's check-In code:

**"AwesomeGit"**

## Let's talk about Git

## Introduction to Git

- Git is a Distributed VCS,
- It was developed by **Linus Torvalds** (main developer of Linux Kernel) in 2005 A.D.
- Git holds 70% of share in VCS market,
- It is the most powerful and widely used VCS,
- It is open-source, fast, versatile, and highly scalable.

- **Note that: Git & GitHub are not same. Git is a VCS but GitHub is a repository hosting platform that uses git in its core.**

## Short History of Git

- Linus was using **BitKeeper** as VCS for Linux Kernel,
- BitKeeper was **proprietary software**, but free for Linux Developers,
- But, things didn't went right and it was made completely proprietary for everyone.
- Linux Kernel had thousands of developer,
- So, Linus developed his own VCS called Git.

## Short History of Git

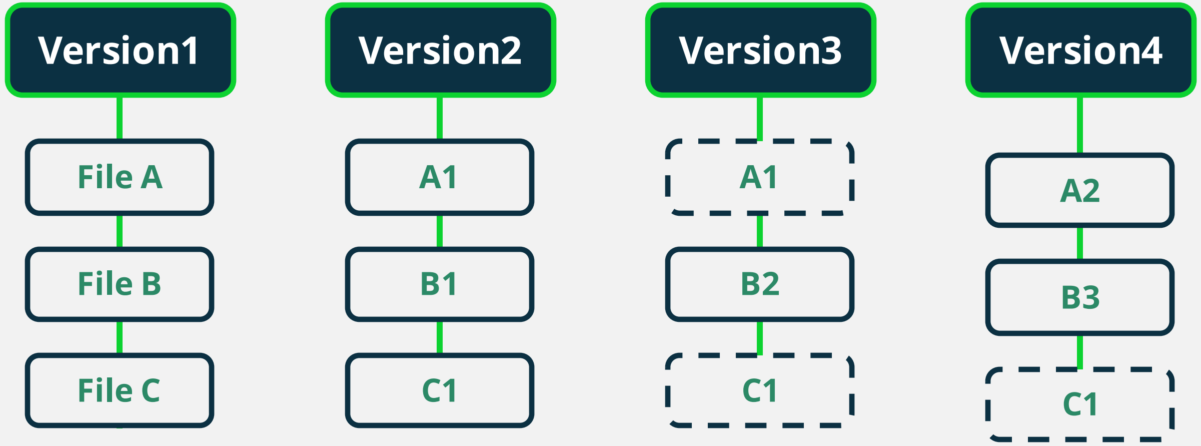
- The main objectives of Git were;
  - Maintaining changes should take **less than 3 seconds**,
  - Should support **distributed workflow**,
  - Should maintain **integrity** of code,
  - Should support parallel **branches**,
  - Should be able to handle **large codebases** without compromisation.

- He often used the term **"Stupid Content Tracker"** for Git.

## Why only Git?

## Use of Snapshots

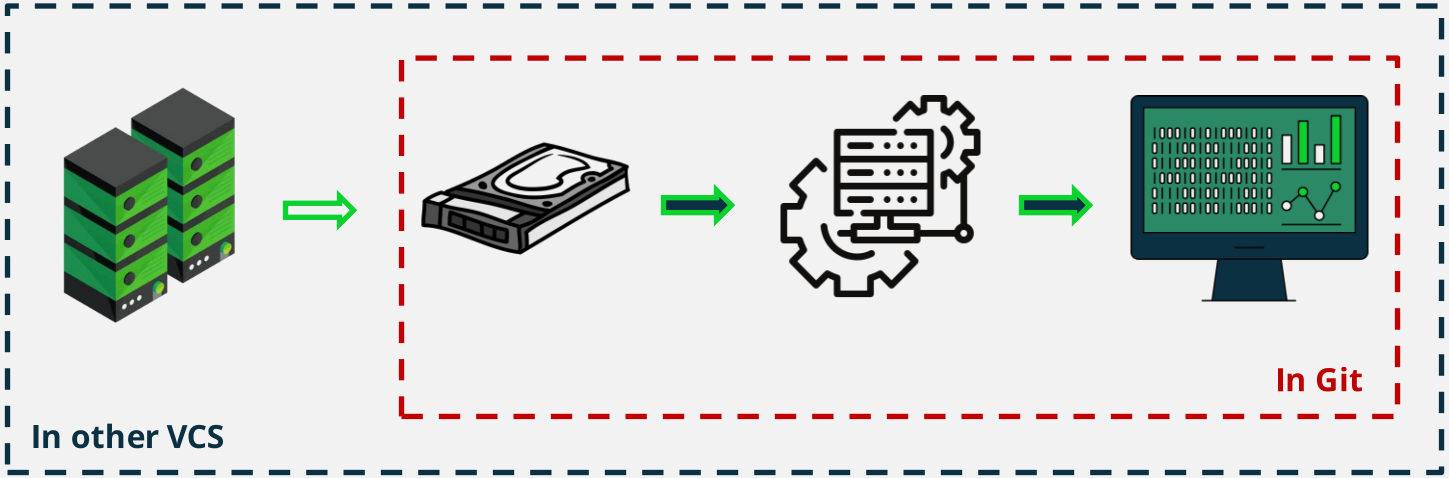
- Git doesn't stores the changes as done by Delta-based VCS.
- When you commit, Git takes a **snapshot** of what all our file looks like and stores a reference to the snapshot.



**This is how snapshots works**

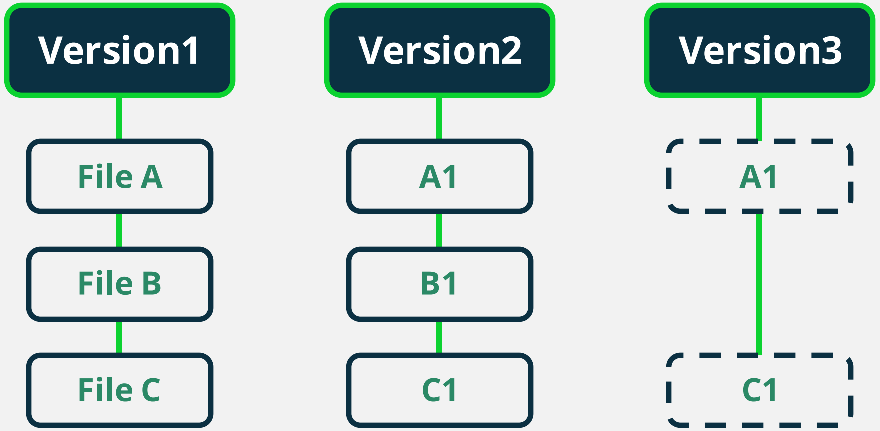
## Every Operation is Local

- Every operation in Git is done within your system. So,
  - Operation is instantaneous.



## Git Only Adds Data

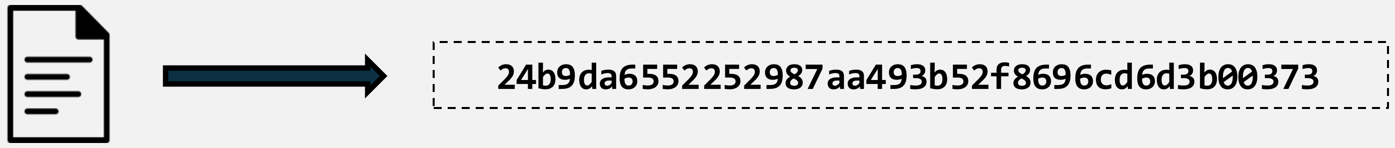
- It makes most of the process recoverable.



This is how deletion works in GIt

## Git Has Integrity

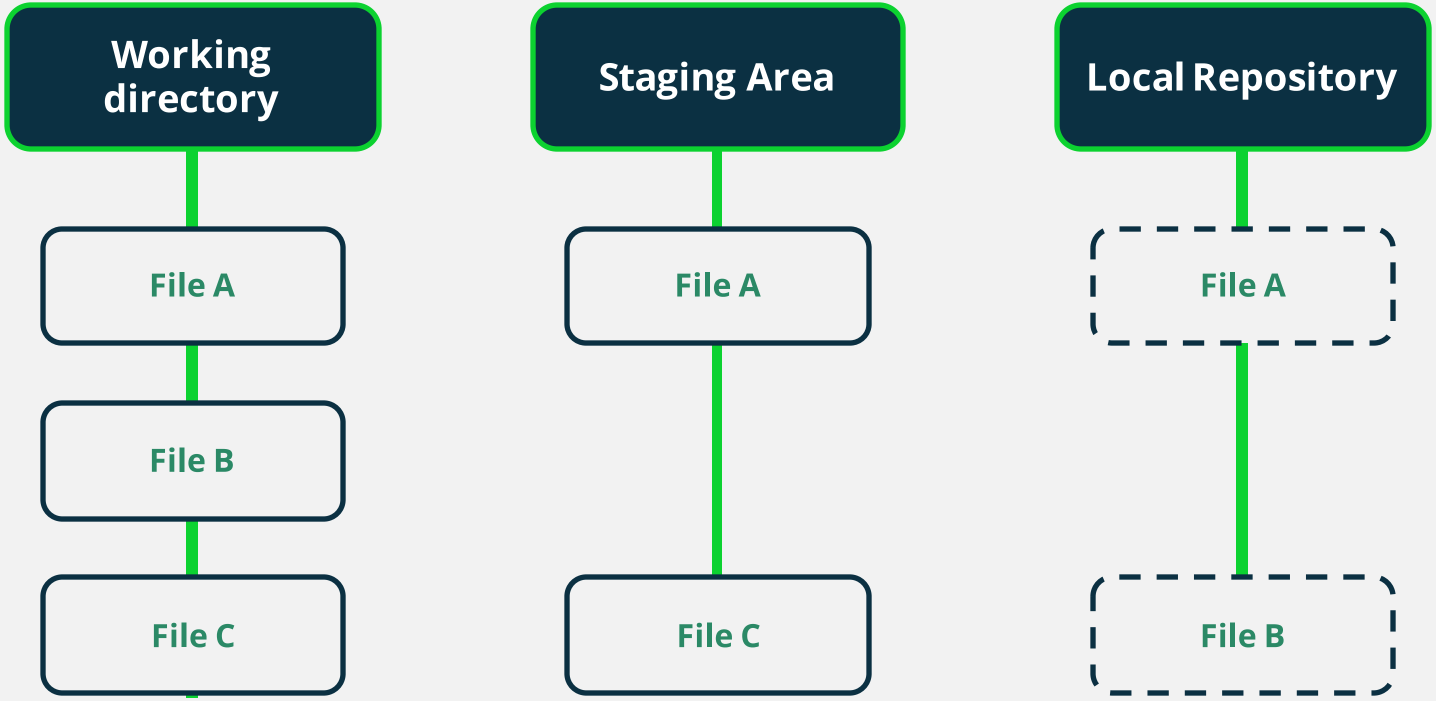
- Everything in Git is checksummed before stored.
- So, it is impossible to change data without Git knowing about it.
- File corruption chances reduces.
- It uses SHA1-hash mechanism for checksumming which generates a 40 Hexadecimals character.



## Let's Recall

- Version is the state of project,
- VCS is used to track the changes,
- There are 3 types of VCS; local, centralized and distributed,
- Git is distributed VCS,
- Benefits of Git over other VCS are;
  - Uses snapshots,
  - Every operation is local,
  - Git generally only adds data,
  - Git has integrity.

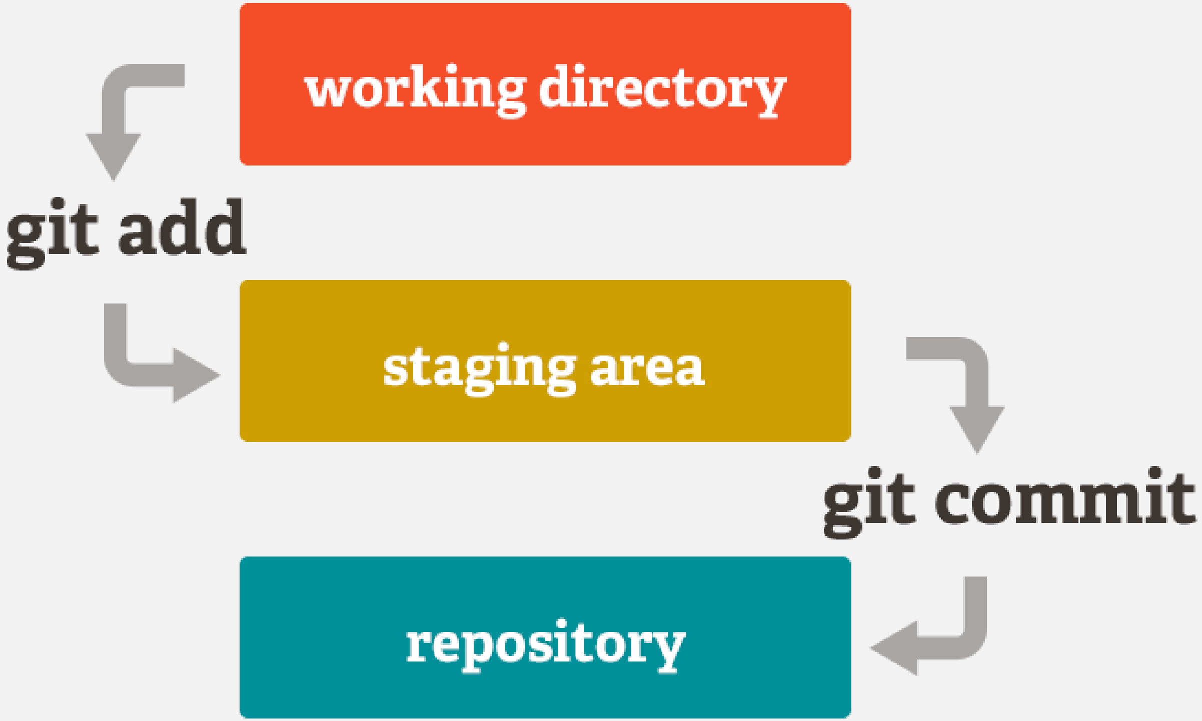
# The Three Git Sections



## The Three Git Sections

- Working directory:
  - This is the place where you work.
- Staging area (index):
  - A place where you can stage or prepare what files should appear in next commit.
- Local repository:
  - A place where all Git Objects are stored.
  - It is .git folder inside Git repository.

## Git Workflow Explained



## Theory Part Ends Here

It's time to have some fun !!

Be ready to

Kahoot!

Supported by:

Sikshya Technology

## Creating Your Identity

- Let's see what levels are in Git;

  - **--global:** The configuration you made is for every repository you create in your system by the user.
  - **--local :** The configuration you made is for only the repository you are working on.
  - **--system:** The configuration you made is for every repository you create in your system.

## Basic Git Commands

- **`git init`** :- It initializes the filesystem as an empty git repository.
  - It creates a folder **.git/** which holds subdirectories for Git objects.
  - It creates a branch named **master** and a **HEAD** file on that branch.

- **`git status`** :- It displays the state of working tree and staging area.
  - It lets you to know what is being tracked and what is not.
  - It helps to predict the structure of next snapshot.

- **`git add`** :- It is the command that tells Git to start to track a file.
  - It is used to stage the changes to prepare for next commit.

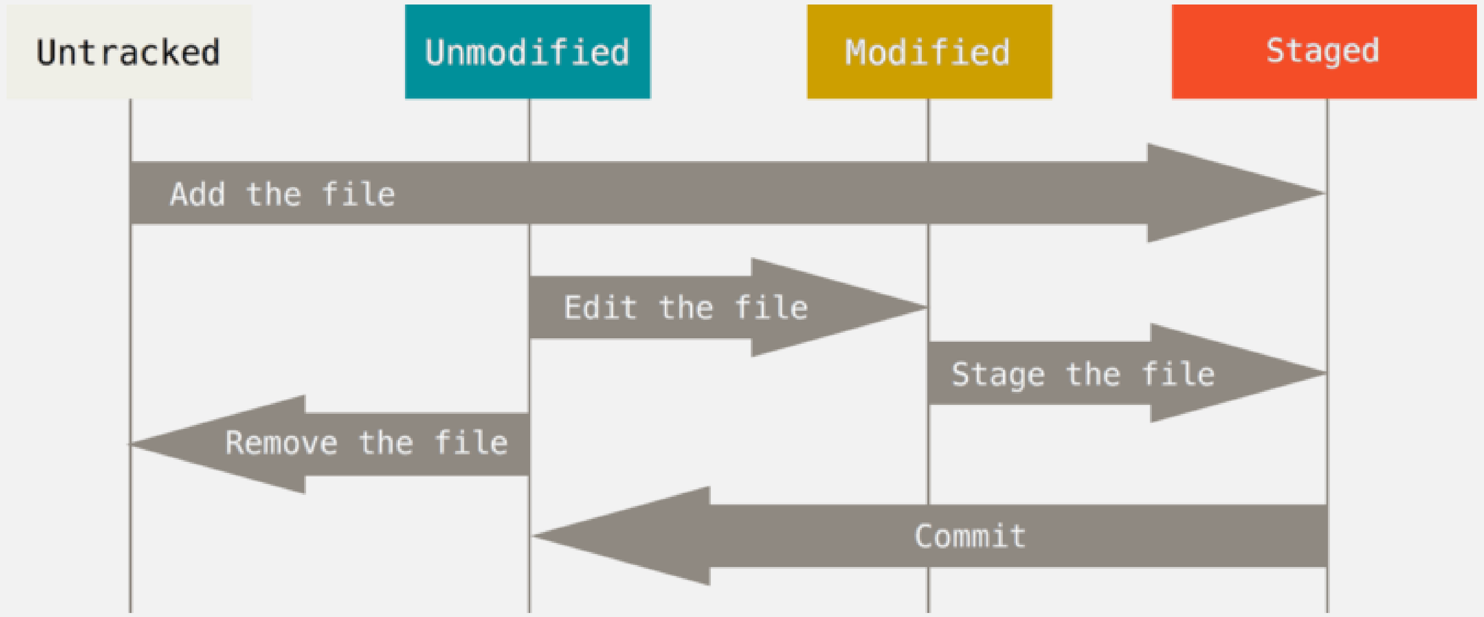
- **`git log`** :- It is used to see the commit history of a repo.

## State of Files in a Git Repo

- **Untracked files (U):** These files are not being tracked by Git.
  - To track a untracked file we use command ***git add <filename>***.

- **Tracked files (A):** These files are being monitored by Git.
  - Staged files: These files have been staged and will appear in next commit.
  - Modified files **(M)**: These files have been modified since last snapshot.
  - Committed files: These files have not been modified since last snapshot.

## State of Files in a Git Repo



## Any Questions?

## Creating Your Identity

- **Your name and e-mail**
  - `git config <level> user.name "<your user name here>"`
  - `git config <level> user.email "<your email here>"`

- **Your default text-editor**
  - `git config <level> core.editor "<your editor>"`

- **Your default branch name**
  - `git config <level> init.defaultBranch "<default branch name here>"`

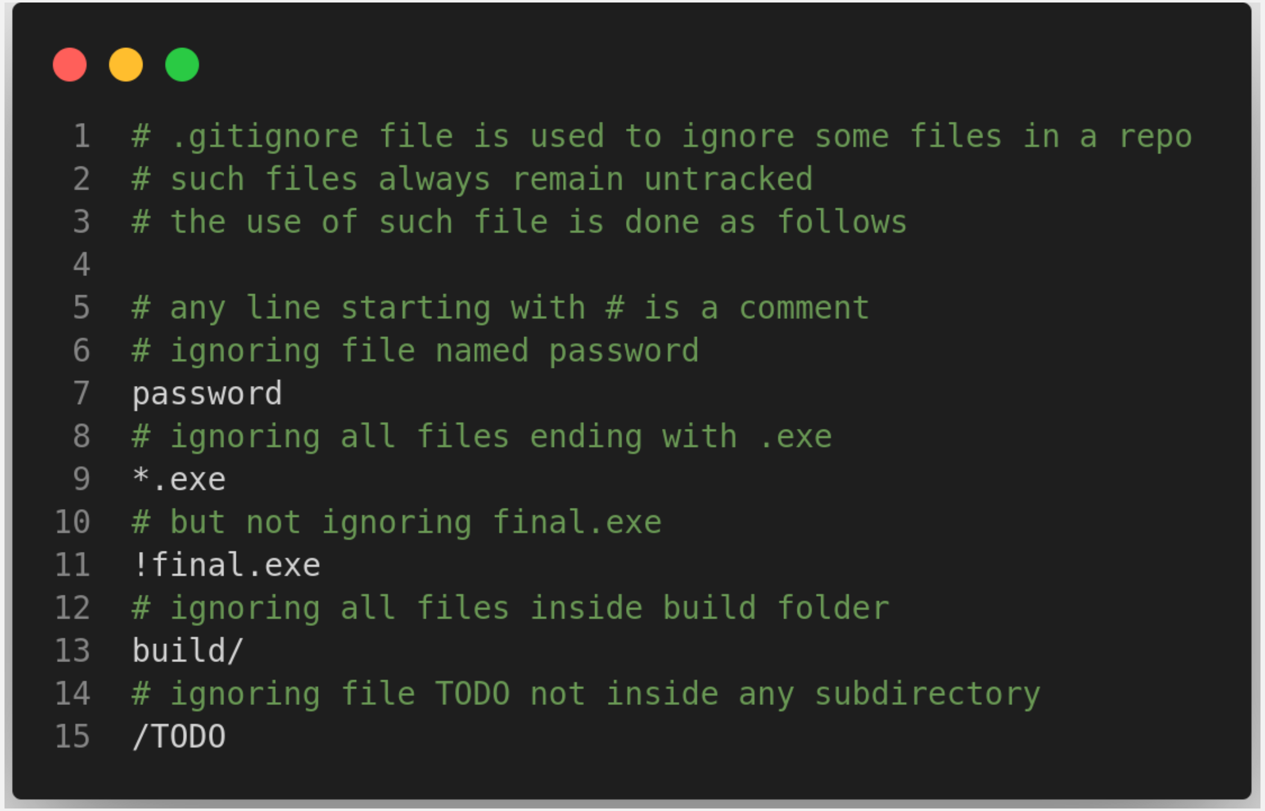
- **Checking your Git configs**
  - `git config <level> --list`

## Tips for the Pros

- We have a command **`git status –s`** to view status of Git repo in short.
- You can track and stage all files at one with help of command **`git add .`**
- Files except mentioned in .gitignore is staged.
- You can skip staging file before commiting by command

  ```
  git commit –a –m "your commit message here"
  ```
- Git does not consider a empty directory in any commit so we use **.gitkeep** or **.keep** file inside any empty directory.

## Ignoring Files *(.gitignore)*



```
# .gitignore file is used to ignore some files in a repo
# such files always remain untracked
# the use of such file is done as follows

# any line starting with # is a comment
# ignoring file named password
password
# ignoring all files ending with .exe
*.exe
# but not ignoring final.exe
!final.exe
# ignoring all files inside build folder
build/
# ignoring file TODO not inside any subdirectory
/TODO
```

## Tips for the Pros

- You can use **`git log --oneline`** to view the commit history in short,
- You can use **`git log --graph`** to view the commit history in graphs,
- You can use **`git log --patch`** to view the changes or patches made in each commit,
- You can use **`git log --stat`** to view the short statistics of the changes made,
- You can view last nth commits by **`git log -n`** E.g. `git log -2` shows last two commits made,
- More on the Git cheatsheet about Git logs.

## Renaming or Moving a File

- Renaming and Moving a file is same operation in Git.
- It can be done by command **`git mv file_from file_to`**
- Which renames the *file_from* to *file_to* or moves the file from *file_from* to *file_to*.
- The command is equivalent to;

```
mv file_from file_to
git rm file_from
git add file_to
```

## Removing a File

- If we delete a file normally on a repo we have to stage the changes again,
- But we can use **`git rm <filename>`** to delete the file from both working directory and staging area.
- This command is equivalent to;

```
rm <filename>
git add <filename>
```

## Git Tags

- Git has tags to highlight important point's in a Repo commit history.
- Like 'v1.0', 'stable version' etc.
- Git supports two types of tags: **annotated tags** and **lightweight tags**.

- **Annotated tags:**
  - These are the tags that are stored as objects in Git history,
  - That is along with this we have info about tagger name, email and date, tagging message same as a commit.
  - You can create tags by **`git tag -a <tagname> -m "<tag message>".`**
  - You can see the tag data by command **`git show <tagname>`**.

## Git Tags

- **Lightweight tags:**
  - These are more like a temporary tag,
  - It does not holds any further info about the tag,
  - To use lightweight tag simply use **`git tag <tagname>`**.

- You can list all the tags by command **`git tag`**.
- You can tag later by command **`git tag -a <tagname> <commit hash> -m <tag message>`**.
- You can delete a tag by command **`git tag -d <tagname>`**.

## Undoing Changes

- You can use **`git commit --amend "<new commit message>"`** to change the existing commit message,
- You can use **`git commit --amend --no-edit`** to amend the new staged changes without a new commit message.
- You can unstage a staged file with the command **`git restore --staged <filename>`**.
- You can unmodify a modified file with command **`git restore <filename>`**.
- You can use **`git restore --source <version> <filename>`** to restore a file of any version.

## Any Questions?

It's time to have some fun !!

Be ready to

Kahoot!

Supported by:

Sikshya Technology

## Thank you!

Please refer to the chat section on our Microsoft Teams for resources and feel free to ask any queries about this session in our discord channel **#git-workshop-query**.