GIT & GITHUB

VERSION CONTROL FOR MODERN DEVELOPMENT

- AP Fall 1403 – Dr. Mojtaba Vahidi Asl
- By Amirmahdi Mashayekhi (mashmool)

LET’S HEAR A STORY TOGETHER

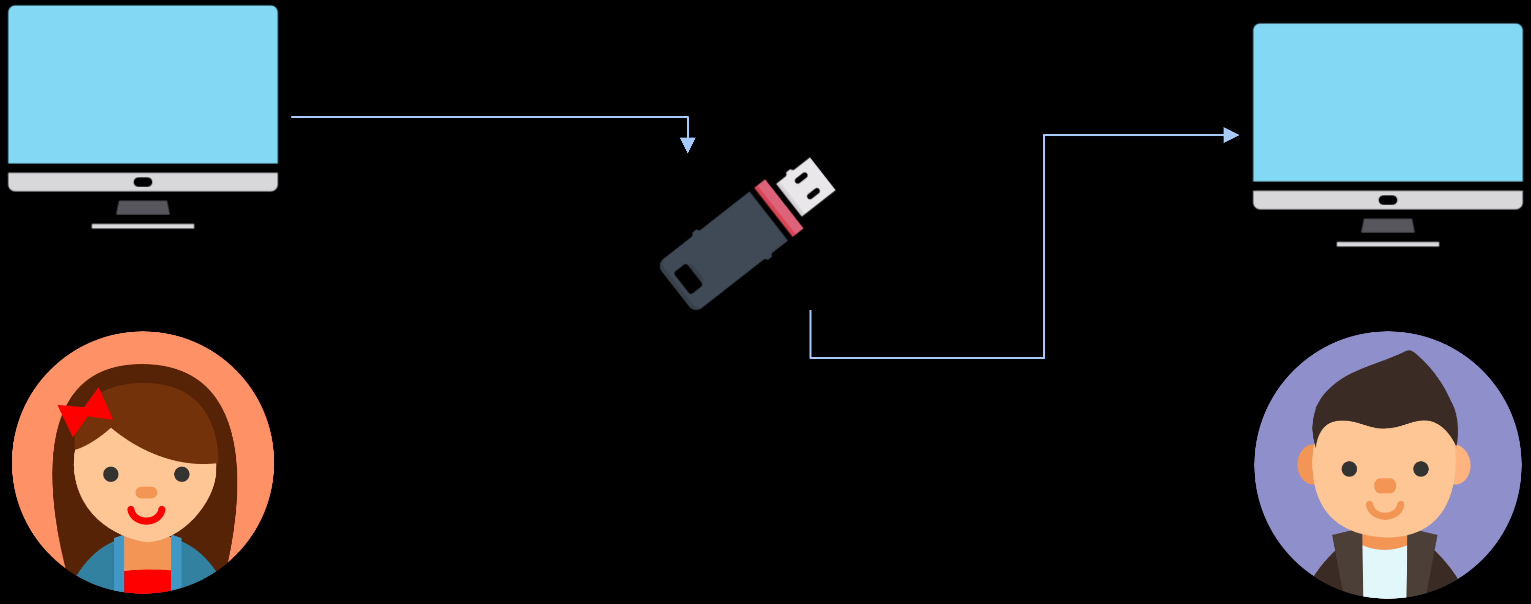
He is Mr. Sam,
a backend developer

She is Ms. Tara
a frontend developer

PROJECT

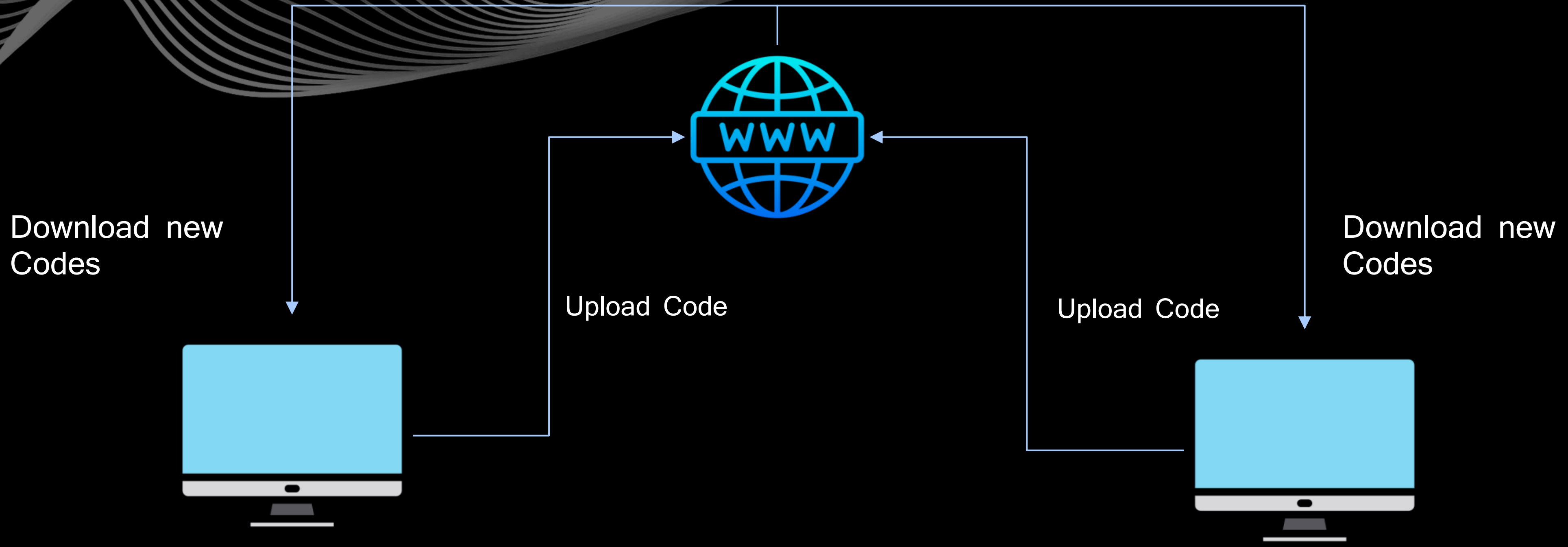
Sam and Tara want to work on a project together.

But they have a problem; they have to transfer their data using a flash drive. Because of this, whenever they want to work, they need to be physically together, and their work speed becomes very slow



IDEA

They came up with an idea



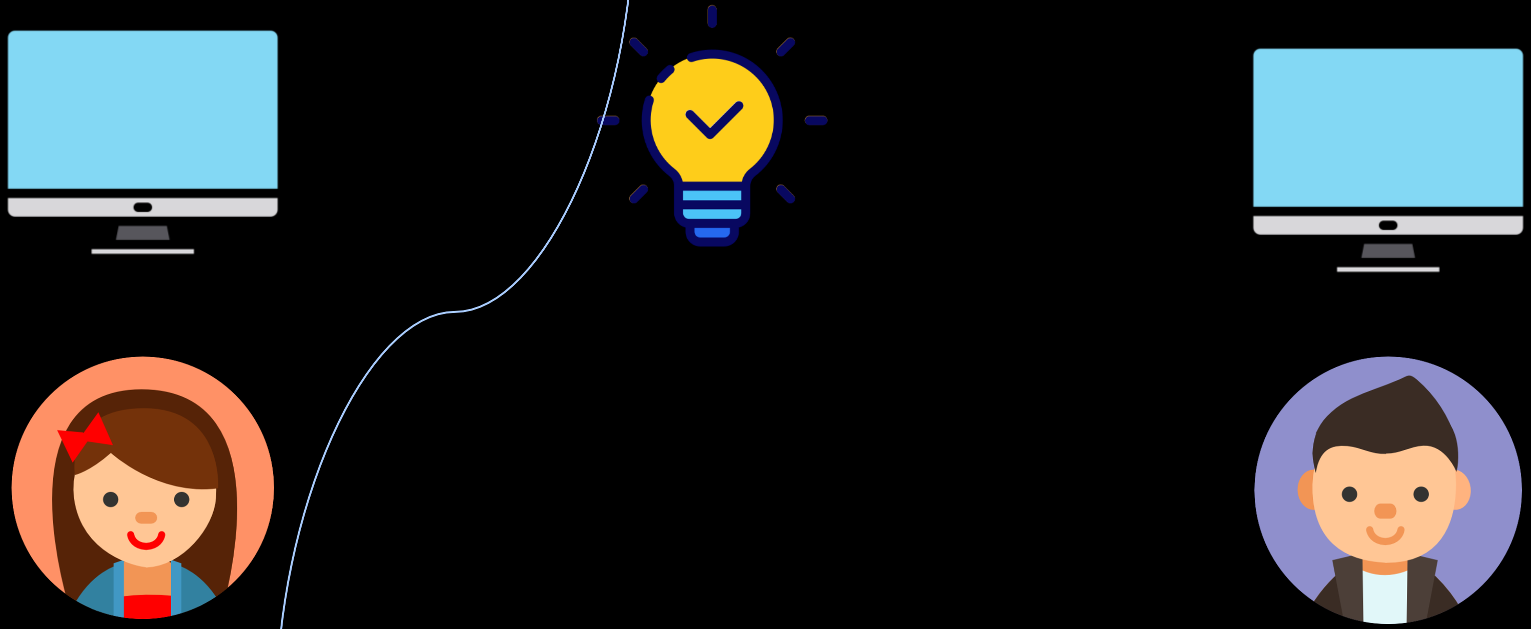
They decided to create a website where they could upload their code and download each other's new code from there.

But they still had some issues. Each time, they had to manually convert their code into a zip file and upload it to the site. The second problem was that sometimes the code conflicted with each other, and they didn't notice it.

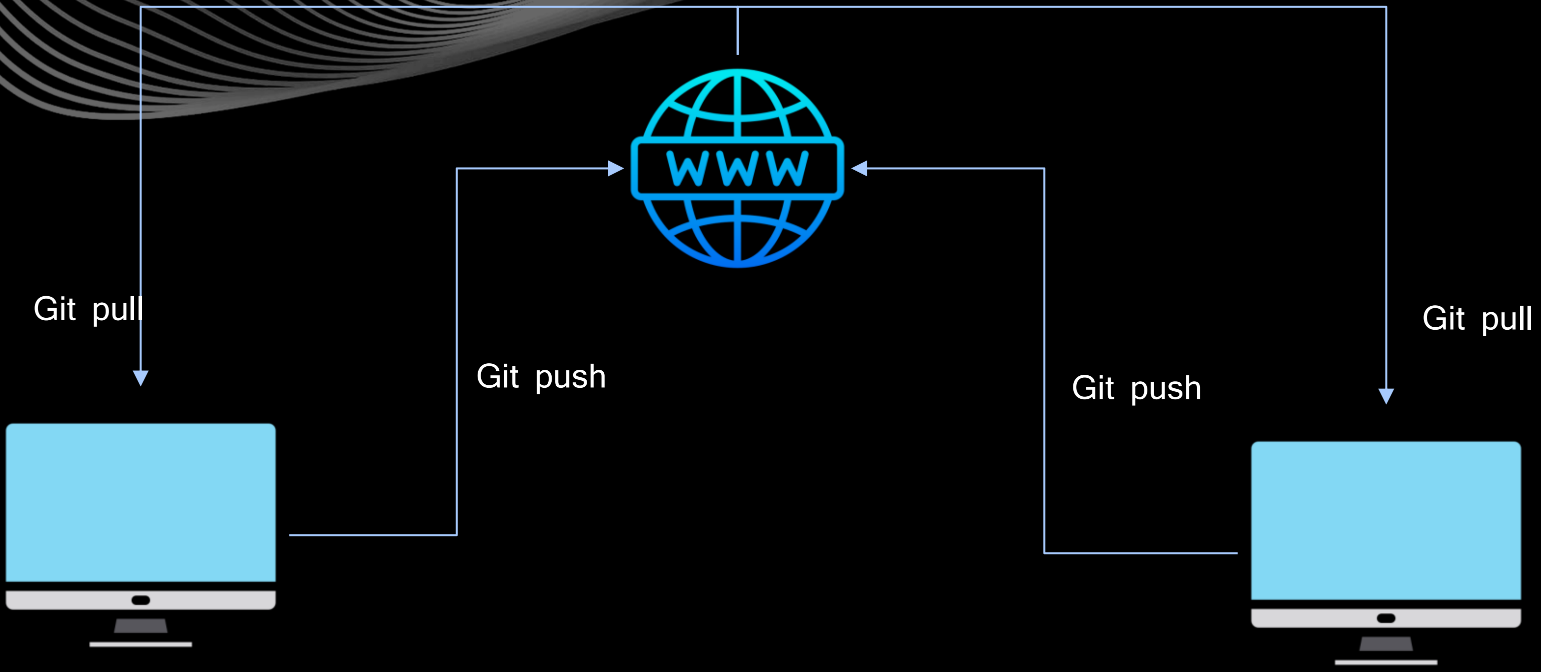
IDEA

They came up with an idea

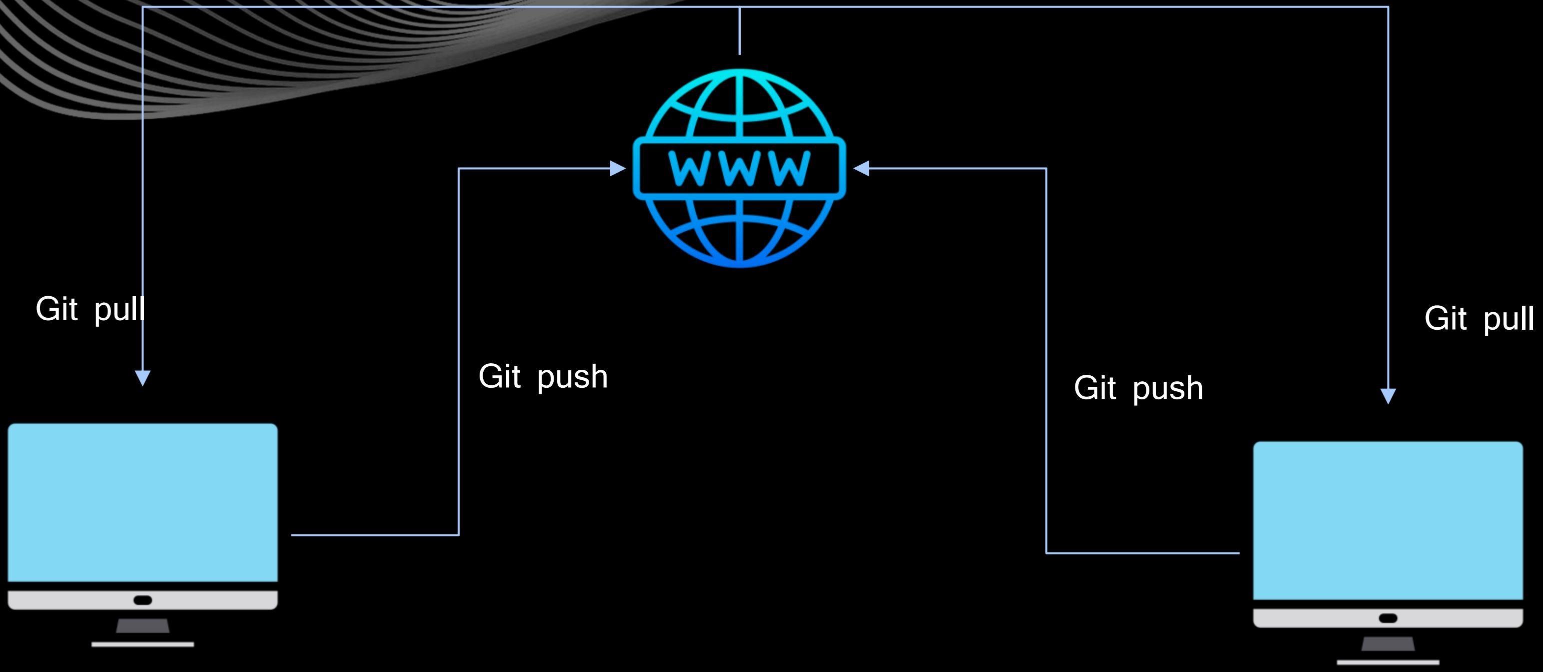
They decided to create a CLI software to solve these problems.



A Command Line Interface (CLI) is a text-based interface used to interact with software and operating systems. Instead of using graphical elements like buttons and windows, users type commands to perform tasks. It's often preferred by developers and system administrators for its efficiency and ability to automate processes.



They named the program 'Git'. In Git, they defined a set of commands for different uses. (In a CLI, when you want to use that software, you need to mention the name of the software before any command.



Over time, they added more features to the software. For example, the program automatically checked the code, and if there were any similarities between their code, it would alert the users with an error to remind them of the conflict. As time went on, the software's features continued to expand.

And in the end, that CLI is the Git we know today, and that website is the GitHub we use now.

THE END

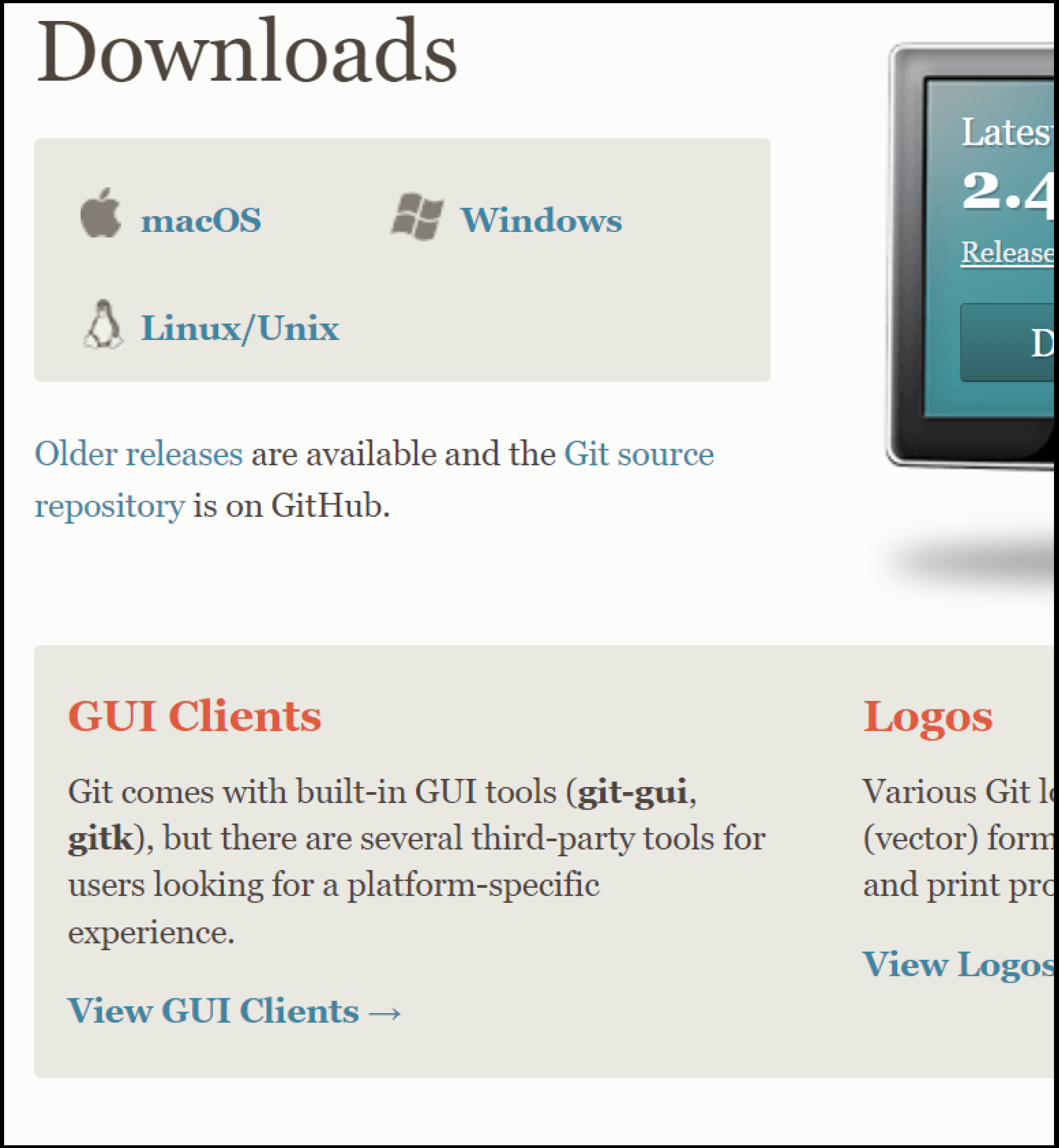
This entire story is a creation of the mind of the author (Amirmahdi Mashayekhi) and is purely for educational purpose

INTRODUCTION TO VERSION CONTROL

- **Main Point**: Version control is a system that helps track changes to files over time. It allows multiple people to work on a project simultaneously without overwriting each other’s changes.
- **Detailed Explanation**:
  - Version control is essential in software development but is also widely used in any collaborative work with digital content (such as writing documents or managing configurations).
  - It helps **manage changes** to a project’s source code, ensuring a record of every modification, which can be reverted or tracked later.

WHAT IS GIT ?

- **Definition**
  - Git is a **Distributed Version Control System** (DVCS).
  - It tracks changes in source code during software development.
- **How Git Works**
  - **Local vs. Remote Repositories**: Work on your local machine and sync with remote servers.
  - **Commits**: Snapshots of your project at a certain point in time.
  - **Branches**: Allows parallel development (e.g., features, bug fixes).
  - **Merges**: Combine changes from different branche



Downloads

macOS Windows

Linux/Unix

Latest 2.4 Release D

Older releases are available and the Git source repository is on GitHub.

GUI Clients

Git comes with built-in GUI tools (**git-gui**, **gitk**), but there are several third-party tools for users looking for a platform-specific experience.

View GUI Clients →

Logos

Various Git l (vector) form and print pro

View Logos

INSTALLING GIT

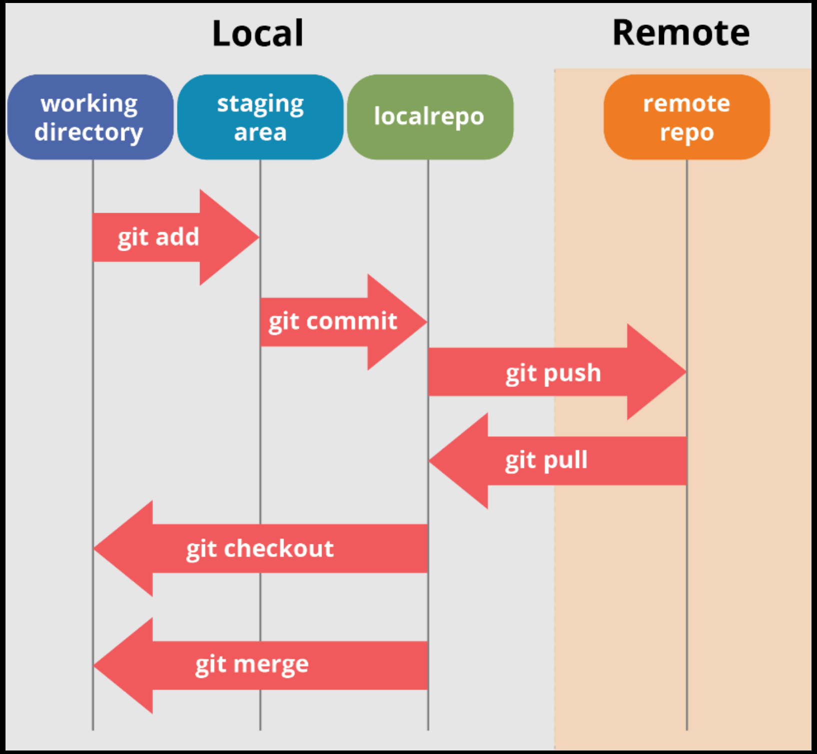
- Include details about a special day from the era you are discussing
- Add an image from the day
- Include the date and, if any, significant people involved
- You can put text in both boxes
- You can put images in both boxes
- You can put text in one box and an image in the other

BASIC GIT WORKFLOW

- **Important Commands :**
  - Git init : Initialize a new repository
  - Git add : Add files to Staging .
  - Git commit : Commit Changes to local repository
  - Git status : Check Current Changes
  - Git log : View commit History

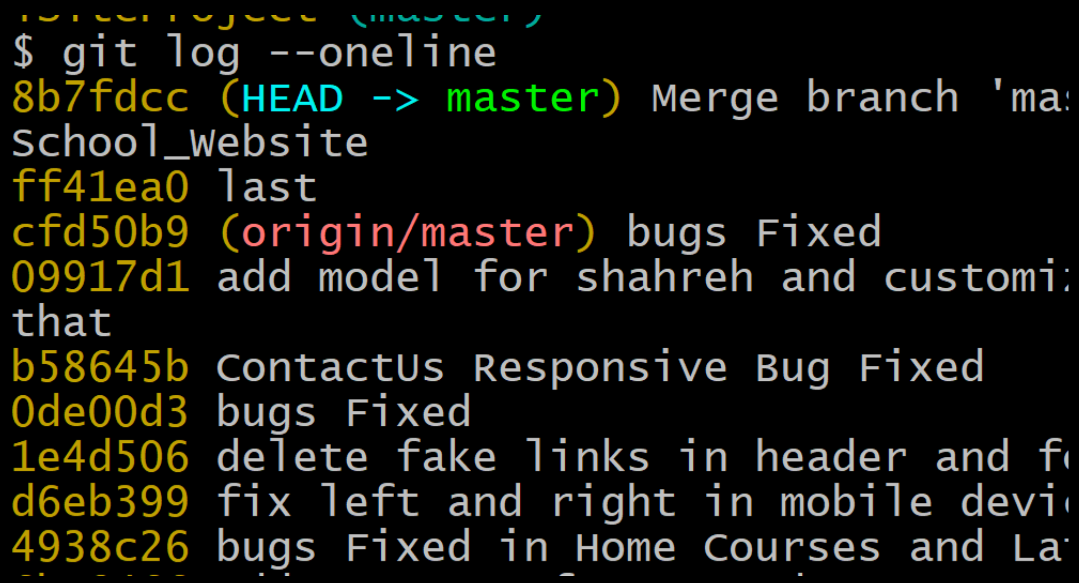
GIT STEPS

- Add a simple flowchart showing the Git steps:
- git init → git add → git commit → git push.
- Why Staging Area ???



WHAT IS LOG ?

- A log is a file or record containing information about activities in a computer system
- Git log
- Git log -oneline
- Git log -n <number>
- …..



```
$ git log --oneline
8b7fdcc (HEAD -> master) Merge branch 'ma
School_Website
ff41ea0 last
cfd50b9 (origin/master) bugs Fixed
09917d1 add model for shahreh and customi
that
b58645b ContactUs Responsive Bug Fixed
0de00d3 bugs Fixed
1e4d506 delete fake links in header and f
d6eb399 fix left and right in mobile devi
4938c26 bugs Fixed in Home Courses and La
```

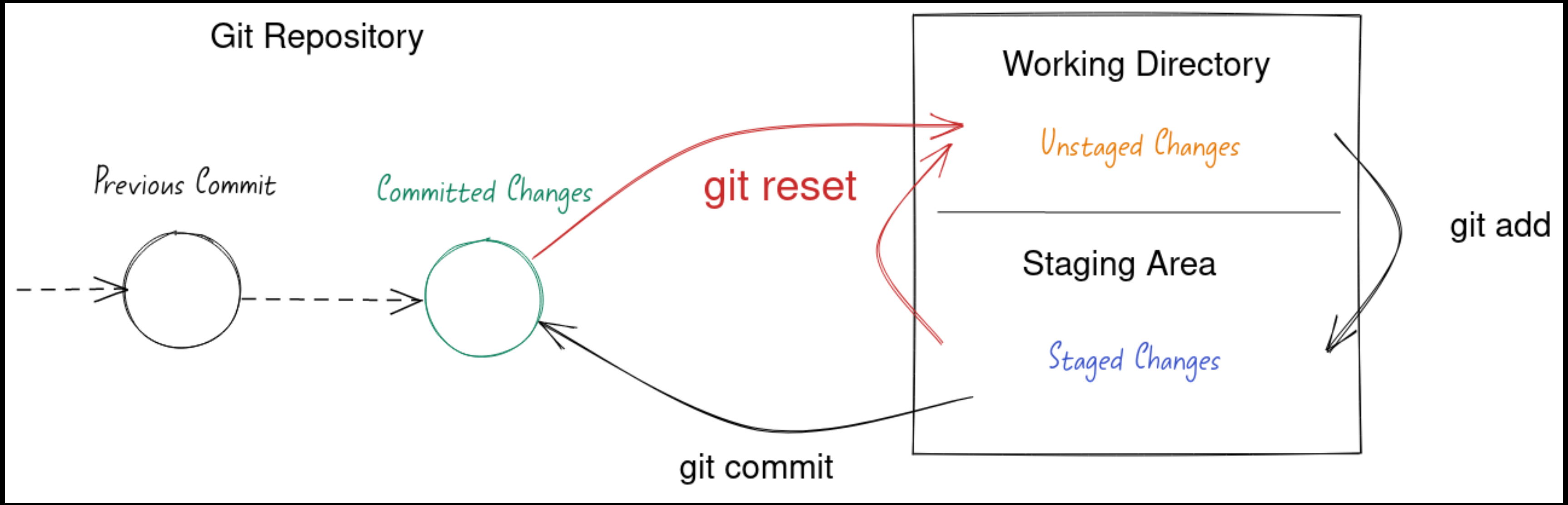
UNDOING UNSTAGED CHANGES

- **1. What are Unstaged Changes?**
  - **These are changes made to files but not yet added to the staging area with git add.**
- **2. Undo Unstaged Changes**
  - **Git restore <file_name>**
  - Restores the specified file to the state it was in at the last commit, discarding local modifications.

UNDOING STAGED CHANGES

- **1. What are Staged Changes?**
- Changes that have been added to the staging area but not yet committed.
- **2. Unstage a File**
  - Git restore -staged <file_name>
  - Creates a new commit that undoes the changes introduced by the given commit. It keeps the commit history intact.

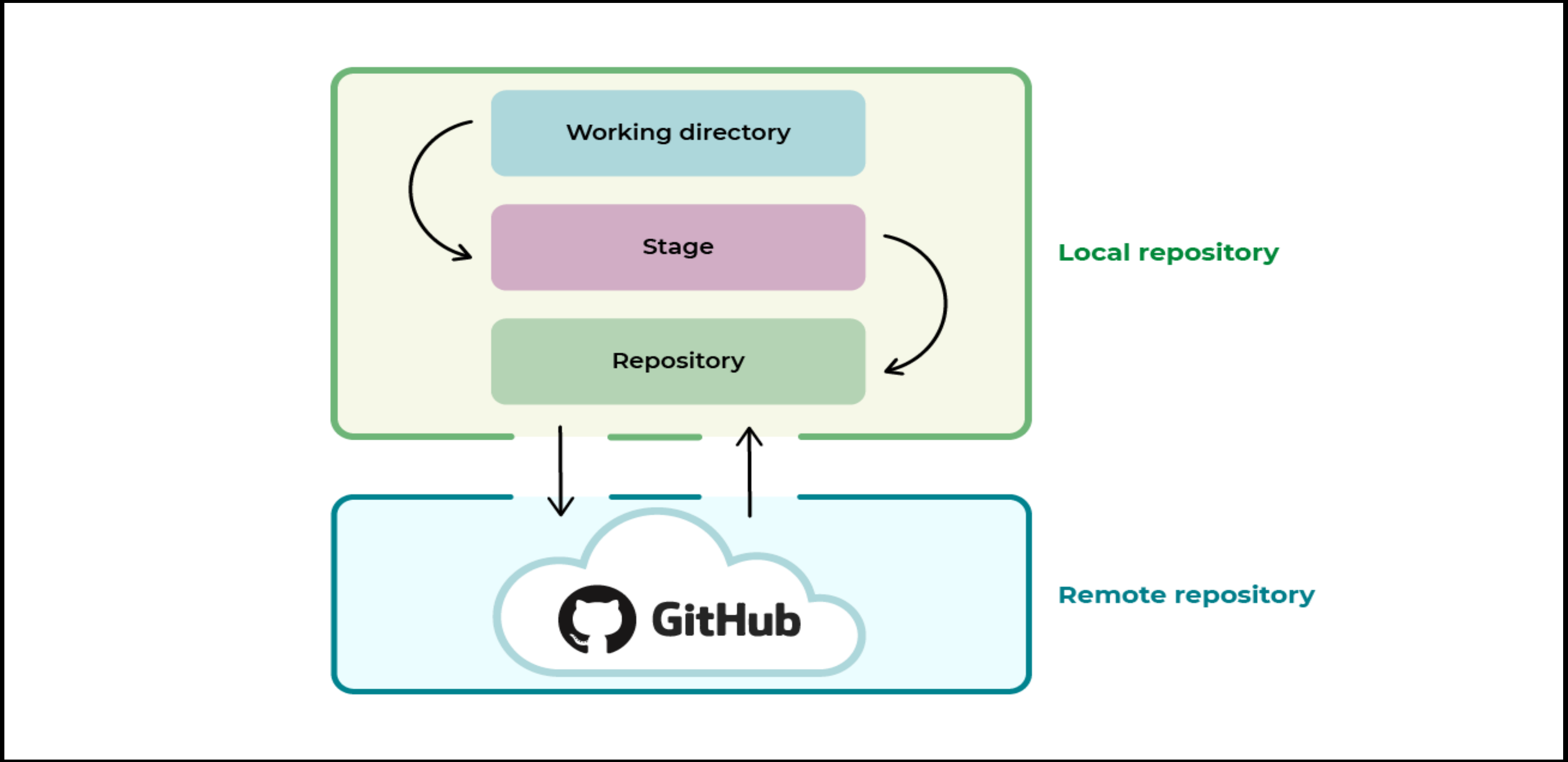
UNDOING IN GIT



REVERTING COMMITS

- **1. What is Reverting a Commit?**
  - A Git operation that undoes a specific commit without removing it from history.
- **2. Revert a Commit**
  - **Git revert <commit_hash> what is hash ?**
  - Discards all local modifications across all files that haven't been staged.

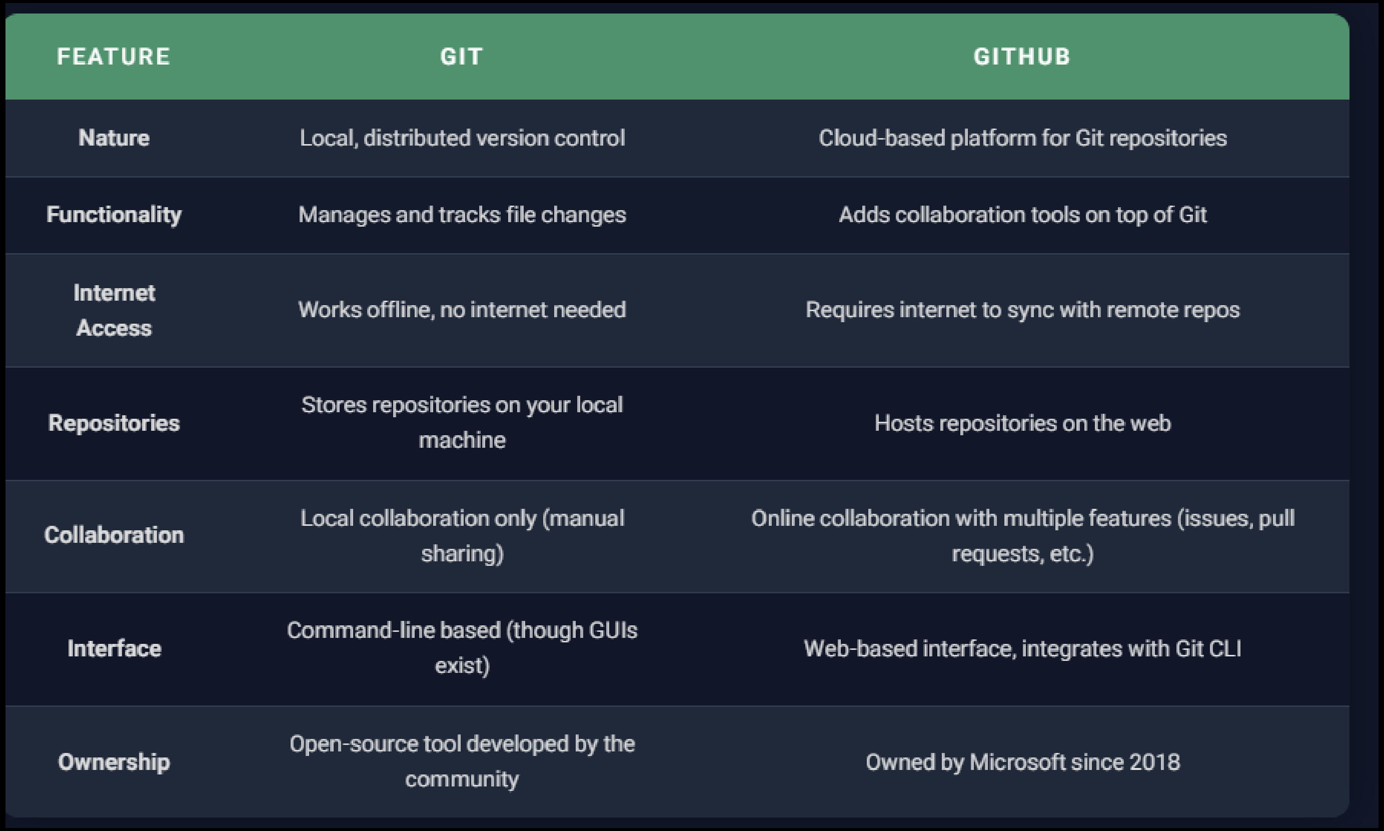
REMOTE REPO


Working directory
Stage
Repository
Local repository
GitHub
Remote repository

WHAT IS GITHUB ?

- **Definition**
  - GitHub is a **cloud-based platform** for hosting Git repositories.
  - It provides tools to help developers collaborate on projects and manage version control more easily
- **Key Features**
  - **Repository Hosting**: Store your Git projects online.
  - **Collaboration**: Work with others using pull requests, code reviews, and issues.
  - **Forking and Cloning**: Fork public repositories to make your own changes, then submit pull requests to contribute back.
  - **Version Control**: Track and manage changes in your project using Git, accessible anywhere through GitHub.

GITHUB VS GIT



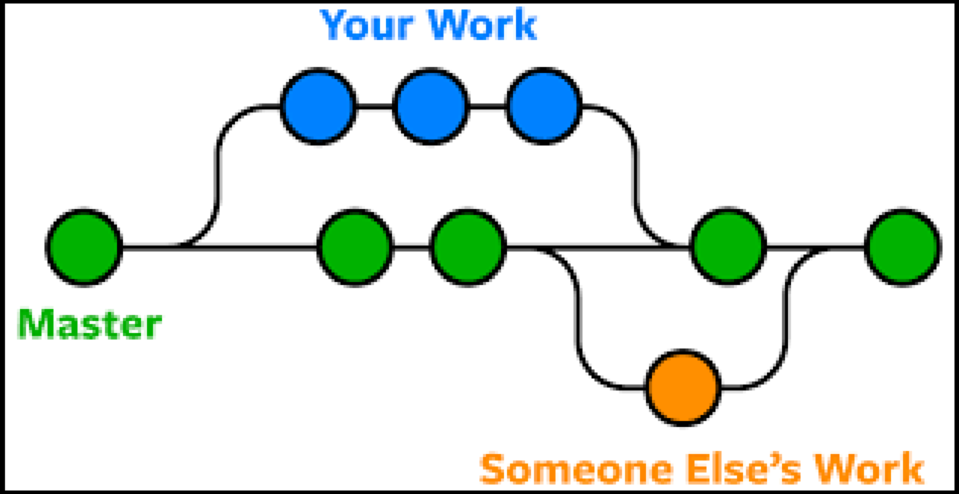
FEATURE	GIT	GITHUB
Nature	Local, distributed version control	Cloud-based platform for Git repositories
Functionality	Manages and tracks file changes	Adds collaboration tools on top of Git
Internet Access	Works offline, no internet needed	Requires internet to sync with remote repos
Repositories	Stores repositories on your local machine	Hosts repositories on the web
Collaboration	Local collaboration only (manual sharing)	Online collaboration with multiple features (issues, pull requests, etc.)
Interface	Command-line based (though GUIs exist)	Web-based interface, integrates with Git CLI
Ownership	Open-source tool developed by the community	Owned by Microsoft since 2018

GitHub

VS

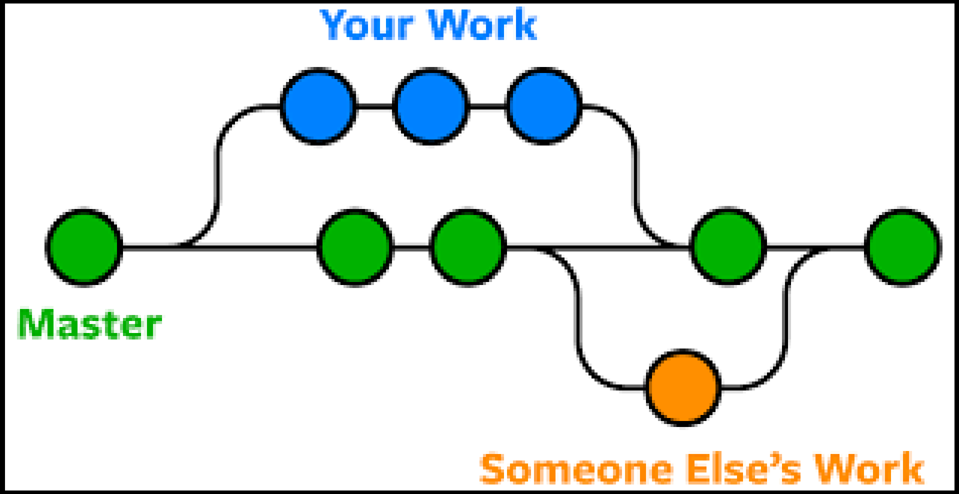
WHAT IS BRANCH ?

- **What is a Branch?**
  - **Definition: A branch in Git is a parallel version of your project. It allows you to work on different parts of your project without affecting the main branch (typically called master or main).**
  - **Purpose: Branches are used to develop new features, fix bugs, or experiment with new ideas while keeping the main codebase stable.**
- **Why Use Branches?**
  - **Parallel Development**: Multiple people can work on different features or fixes at the same time.
  - **Isolated Changes**: Each branch can contain its own set of changes without interfering with others. This is useful for managing multiple aspects of a project.
  - **Safe Experimentation**: Test new features or code without affecting the rest of the



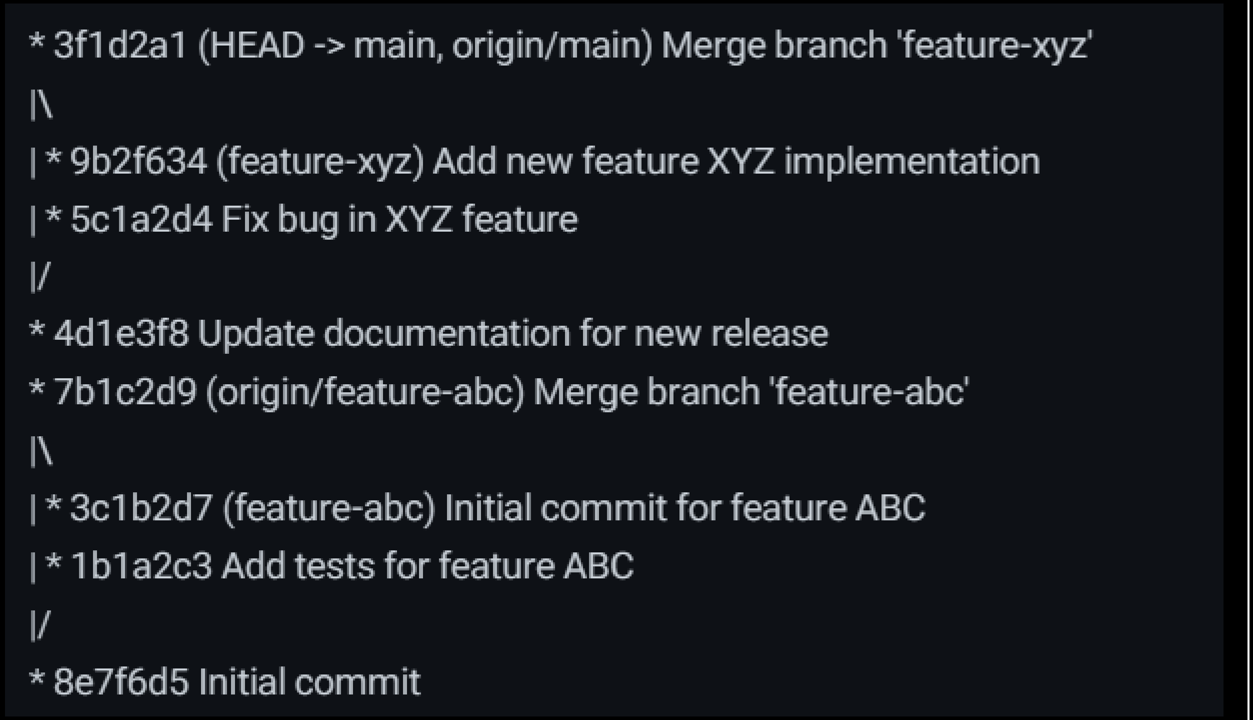
HOW BRANCHES WORK ?

- **Create new branch :**
  - **Git branch <branch_name>**
- **Switch to a branch :**
  - **Git checkout <branch_name>**
- **Create and Switch to a New Branch :**
  - **Git checkout -b <branch_name>**



BRANCH AND LOGS ?

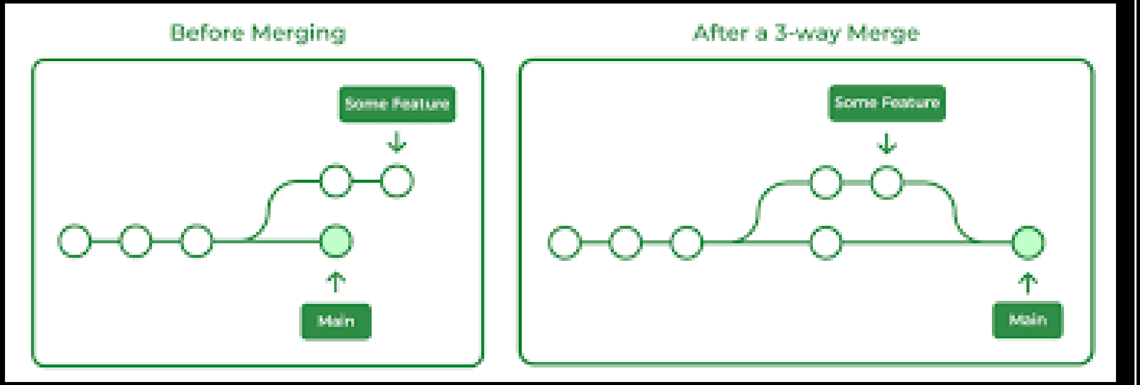
- **git log --oneline --graph --decorate -all**
  - **--oneline: Displays one commit per line.**
  - **--graph: Shows an ASCII graph of the branch and merge history next to the log.**
  - **--decorate: Adds branch and tag names to the commit display.**
  - **--all: Shows all commits from all branches.**



```
* 3f1d2a1 (HEAD -> main, origin/main) Merge branch 'feature-xyz'
|\
| * 9b2f634 (feature-xyz) Add new feature XYZ implementation
| * 5c1a2d4 Fix bug in XYZ feature
|/
* 4d1e3f8 Update documentation for new release
* 7b1c2d9 (origin/feature-abc) Merge branch 'feature-abc'
|\
| * 3c1b2d7 (feature-abc) Initial commit for feature ABC
| * 1b1a2c3 Add tests for feature ABC
|/
* 8e7f6d5 Initial commit
```

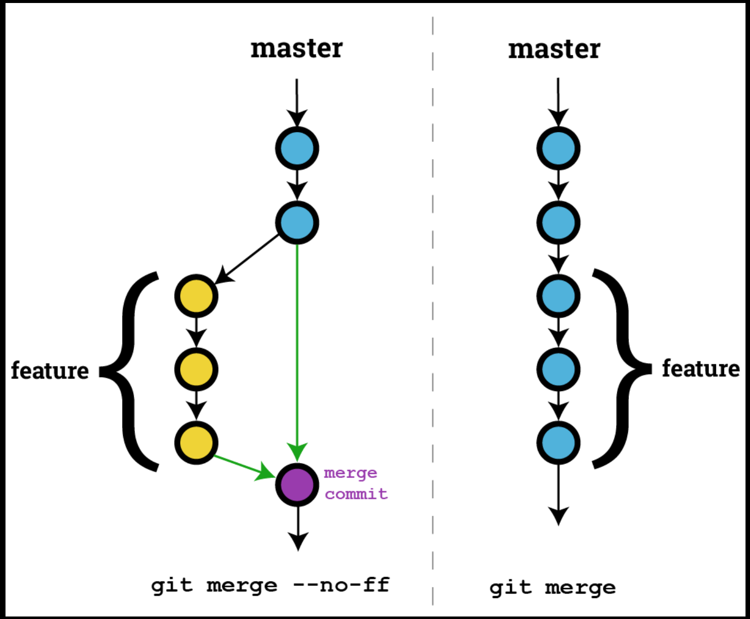
WHAT IS MERGING ?

- **What is Merging?**
  - **Definition**: Merging in Git is the process of combining changes from one branch into another. Typically, after finishing work on a feature branch, you merge it back into the main branch.
  - **Purpose**: To integrate the changes from multiple branches and create a unified version of the project.
- **Types of Merges**
  - **Fast-Forward Merge**: If no new commits have been made on the main branch, Git simply moves the branch pointer forward. This type of merge is simple and doesn't create a new commit.
  - **Three-Way Merge**: If both the feature branch and the main branch have diverged, Git creates a new "merge commit" that incorporates changes from both branches.



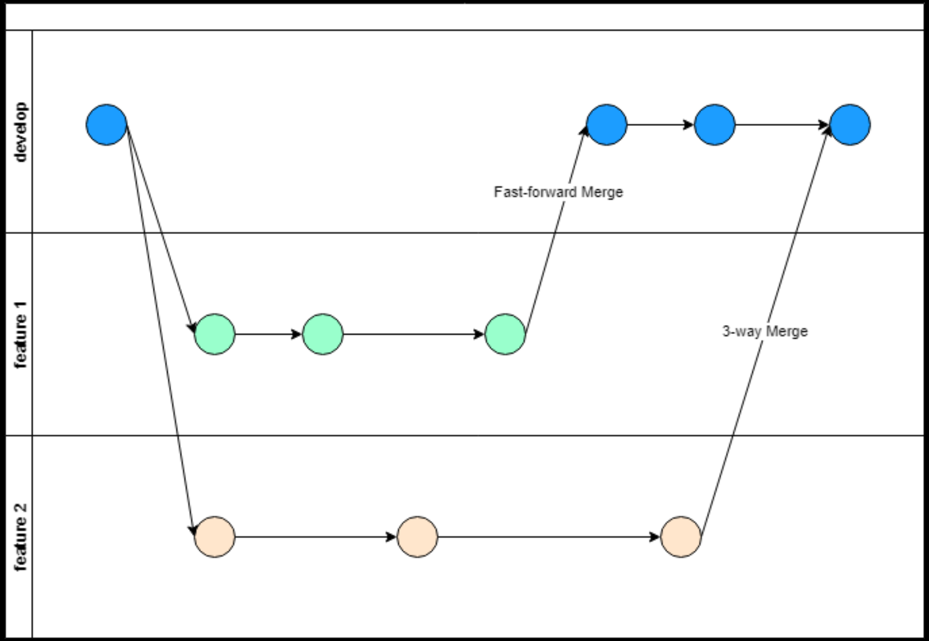
TYPES OF MERGES

- **1. Fast-Forward Merge**
  - **Definition**: When no new commits have been made on the main branch since the branch was created, Git simply moves the branch pointer forward to the latest commit on the feature branch.
  - **Characteristics**:
    - No new commit is created during the merge.
    - The branch history looks linear, as if there was no divergence.
    - git merge feature-xyz

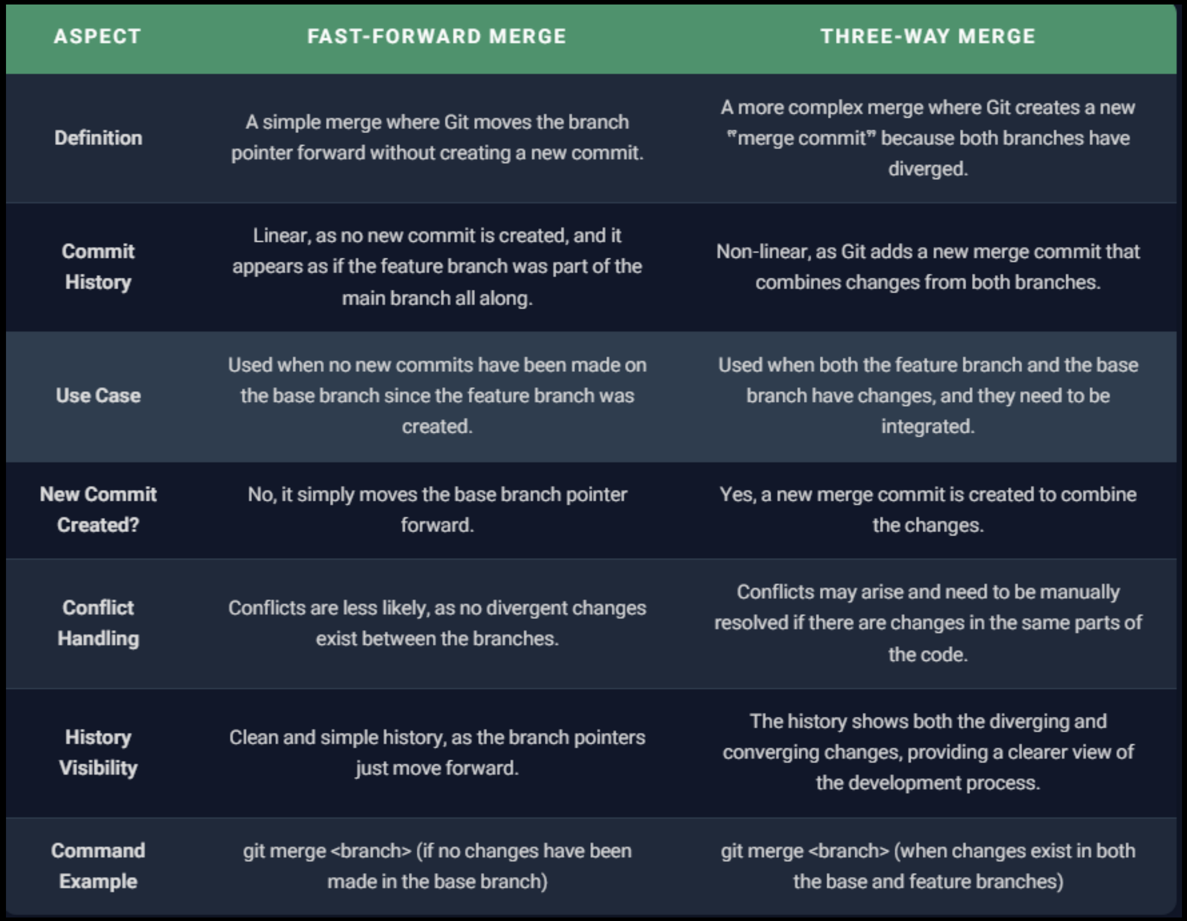


TYPES OF MERGES

- **2. Three-Way Merge**
  - **Definition**: A three-way merge occurs when both the main branch and the feature branch have diverged. Git combines changes from both and creates a new commit known as a “merge commit”.
  - **Characteristics**:
    - A new commit is created that links the two branches.
    - The branch history shows the divergence and convergence.
    - git merge feature-abc



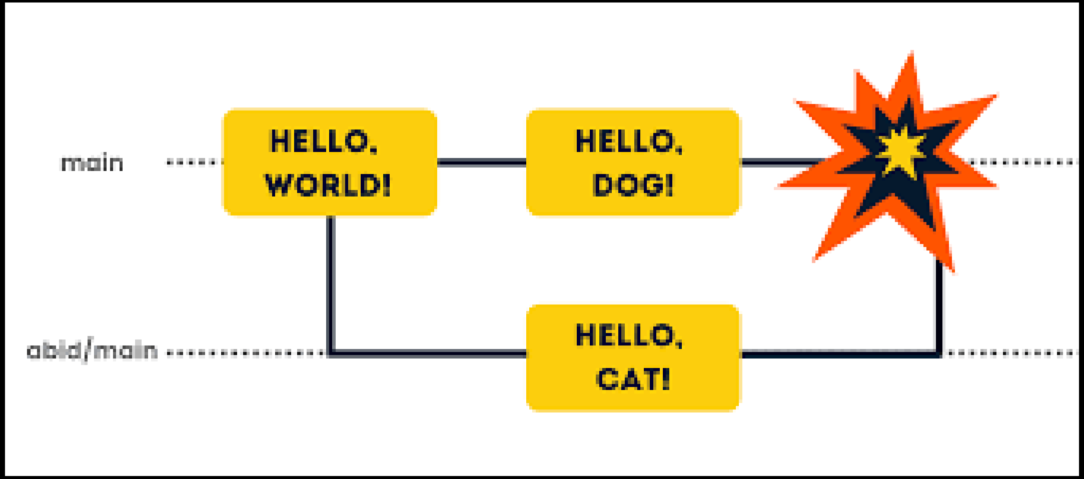
THREE-WAY VS FAST-FORWARD



ASPECT	FAST-FORWARD MERGE	THREE-WAY MERGE
Definition	A simple merge where Git moves the branch pointer forward without creating a new commit.	A more complex merge where Git creates a new "merge commit" because both branches have diverged.
Commit History	Linear, as no new commit is created, and it appears as if the feature branch was part of the main branch all along.	Non-linear, as Git adds a new merge commit that combines changes from both branches.
Use Case	Used when no new commits have been made on the base branch since the feature branch was created.	Used when both the feature branch and the base branch have changes, and they need to be integrated.
New Commit Created?	No, it simply moves the base branch pointer forward.	Yes, a new merge commit is created to combine the changes.
Conflict Handling	Conflicts are less likely, as no divergent changes exist between the branches.	Conflicts may arise and need to be manually resolved if there are changes in the same parts of the code.
History Visibility	Clean and simple history, as the branch pointers just move forward.	The history shows both the diverging and converging changes, providing a clearer view of the development process.
Command Example	git merge <branch> (if no changes have been made in the base branch)	git merge <branch> (when changes exist in both the base and feature branches)

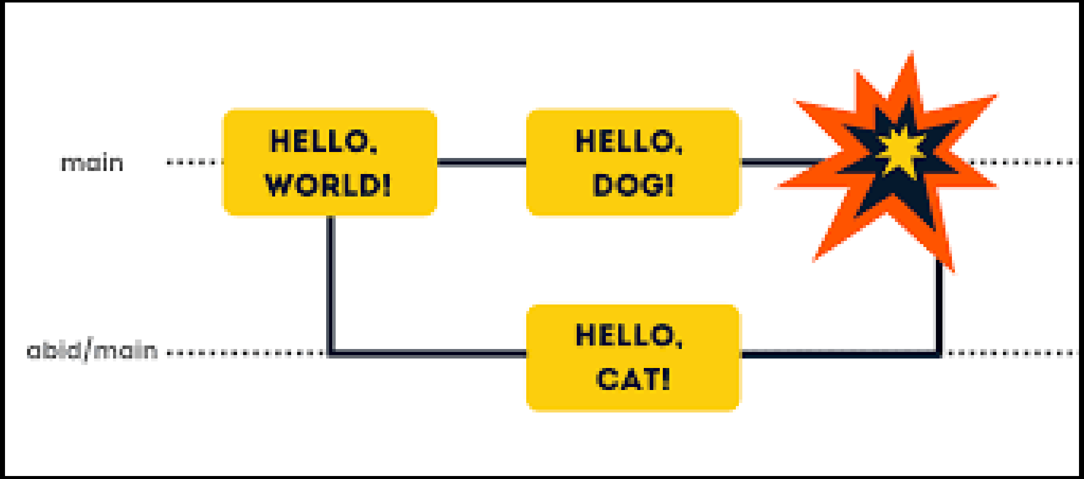
WHAT IS CONFLICT ?

- **1. Definition of Merge Conflicts**
  - A **merge conflict** happens when Git is unable to automatically resolve differences between two branches being merged.
  - This usually occurs when two or more developers modify the same line(s) of code in different ways, or when one developer deletes a file that another developer modifies.
-



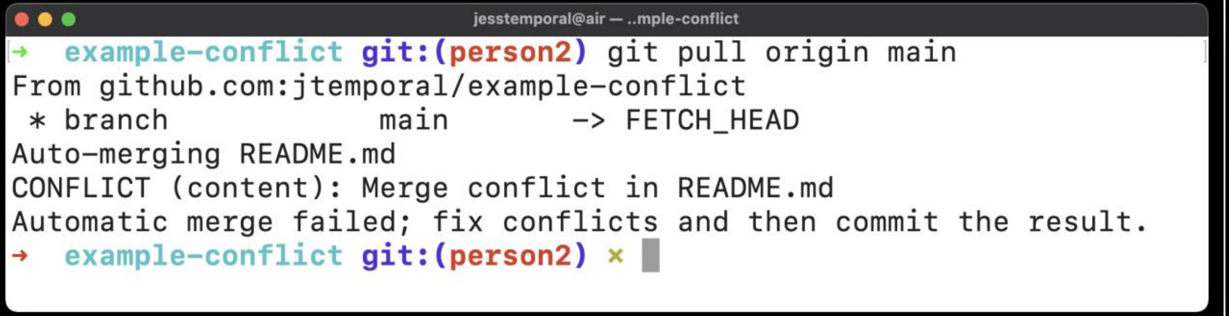
WHAT IS CONFLICT ?

- **When Do Conflicts Arise?**
  - **Simultaneous Changes**: The same file is edited in both branches.
  - **Contradictory Changes**: The same line of code is altered differently in both branches.
  - **File Deletion vs. Modification**: One branch deletes a file that the other modifies.
  - **Overlapping Structural Changes**: Large-scale changes to the project structure, such as moving or renaming files or directories, can cause conflicts.



HOW TO DETECT AND IDENTIFY CONFLICTS

- **1. Git's Response to a Conflict :**
  - When a conflict arises, Git halts the merge and alerts you with a message similar to:
    - CONFLICT (content): Merge conflict in <file-name>
    - Automatic merge failed; fix conflicts and then commit the result.

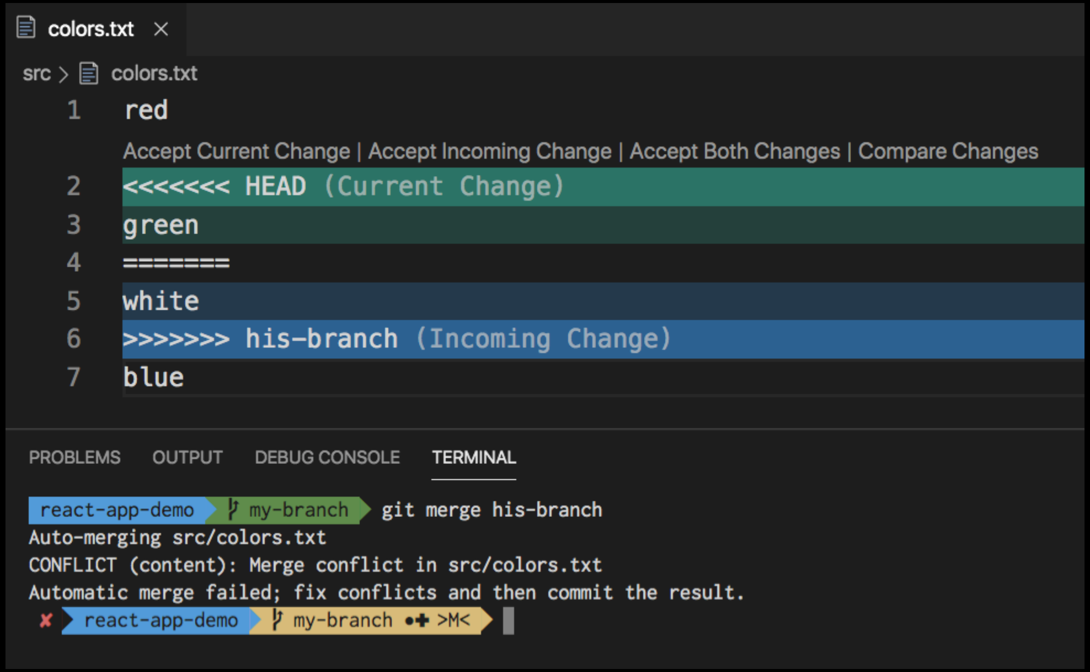


jesstemporal@air — ..mple-conflict

```
→  example-conflict git:(person2) git pull origin main
From github.com:jtemporal/example-conflict
 * branch            main       -> FETCH_HEAD
Auto-merging README.md
CONFLICT (content): Merge conflict in README.md
Automatic merge failed; fix conflicts and then commit the result.
→  example-conflict git:(person2) ×
```

HOW TO DETECT AND IDENTIFY CONFLICTS

- **2. Markers in Conflicted Files :**
  - Git inserts conflict markers in the affected file(s):
    - <<<<<<< HEAD Code from the current branch ======= Code from the branch being merged >>>>>>> branch-name
- **3. Command to Identify Conflicted Files:**
  - Use the following command to list all files with conflicts :
    - git status
    - Conflicted files will be listed under the "Unmerged paths" section.

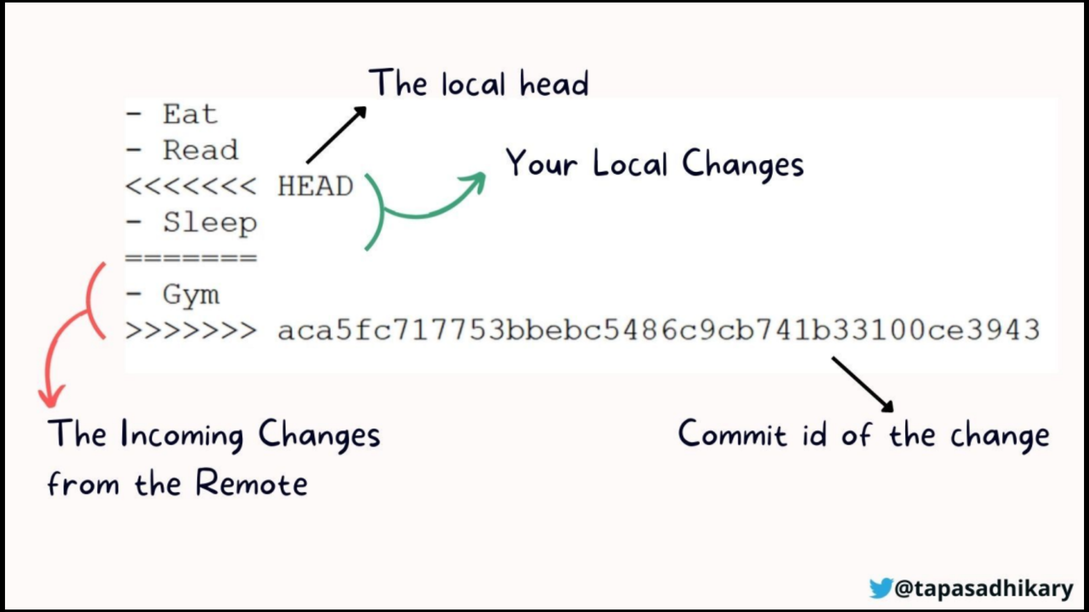

colors.txt
src > colors.txt
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL

```
red
<<<<<<< HEAD (Current Change)
green
=======
white
>>>>>>> his-branch (Incoming Change)
blue
```

```
react-app-demo  my-branch  git merge his-branch
Auto-merging src/colors.txt
CONFLICT (content): Merge conflict in src/colors.txt
Automatic merge failed; fix conflicts and then commit the result.
✘  react-app-demo  my-branch ●✚ >M<
```

STEPS TO RESOLVE A MERGE CONFLICT

- **1. Manual Resolution Process**
  - **Step 1: Open the conflicted files using your text editor or IDE.**
  - **Step 2: Locate the conflict markers (<<<<<<<, =======, >>>>>>>).**
  - **Step 3: Decide which code to keep or combine the changes from both branches as needed.**
  - **Step 4: Remove the conflict markers after resolving the conflict.**



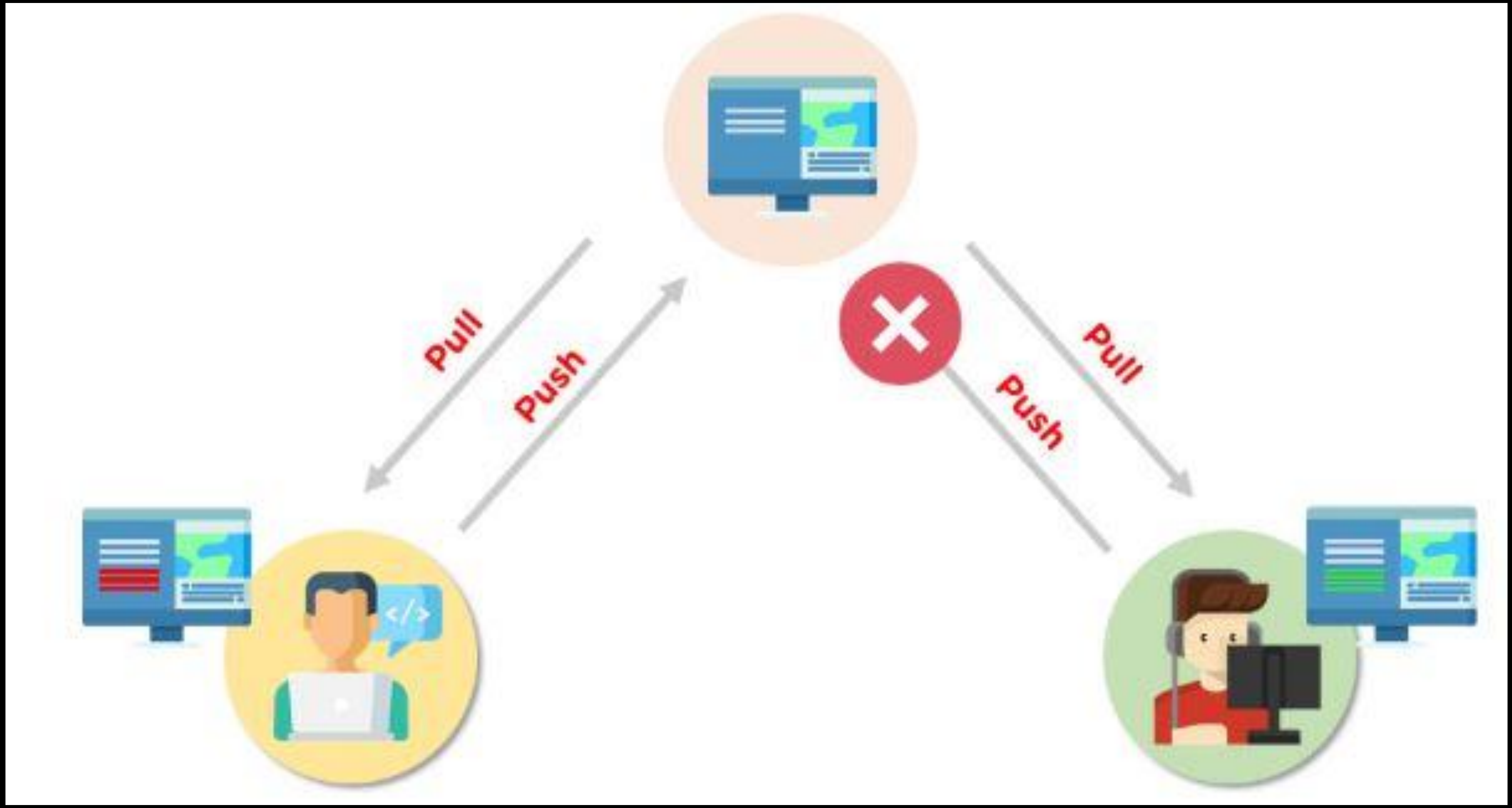
```
- Eat
- Read
<<<<<<< HEAD
- Sleep
=======
- Gym
>>>>>>> aca5fc717753bbebc5486c9cb741b33100ce3943
```

- **2. Adding Resolved Files :**
  - Once conflicts are resolved, mark the file as resolved by staging it:
    - git add <file-name>
    - git commit

STEPS TO RESOLVE A MERGE CONFLICT

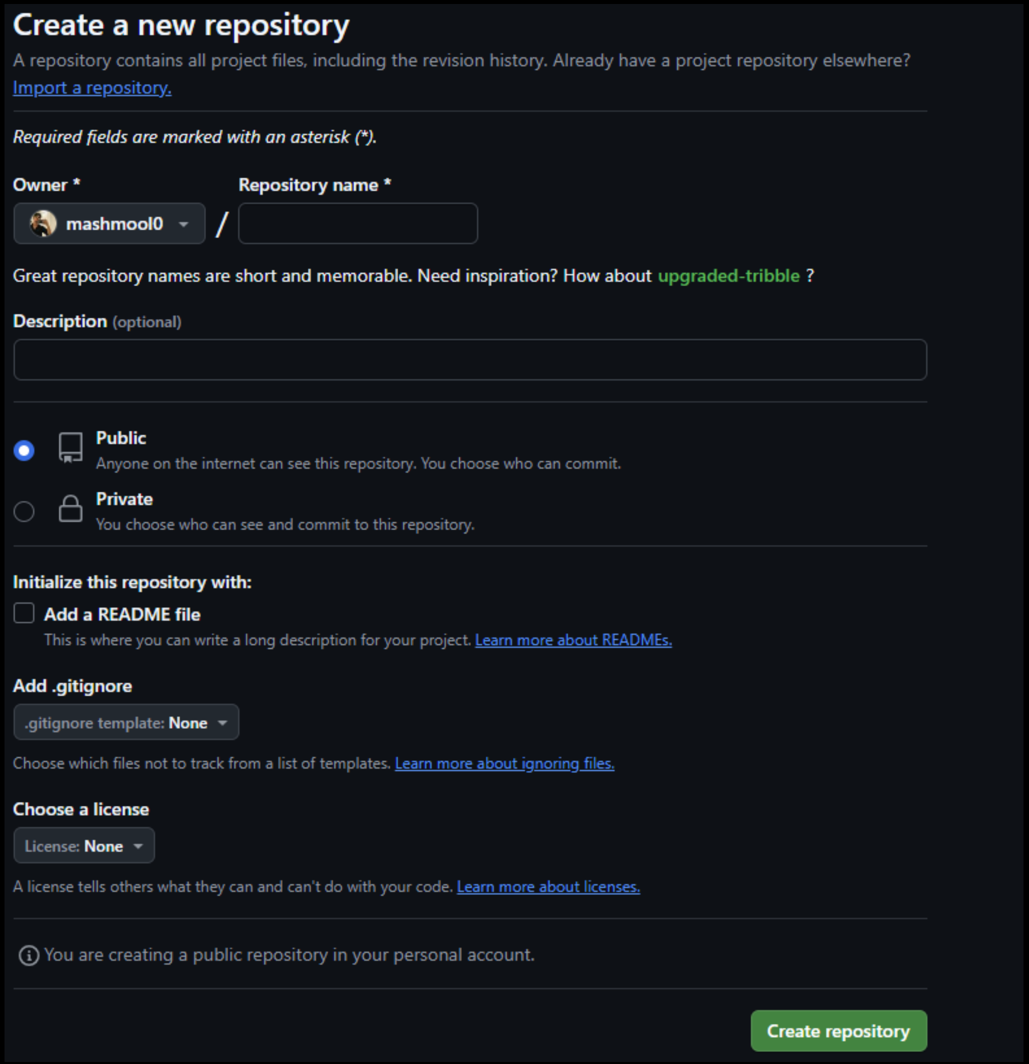
- **4. Aborting a Merge :**
  - If the conflict is too complex or you wish to cancel the merge, you can abort the process:
    - git merge --abort

HOW TO DETECT AND IDENTIFY CONFLICTS



WHAT IS A REPOSITORY IN GITHUB?

- **1. Definition of a Repository**
  - A **repository** (or **repo**) in GitHub is a storage space where your project's files and version history are kept. It can include code, documentation, images, or any other resources needed for the project.
- **2. Key Features of a GitHub Repository**
  - **Version Control**: Every change made to the codebase is tracked, allowing users to go back to previous versions if necessary.
  - **Collaboration**: Multiple developers can work on the same repository simultaneously, using branches and pull requests to propose and review changes.
  - **Remote Access**: A GitHub repository is hosted online, enabling global access and contribution from anywhere.
  - **Visibility**: Repositories can be **public** (visible to everyone) or **private** (only accessible to



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 * mashmool0 / Repository name *

Great repository names are short and memorable. Need inspiration? How about upgraded-tribble ?

Description (optional)

[✓ **Public**] Anyone on the internet can see this repository. You choose who 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.

Add .gitignore

.gitignore template: None

Choose which files not to track from a list of templates. Learn more about ignoring files.

Choose a license

License: None

A license tells others what they can and can't do with your code. Learn more about licenses.

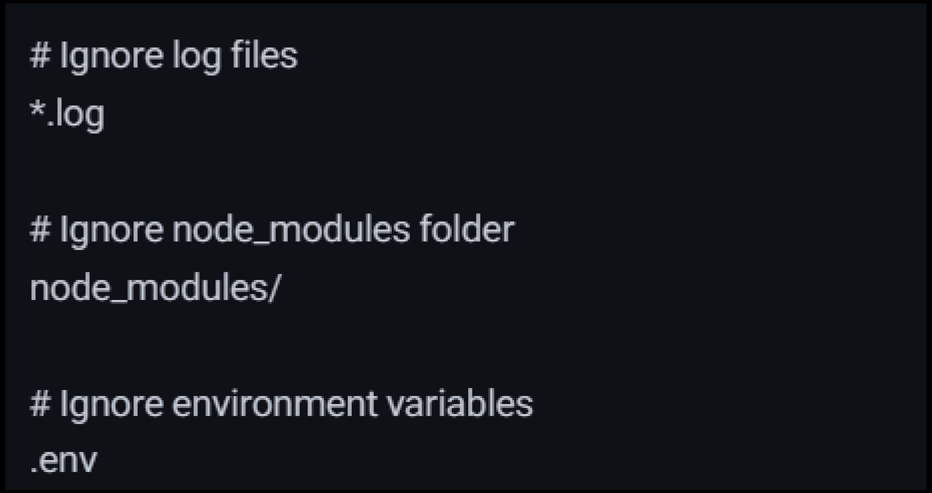
ⓘ You are creating a public repository in your personal account.

WHAT IS .GITIGNORE ?

- **1. Definition of .gitignore**
  - **The .gitignore file is a configuration file that tells Git which files or directories to ignore when committing changes to the repository. It prevents unnecessary files (e.g., build artifacts, logs, or sensitive data) from being tracked.**
- 2. Purpose of .gitignore
  - **Maintains a clean repository**: Keeps unnecessary files like build artifacts, temporary files, or system files out of the version history.
  - **Protects sensitive data**: Prevents sensitive files like credentials, API keys, or configuration files from being accidentally pushed to the repository.

WHAT IS .GITIGNORE ?

- A sample .gitignore file:



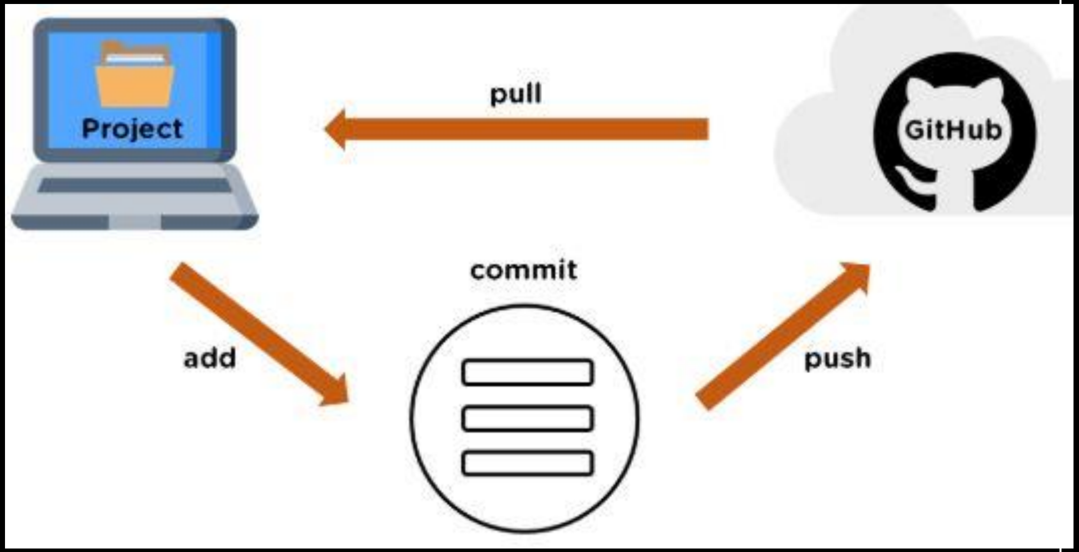
```
# Ignore log files
*.log

# Ignore node_modules folder
node_modules/

# Ignore environment variables
.env
```

HOW TO USE GITHUB WITH GIT

- **1. Pushing Local Changes to GitHub**
  - After making commits locally, you can push your changes to a remote repository on GitHub :
    - git push origin main
- **2. Cloning a Repository from GitHub**
  - To download a copy of a repository from GitHub to your local machine, use:
    - git clone <repository_url>



LETS SEE

THANKS FOR YOUR ATTENTION