

# Version Control System - Git

# Today's Agenda

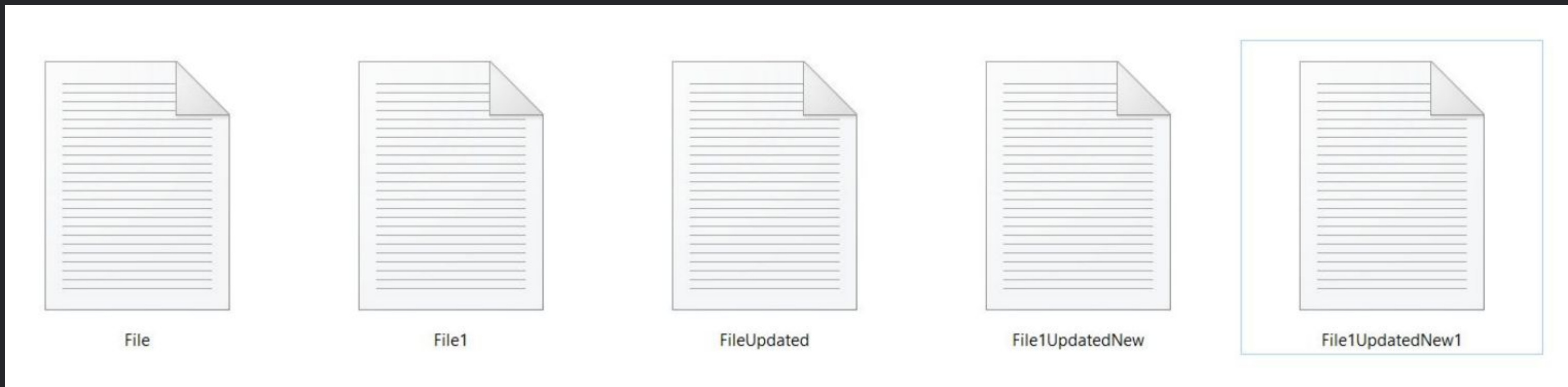
1. Introduction to Version Control System and Git
2. Starting with Git
3. Making changes to the local repository
4. Branching in Git
5. Syncing with the central repository

# Introduction to Version Control System and Git

# Introduction to VCS

**Question:** How will you maintain different versions of a file?

**Answer:** Create a copy of the file and make changes to the new file.



# Introduction to VCS

## Issues with manual version control:

- Difficult to identify the changes between two files
- Not efficient with projects having thousands of files
- Difficult to collaborate with the team and other programmers
- Difficult to store metadata such as who made what changes, date and reason

It is quite difficult to maintain file versions manually. This is where the Version Control System helps you.

The basic Version Control System is available on Google Drive also (if any one of you have used it)

# Introduction to VCS

## What is VCS?

The Version Control System (VCS) helps you manage the changes made to a specific directory or a project over the course of time.



# Introduction to VCS

## Why Use VCS?

- It helps you keep track of all the modifications that are made to the code.
- It helps you collaborate with other programmers efficiently.

# Advantages of VCS

**Following are the advantages of VCS over manually controlled versions:**

- Makes it easy to find out the changes between different versions of a file
- Highly efficient for large projects with thousands of files
- Makes it easy to collaborate with other programmers
- Makes it easy to store metadata
- Makes it easy to create backups and revert changes
- Makes it easy to share code on multiple systems/users

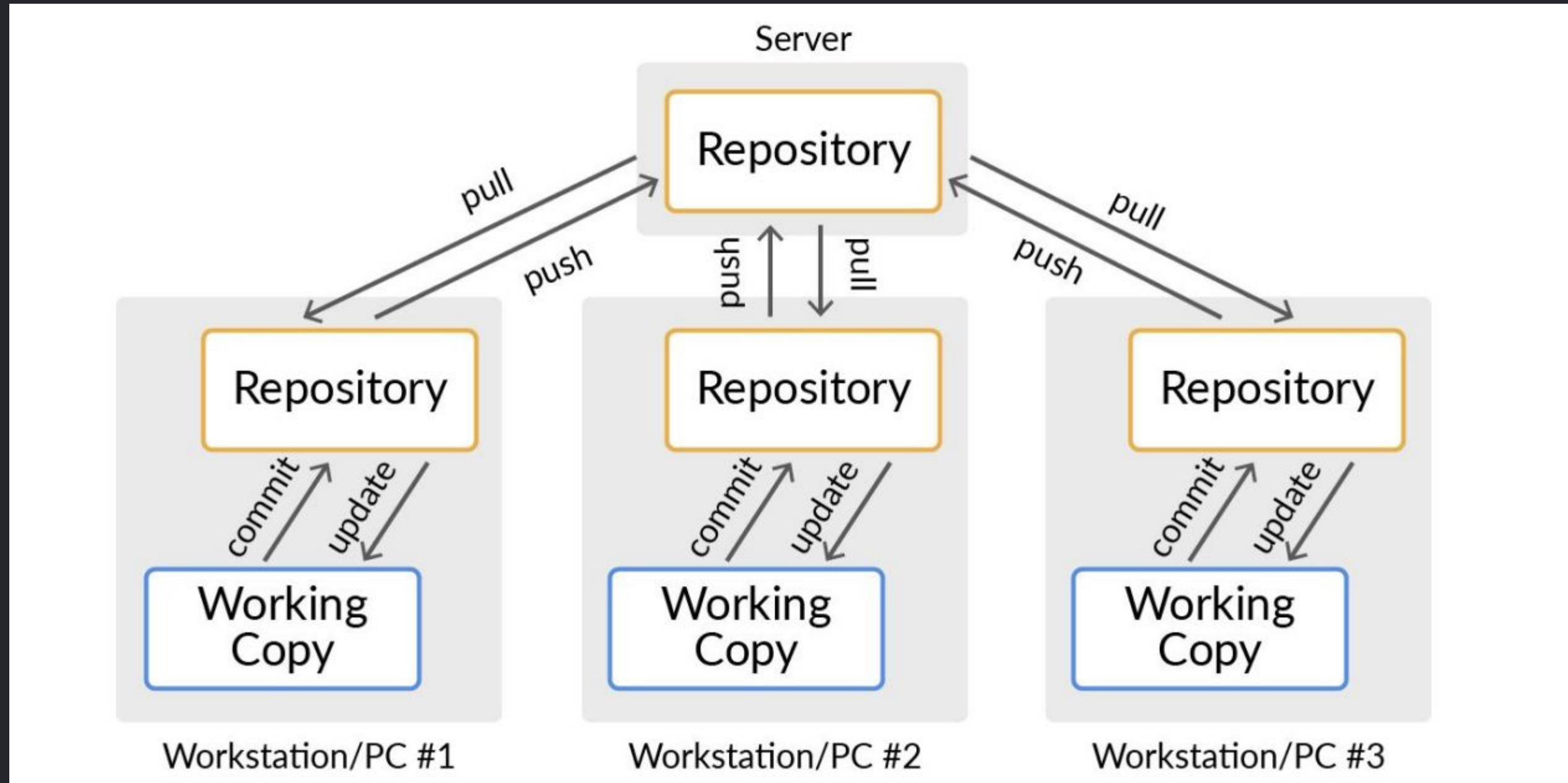


# How does the VCS Work?

## Collaboration:

- You can upload your local Git repository to a server, which includes all the versions of code. Such a repository is called a remote Git repository, which is also known as a central repository.
- Collaborators can download this remote Git repository as their local Git repo.
- They can make changes to their local Git repo and add new versions.
- They can also update the remote Git repo by pushing these newly added versions to remote Git repo.
- Other collaborators can download and update their local copies with the new changes

# How does the VCS Work?








# How does the VCS Work?

Some keywords used while discussing VCS:

- **Remote/Central/Master Repository** - A repository hosted on server/cloud service like GitHub, Bitbucket etc., where we can store the main codebase
- **Local Git Repository** - A local repository where we store all the changes before pushing to the remote repository
- **Pushing** - Pushing the changes made locally to remote repo
- **Pulling** - Pulling the current state of a project including all the updates/changes from a remote repository
- **Master** - Main branch in which all the deployable changes are stored
- **Head** - Pointer to the current branch/state of the code

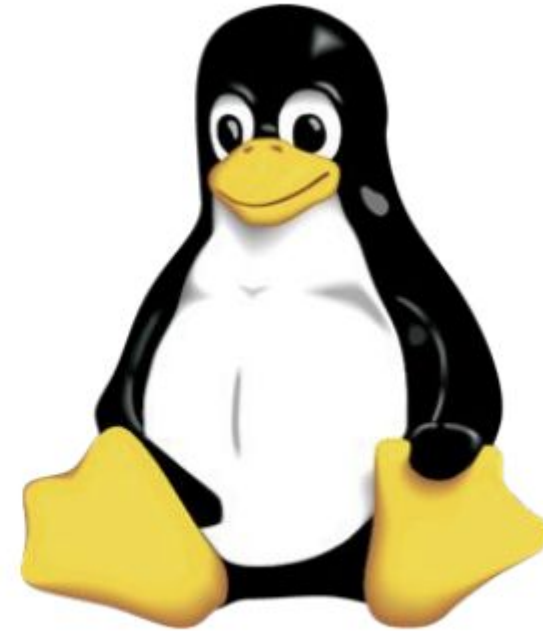
# What is Git?

Git is one of the most popular VCSs

Technology	Market Share (Est.) ⓘ	Customers ⓘ
 Git	89.46%	82,390
 Microsoft Azure DevOps Server	8.65%	7,963
 TortoiseSVN	1.48%	1,367
 Subversion	0.26%	240
 Serena PVCS Version Manager	0.14%	133

# Git Trivia

Git was created by Linus Torvalds, who was also the creator of Linux.



# Benefits of Git

- Faster as most of the operations are performed locally
- Enables flexible version control
- Easy to revert changes and move to the previous state
- Useful for distributed teams
- Maintains integrity while using hash values

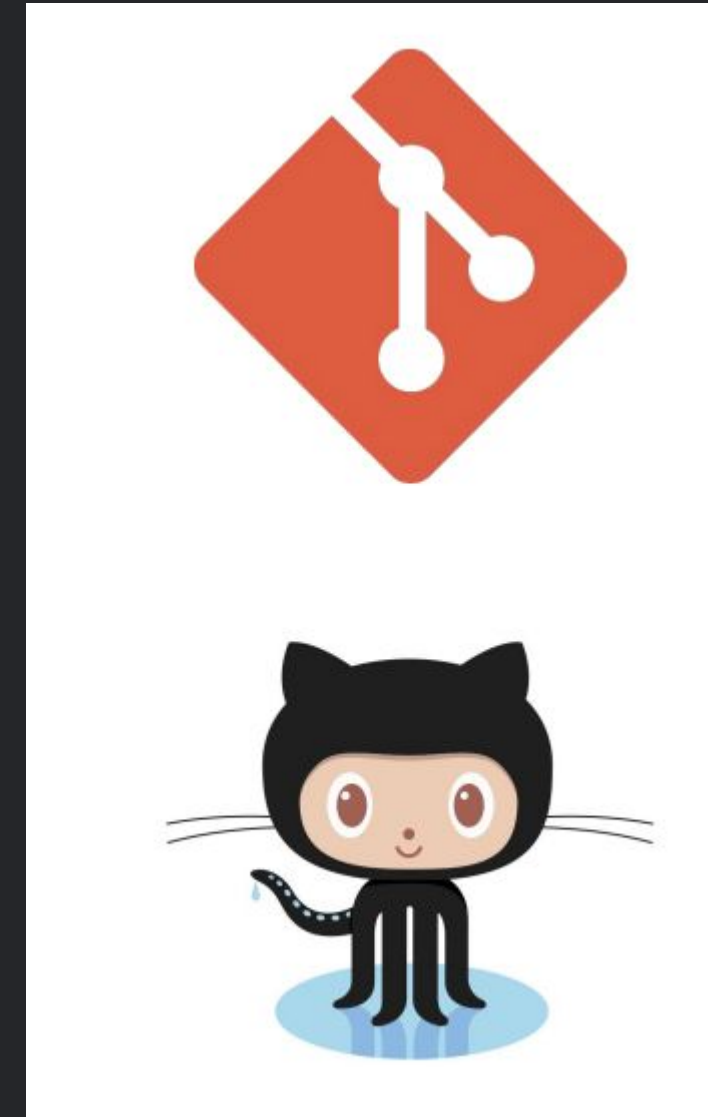
# Git vs GitHub

**Are Git and GitHub the same?**

**Answer: No**

**Git:** It is an open-source version control tool that is used by programmers for controlling versions of their codebase and collaborate with others. You can install your own Git server on your server. Most cloud providers also give it.

**GitHub:** It is the most common online Git service where we can create remote repositories based on Git.



# Alternatives to GitHub

## 1. BitBucket



## 2. GitLab



## 3. SourceForge



## 4. Most Cloud Providers



# Q1

**Which of the following is an important usage of the Version Control System?**

**Multiple answers may be correct.**

1. Testing the code
2. Checking the code functionality
3. Collaborating with the team
4. Maintaining versions

# Q1 (Answer)

Which of the following is an important usage of the Version Control System? Multiple answers may be correct.

1. Testing the code
2. Checking the code functionality
- 3. Collaborating with the team**
- 4. Maintaining versions**

# Starting with Git

# Additional Note: Vim Editor

In this session, we will be using vim editor to create and modify files.

- To create a new file and open or open an already existing file to modify -

**vim < FileName >**

- To start inserting or editing a file, you need to press **"i"**.
- To save and exit a file, you need to press **"Esc"** and then use **":wq"**.

Press 'i' to go to INSERT  
mode

Press 'esc' and then ':wq' to  
save and exit

# Additional Note: cmd Commands

In this session, we will be using the following two cmd commands.

1. cd: to change directory
2. dir: to list down the content of the directory
  - a. ls: to list down the content of the directory for UNIX systems

You can get a complete list of commands [Link](#)

# Starting with Git

You can follow either of the options given below to start with Git:

1. You can turn the local directory that is not currently under version control into a Git repository.
2. You can **clone** (copy with all details) an existing Git repository from any location or server such as GitHub and Bitbucket.

# Command - git init

## 1. Initialize an Existing Directory to Git Repo

First, Open Git Bash or Terminal and go to the project directory using the cd command.

Second, use the command - *git init*

This command will create a **.git** folder inside the current directory to convert it to a Git repository.

Most of the Git commands do not work outside of a Git repository. So, this is usually the first Git command you will run in a new project.

# Command - git clone

## 2. Clone an existing Git repository

First, Open Git Bash or Terminal and go to the project directory using the cd command.


Second, use this command: ***git clone <SSH or HTTPS URL>***

This leads to the following:

- It creates a directory named phone-directory inside the projects directory.
- It downloads the •git folder from the remote Git repository.
- It provides a working copy of the latest version of the phone-directory repository.



# Create Github Repository

 New repository

### 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 \*

Repository name \*

 PRABIR07


/

Great repository names are short and memorable. Need inspiration? How about **redesigned-succotash** ?


Description (optional)

---

☒

 **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.](#)

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.](#)

# Command - git remote add origin

## Add a new remote to your Git repository

While creating a Git repository, you need to add a remote/cloud-based repository for the following purposes:

- To keep a copy of the source code available at any time or location
- To collaborate with team members and other programmers across multiple locations.

You can add a remote to your Git repository using the following command:

***git remote add origin <SSH or HTTPS URL>***

We can have multiple remote locations (remote repositories) added to our local Git repository. This command adds a new remote location with the name 'origin' and connects to the given link of the remote repository.

# Command - git config

## Set username and email to Git bash

You can set username and email to your Git bash by executing the following commands:

```
git config --global user.name "<your name>"  
git config --global user.email <your email>
```

You can check your username and email by executing the following commands:

```
git config user.name  
git config user.email
```

## Q2

**Which of the following commands will help you to download a new Git repository from the remote?**

1. git init
2. git clone
3. git remote add origin
4. None of the above

## Q2 (Answer)

Which of the following commands will help you to download a new Git repository from the remote?

1. git init
- 2. git clone**
3. git remote add origin
4. None of the above

# Making Changes in the **Local Git Repo**

# Making Changes in the Local Git Repo

To understand how Git works, you need to understand the three states and areas of a Git project:

## 3 States

1. Modified, Untracked
2. Staged
3. Committed

## 3 Areas

1. The Working Tree
2. The Staging Area
3. The Git directory

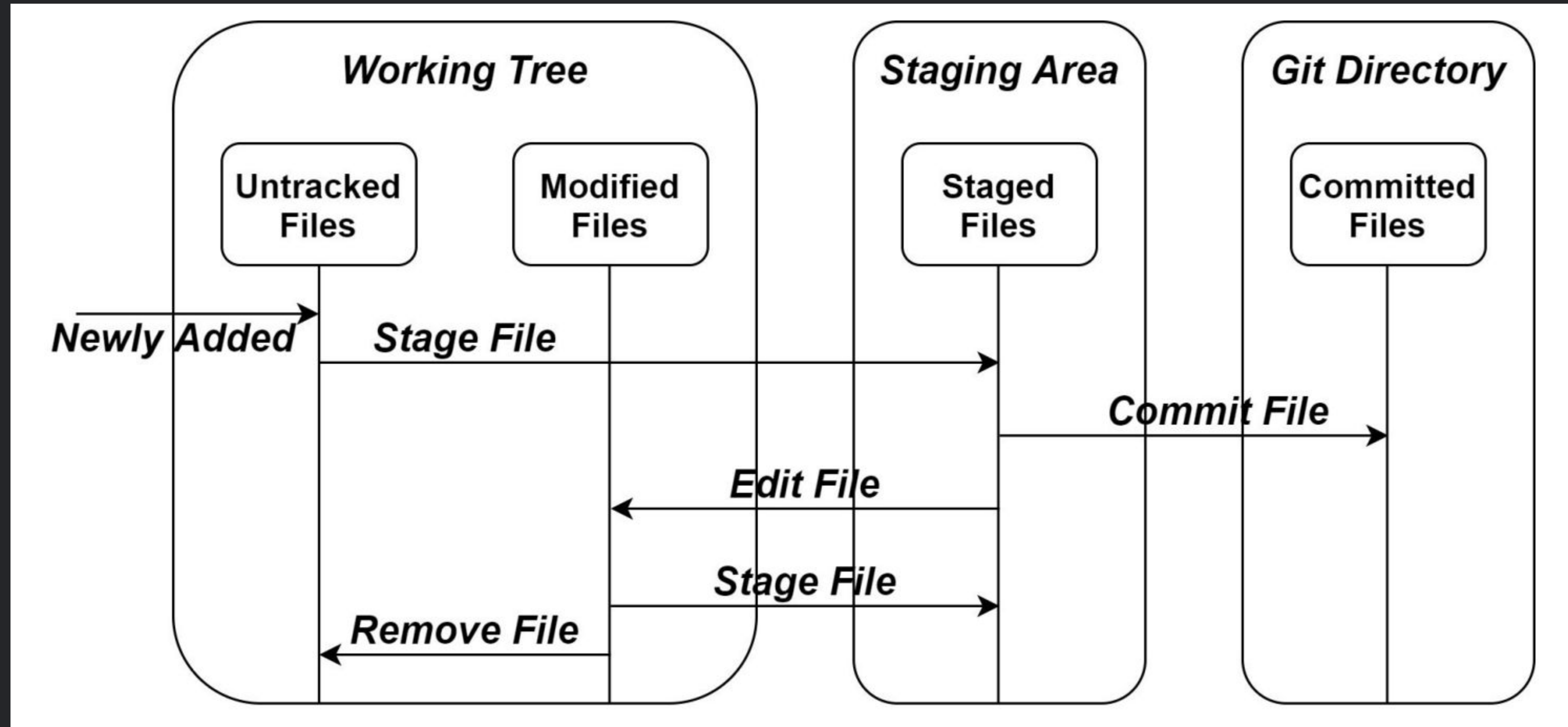
# Making Changes in the Local Git Repo

## The Git Workflow

1. When you add new files to the local Git repo, those files are called **untracked files** and reside in the **Working Tree**.
2. The files you want to save in the next version need to be moved from **Working Tree** to **Staging Area**. Such files are called **staged files**.
3. You can save the staged files in the next version by moving them to the Git directory. These files will be called **committed files**.
4. If you make changes to the staged files before moving them to the Git directory, they are moved back to the **working tree** and are called **modified files**.



# Making Changes in the Local Git Repo



# Command - git status

- Used to check the status of the working tree and the staging area
- Shows the status of all the untracked and tracked files (whether staged or unstaged)

Syntax: **git status** or **git status -s**

# Additional Note : Symbols in short Status

In the short status, each file is appended with a symbol in the form of XY, where X shows the status of staging area and Y shows the status of working tree.

## Different symbols

1. ' ' - Unmodified
2. M - Modified
3. A - Added
4. ?? - Untracked

## Different combinations

1. A - Added to staging area and not modified (tracked + staged)
2. AM - Added to staging area but again modified (tracked + unstaged)

You can see the complete list of symbols and combinations [here.](#)

# Command - git add

- This command is used to move the unstaged files (both tracked and untracked) files from the working tree to the staging area.
- The files present in the staging area will be present in the next commit.

## Syntax:

- ***git add <fileName1> <fileName2 >*** - Used to stage the two files named fileName1 and fileName2
- ***git add - -all*** or ***git add .***

Used to move all the files to the staging area

# Command - git commit

- It is used to commit/save all the changes to the local Git repo.
- It is used to move staged files from staging area to git directory.

## Syntax:

- ***git commit -m <message>***: Commits all the staged files with the given commit message
- ***git commit -a -m <message>***: Used to stage all the tracked files and commit them with the given message. This command lets you skip the step of executing git add command.

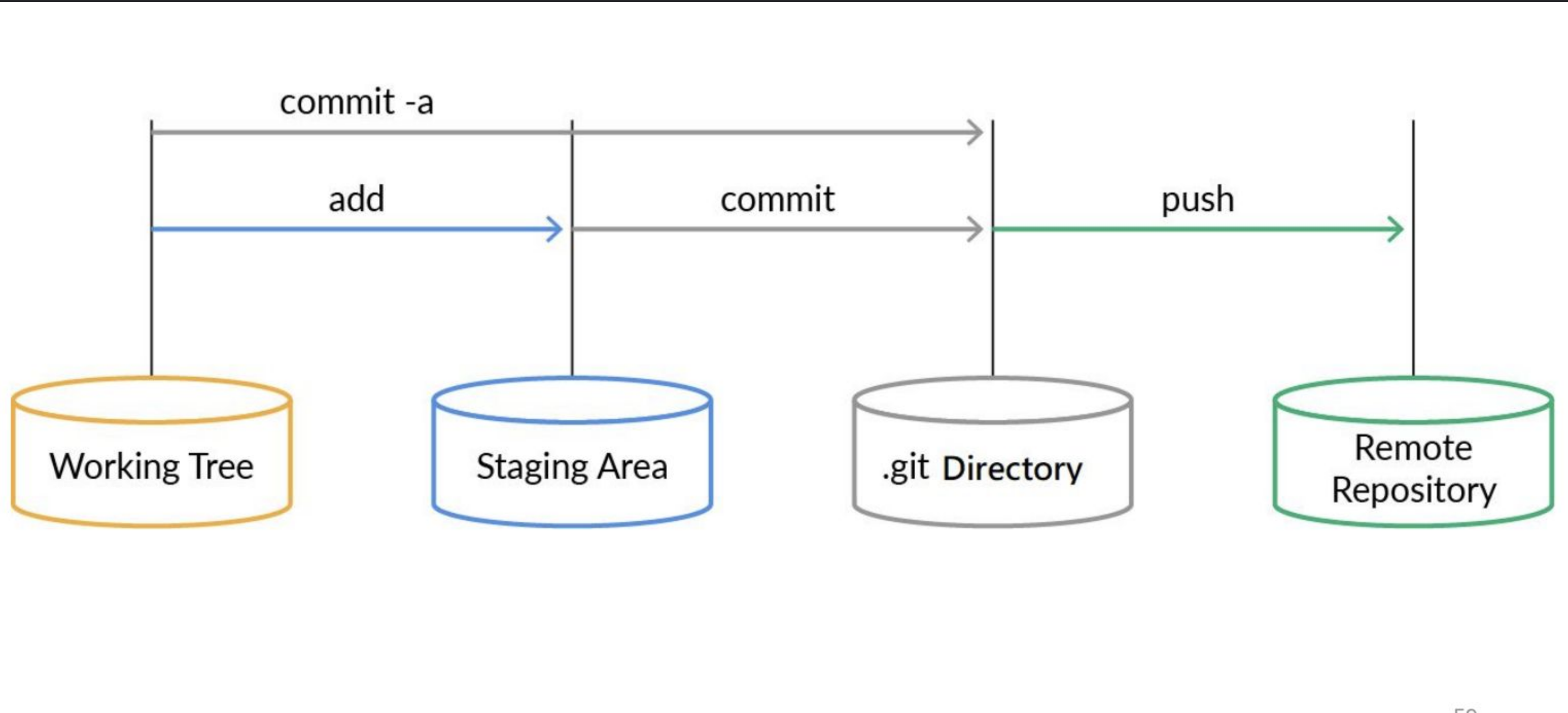
# Command - git log

- Helps to view information about previous commits.
- We can filter commits based on multiple properties such as number of days, author, recent number of commits done.

## Syntax:

- ***git log --all*** OR ***git log***  
shows details of all the commits
- ***git log -n***: shows details of last 'n' number of commits
- ***git log --committer="<name>"***: shows details of the commits made by "name"
- ***git log --oneline***: shows details about each commit in one line

# Flow



# Branching in Git



# Why do we need Branching in Git ?

**Scenario:** You have a running live website, and you want to add some new features to it.

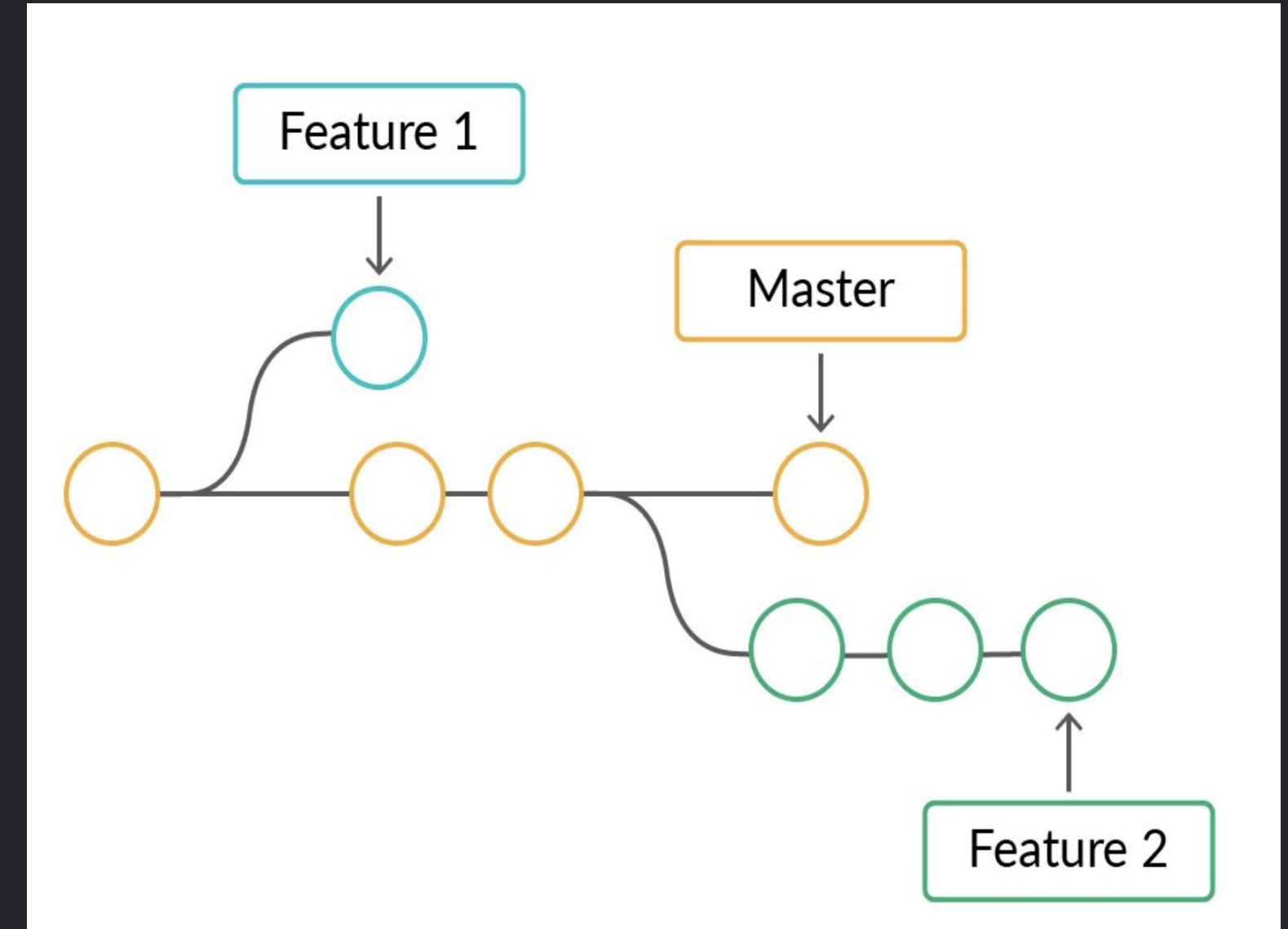
**Approach 1:** Stop running the website and then make changes to the code in the master codebase.

**Approach 2:** Make changes to the code in the master codebase of the running website.

*Is there a way in which the master codebase of the website remains untouched until you complete the updation in the code of the new feature?*

# Branching - Introduction and Need

- Branching is a Git feature that helps you create new branches.
- You need to create new branches to add or fix changes in the source code, without affecting the main codebase.
- When you finish making the changes, you can merge those branches with the master branch and update the main codebase.



# Command - git branch

- Used to create new branches
- Used to delete or list down the existing branches

## Syntax:

- **git branch:** Used to list down all the existing branches and highlight the current branch
- **git branch <branchName>:** Used to create a new branch with the given branch name
- **git branch -d <branchName>:** Used to delete the branch with the given branch name

# Command - git checkout

- Used to navigate between branches
- Used to create and simultaneously checkout the new branch
- Used to move to a previous version in the same branch

## Syntax:

- ***git checkout <branchName>***: Used to checkout the branch with the given branch name
- ***git checkout -b <branchName>***: Used to create a new branch with the given branch name and simultaneously checkout to the newly created branch
- ***git checkout <commit id>***: Used to checkout the code to the commit ID provided

# Merge Conflicts

## What are merge conflicts in Git?

- A merge conflict occurs when you are merging two branches and Git is unable to automatically resolve the differences in code/files between the two branches.

## When does a merge conflict occur?

- A merge conflict occurs when two different changes are made to the same line in the same file.

For example, if one user deletes a line and another user modifies the same line on a different branch, then a merge conflict occurs.

And later, when two branches having different changes on the same line are to be merged, merge conflict occurs.

# Merge Conflicts

## How to resolve merge conflicts?

- Git displays merge conflict areas in the file between the conflict markers as shown below.

```
<<<<<< HEAD
```

```
<Changes in the current branch>
```

```
=====
```

```
<Changes in the incoming branch>
```

```
>>>>>> branchName
```

- We can resolve the conflict manually by editing the piece between the markers.
- We should also remove the markers added by Git.
- We can then commit the changes and, thus, fix the merge conflict.

# Merge Conflicts

**Git commands are used to resolve merge conflicts.**

You can resolve merge conflicts by editing the file and removing conflicts, but some Git commands can help you in a better way. These include the following:

- ***git status***: Gives an idea of merge conflicts that have occurred.
- ***git log --merge***: Produces a log with all the commits that have caused conflicts during the merging of branches.
- ***git diff***: Finds the differences between the states of a repository/files  
This can help you find and prevent merge conflicts.



# Merge Conflicts

Use the following Git commands to fix merge conflicts:

***git checkout:*** Used to undo changes made to unstaged files (Recall that this is also used for switching to branches or commits.)

***git reset:*** Used to undo changes made to the working directory and the staging area to the last stable commit. This basically works for the set of files in the aforementioned areas unlike git checkout, which can even work on specified files.

***git merge --abort:*** Used to exit from the merge process and return the branch to the state prior to the start of the merging process



# Syncing with the **Central Repository**

# Command - **git clone**

- Used to copy a remote repository to a local repository

## Syntax:

- **git clone <urL>:** Used to clone a repo from the provided remote repo, it will create a directory with the name of the remote repo.
- **git clone <url> <directory\_Name>:** Used to clone a repo from the provided remote repo and save it in a new directory with the name directory\_Name

# Command - git pull

Used to update all the changes from a remote repository to the local repository

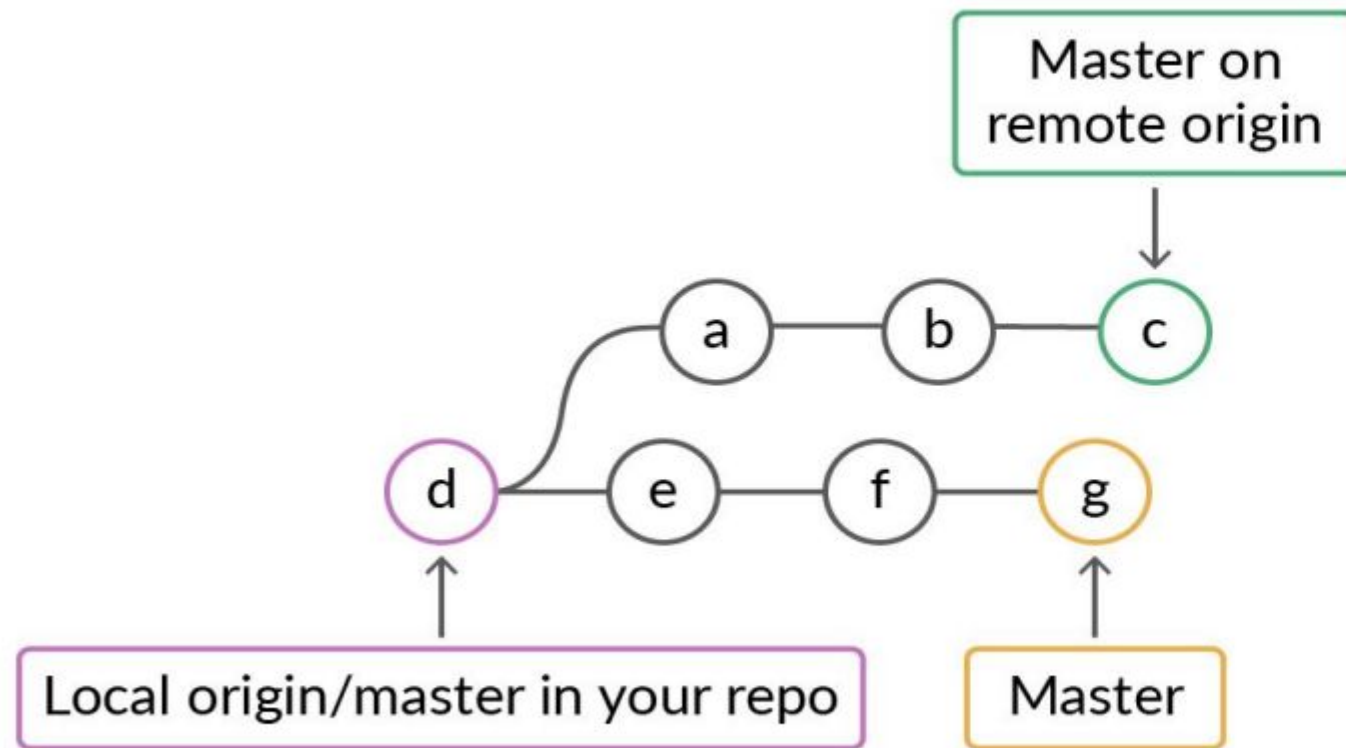
## Syntax:

***git pull <remote> <branch>***: Pulls updates from the given remote repo and branch and merges them with the current branch.

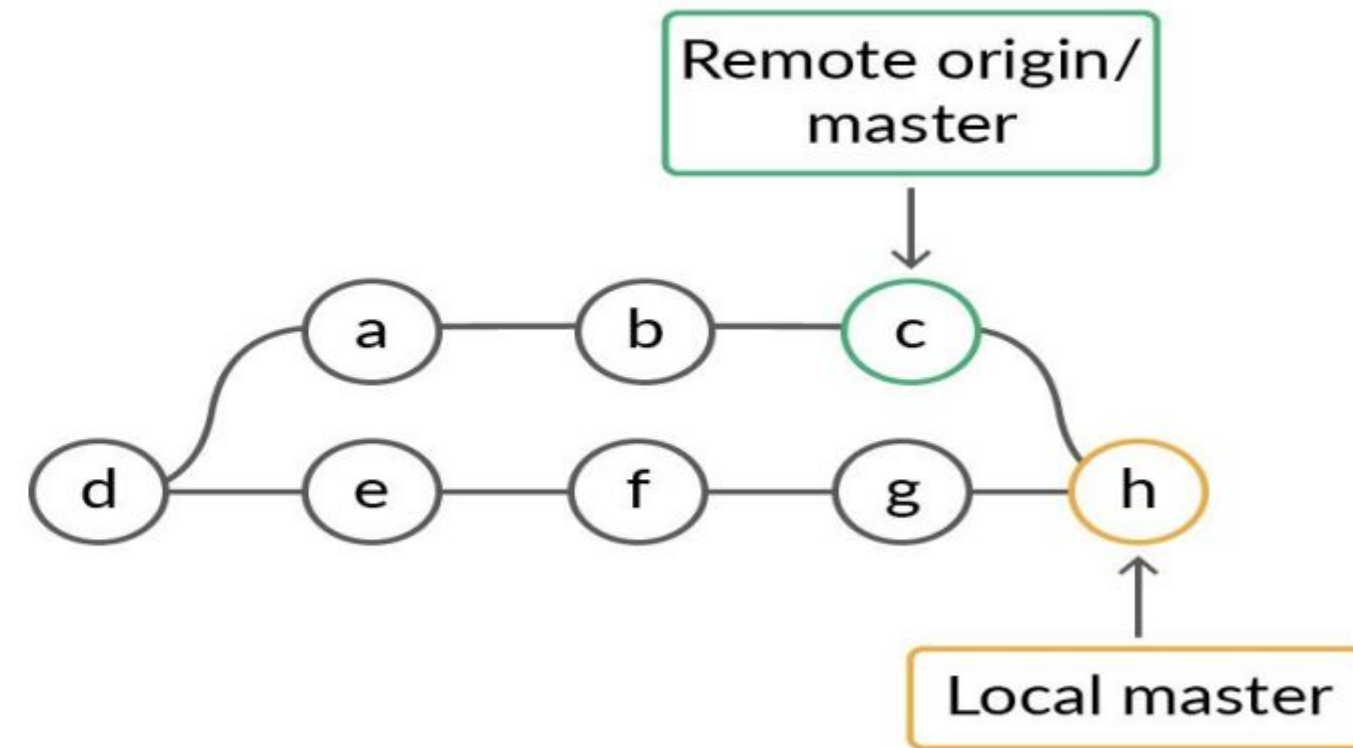
***git pull <remote>***: Pulls updates from the given remote repo and master branch and merges them with the current branch.

***git pull***: Pulls updates from the given origin and master branch and merges them with the current branch.

# Command - git pull



Before git pull



After git pull

# Command - git fetch

- Used to download all the changes from a remote repository to the local repository
- Also, used to check all the updates in the remote repository

## Syntax:

***git fetch <remote>***: Fetches all the data from the given remote repo link

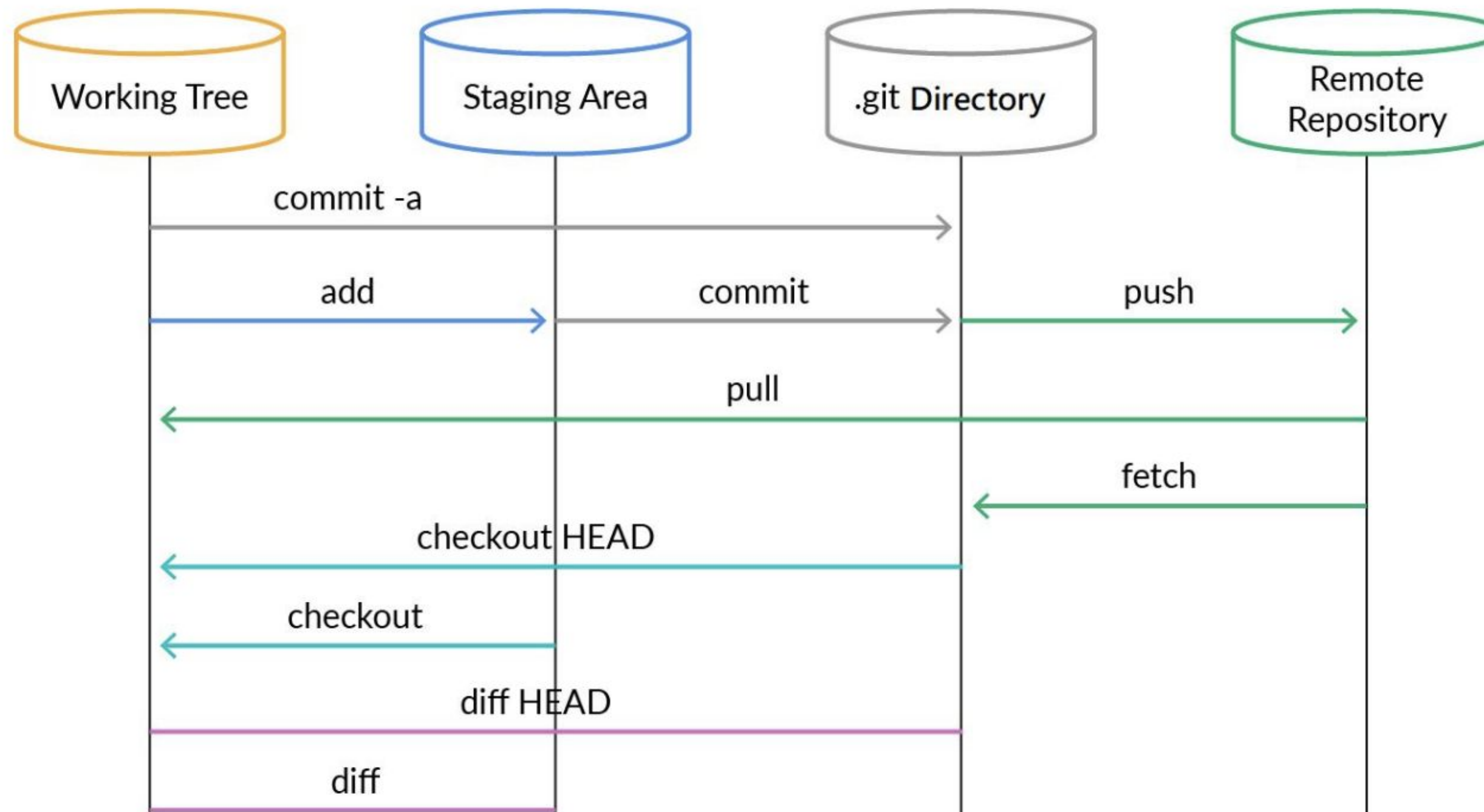
***git fetch <remote> <branch>***: Fetches all the data from the given remote repo link and the specified branch

***git fetch --all***: Fetches all the data from all the remote locations added in the local repository.

# git fetch VS git pull

- The git pull command is a combination of two other commands, git fetch followed by git merge.
- For example, suppose you are currently in the feature 1 branch and execute git pull origin feature2 command.
- In the first stage of the operation, git will execute git fetch origin feature2 command and download all the changes from the remote repository to the local repository corresponding to the feature2 branch.
- Once the content is downloaded, git will execute git merge feature2 command and merge the feature2 branch with the current branch, which is feature1, if there is no merge conflict.

# Final Flow



# .gitignore file

- Used to ignore untracked files.
- For example, suppose you do not want to include the log files with .log extension. You can achieve this using the gitignore file.
- In the **.gitignore** file, you have to enter a line with the following pattern.
  - \*.log
- Now, all the files with the .log extension will be ignored by the Git.
- This file should be present at the root level (the same folder in which git folder is present).
- You can see the complete list of rules for writing patterns in the gitignore file [here](#)



Thank  
You