Git

What is Version Control?

- A version control system (VCS) allows you to manage a collection of files and gives access to **different versions of these files**.
- The VCS allows you to capture the content and structure of your files at a certain point in time.
- **Allows you to revert any changes and go back to a previous state.**
- Each version captures a snapshot (**point in time change**) of the files at a certain point in time and the VCS allows you to switch between these versions.
- The different versions are stored in storage system which is typically called a **repository**.
- Allows you to track the history of a collection of files.
- Allows collaborative development, and allows you to know who made what changes and when.

Types of Version Control System

- Version control systems are the process management systems that keep **tracks of every individual modification to the code by every contributor.**
- Ability to have as many developers working on the same code base.
- Provide merging and tracking capabilities of the recorded changes.
- Revert back your files incase there is some problem.
- Allow you to share those changes

Types of VCS

- **Centralized Version Control Systems (CVCS)**
- **Distributed Version Control Systems (DVCS)**

Localized Version Control Systems

- A localized version control system keeps local copies of the files. This approach can be as simple as creating a manual copy of the relevant files.

  Major drawback : **Single point of failure** of the local computer.
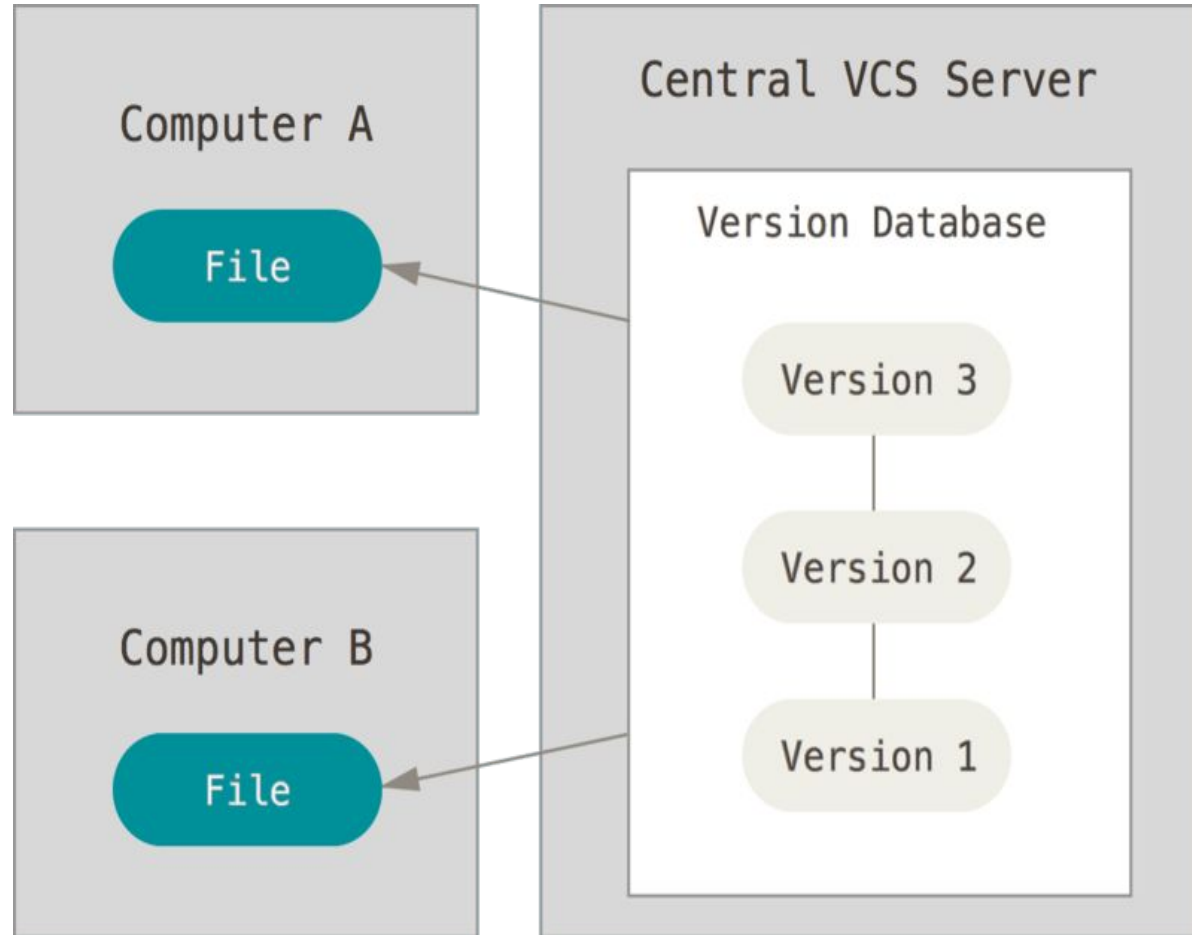
Centralized Version Control Systems

- Centralized version control system (CVCS) uses a **central server to store all files** and enables team collaboration.

  Major drawback : **Single point of failure** of the Central Server.

- A centralized version control system provides a server software component which stores and manages the different versions of the files. A developer can copy (checkout) a certain version from the central sever onto their individual computer.
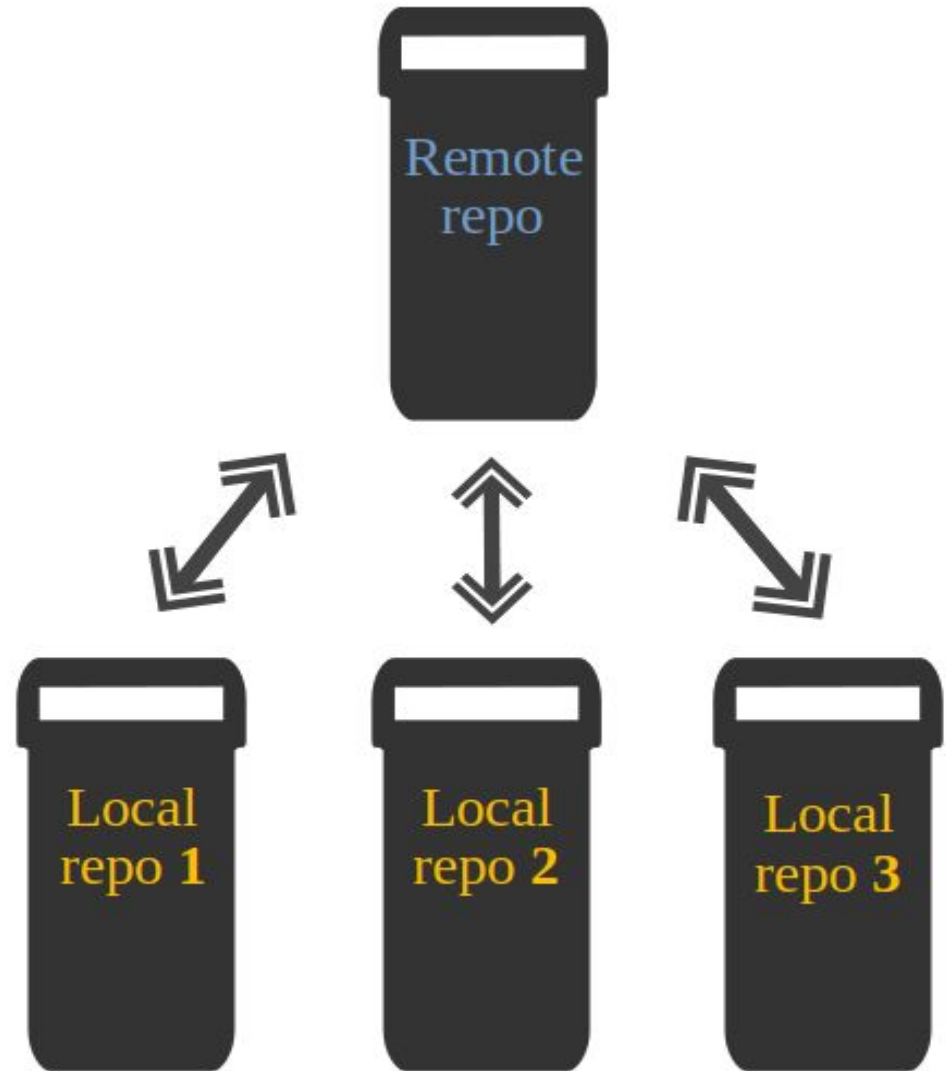
Centralized Version Control Systems

- Centralized version control system (CVCS) uses a **central server to store all files** and enables team collaboration.
- major drawback of CVCS is its **single point of failure.**

Distributed Version Control Systems

- **Distributed** means it has a remote repository which is stored in a **remote server** and a local repository which is stored on the **local computer** of every developer working on a project.
- **Every developer has a full copy** of the repository on his local computer.

What is Git?

- **Git** is currently the most popular implementation of a **distributed version control system (DVCS)**
- Git is an Open Source Project that originates from the Linux kernel development and was founded in 2005 by **Linus Torvalds**.
- Nowadays it is used by many popular open source projects, e.g., the Android or the Eclipse developer teams, as well as many commercial organizations.
- Imagine **git as something that sits on top of your file system** and manipulates files.
- This something is a **tree structure** where **each commit creates a new node in that tree**.

Git Repositories

- A Git repository contains the history of a collection of files starting from a certain **directory**.
- Git has two repository types:
- **Local Repository**
  - Local Repository is on your **own machine**, so you have direct access to it.
- **Remote Repository**
  - Remote Repository is usually a centralized server.
  - Remote Repository storage provided by **Source Control Management Providers**:
    - Github - **www.github.com**
   - AWS CodeCommit
    - BitBucket

Git Repositories
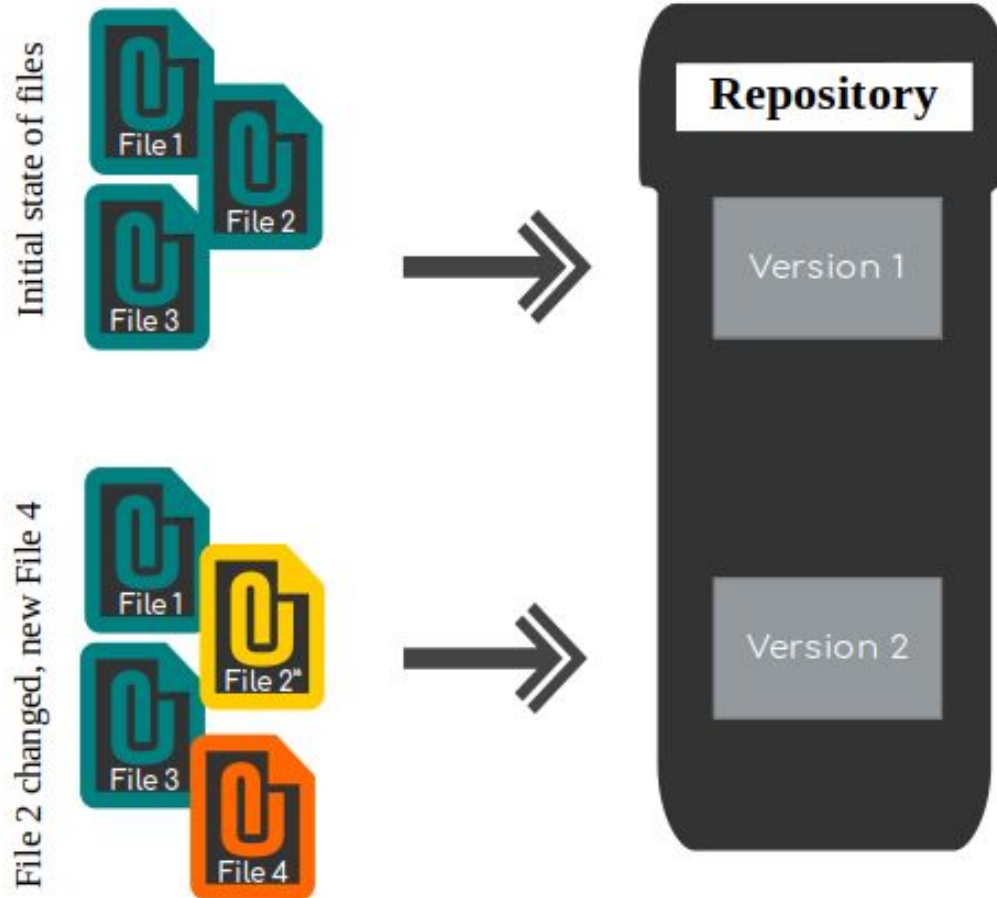
Working tree

- The Git **working tree** is comprised of files in their present state as they exist on the file system.
- The user can change the files in the working tree by modifying existing files and by creating and removing files.
- A file in the working tree of a Git repository can have different states.
  - **untracked**: the file is not tracked by the Git repository. This means that the file was never **staged** nor **committed**.
  - **tracked**: committed and not staged.
  - **staged**: staged to be included in the next commit.
 - **modified**: the file has changed but the change is not staged.

Git Staging and Committing
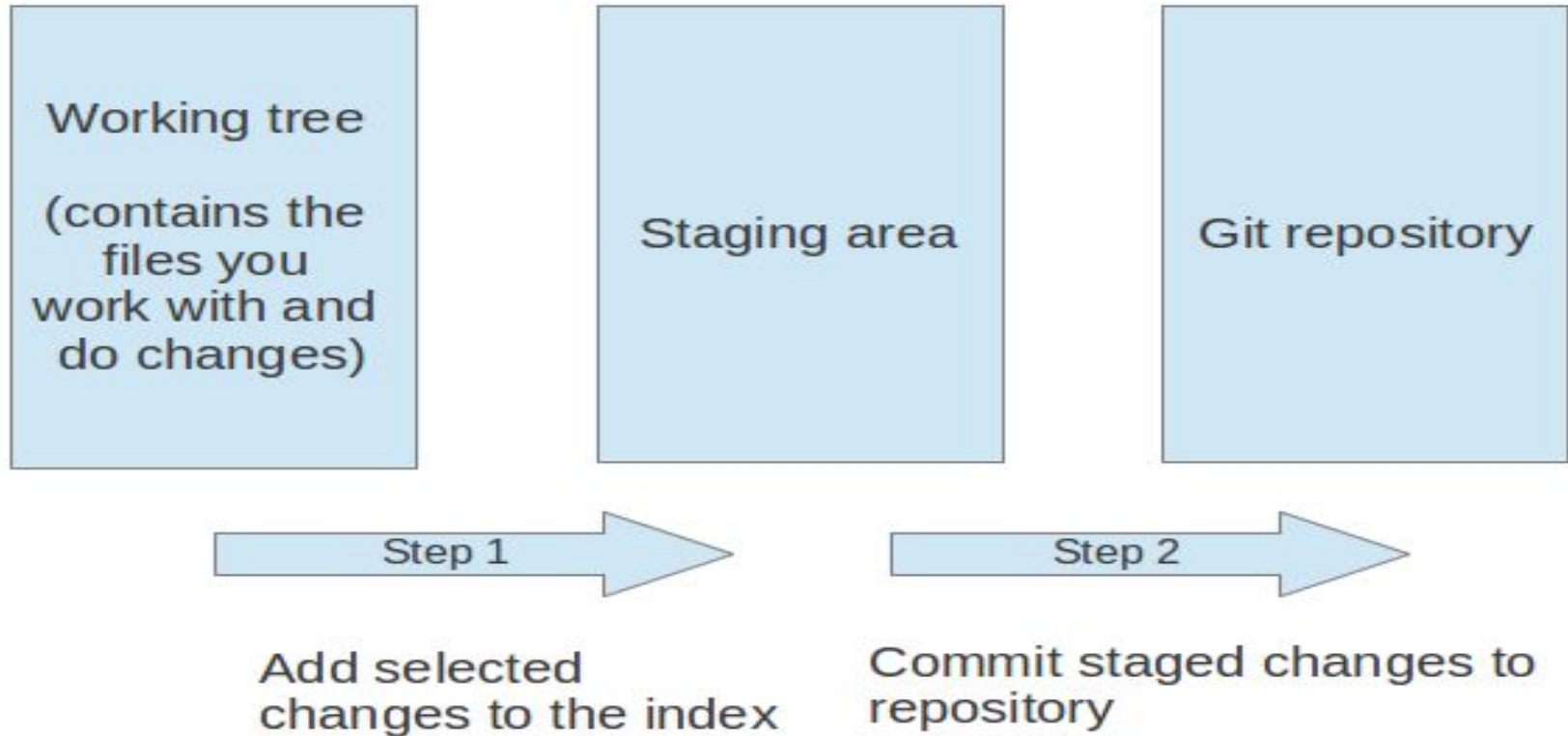
Git Staging and Committing

- The **git add** command stores a snapshot of the specified files **in the staging area**.
- It allows you to incrementally modify files, stage them, modify and stage them again until you are satisfied with your changes.
  - After adding the selected files to the staging area, you can **commit** these files to add them permanently to the Git repository (Local Repo).
  - Committing creates a new persistent snapshot (called **commit or commit object**) of the staging area in the Git repository.
  - A commit object, like all objects in Git, is immutable.
  - The **staging area** keeps track of the snapshots of the files until the staged changes are committed.
  - For committing the staged changes we use the **git commit** command.

3 Steps of git
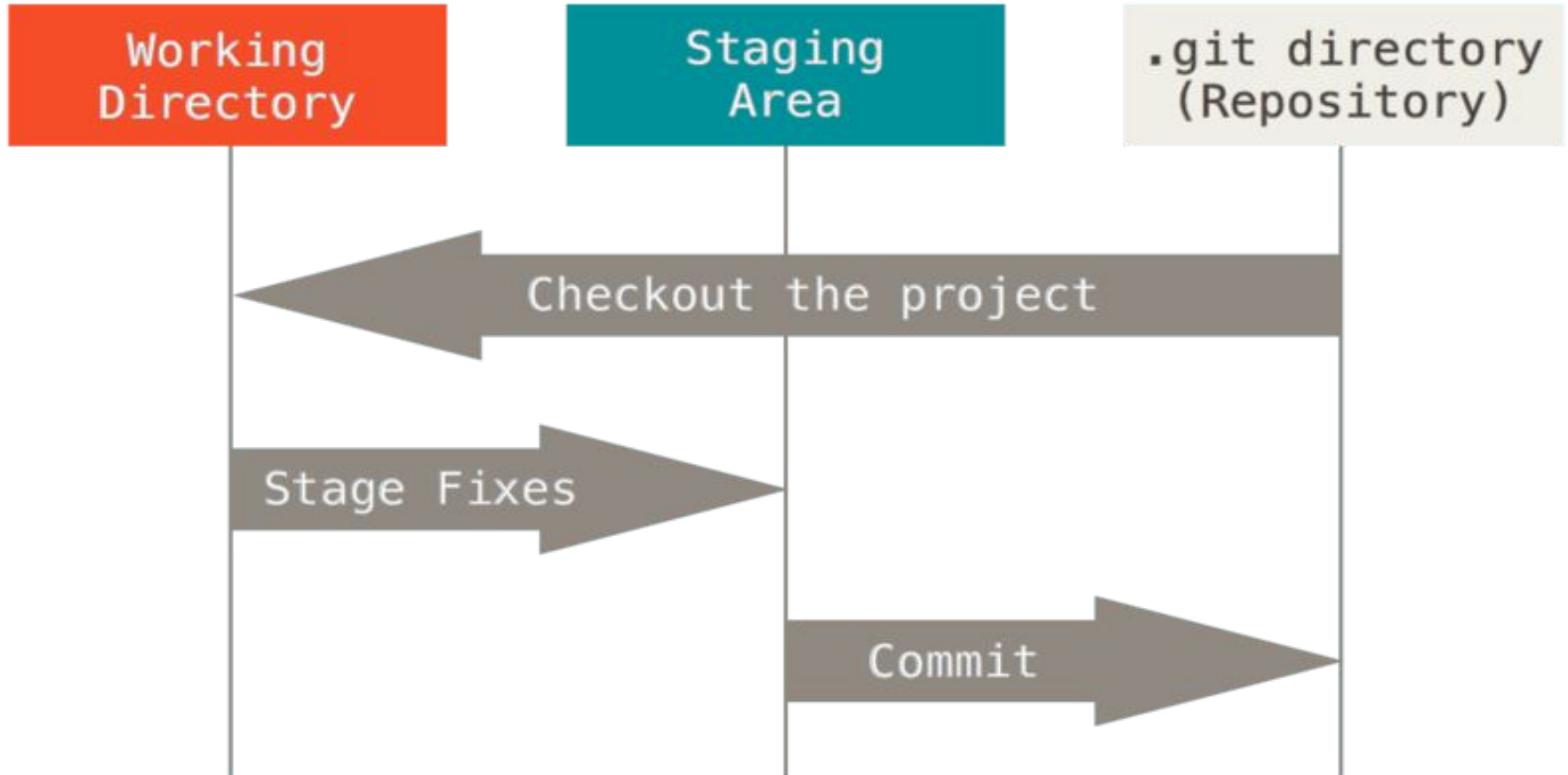
The Three States

- **modified**

  Modified means that you have changed the file but have not committed it to your Git Repository yet.

- **staged**

  Staged means that you have marked a modified file in its current version to go into your next commit snapshot.

- **commited**

  Committed means that the data is safely stored in your local repository/database.

What is a Commit?

- A commit represents a **safe point** in a project's history.
- It is used to record the changes in the repository.
- A commit object mainly consists of three things:

  - **Commit ID (SHA-1 Hash)**

  - **Author:** git config user.name and user.email

 - **Commit Message** describing reason for the changes

    A **hash**, a 40-character string that uniquely identifies the commit object

HEAD : It is a pointer to last commit in the **current checkout branch**.

- A git repository contains, mainly **Set of commits**

3 Steps of git

- **Introduce a change**
  introduce a change to a file that is being tracked by git
- **Add the actual staging area**
  Add the change you actually want using **git add**
- **Commit**
  Commit the change that has been added using
  git commit -m "Initial Commit"

Working directory

Git add operation

Staging area

Git commit operation
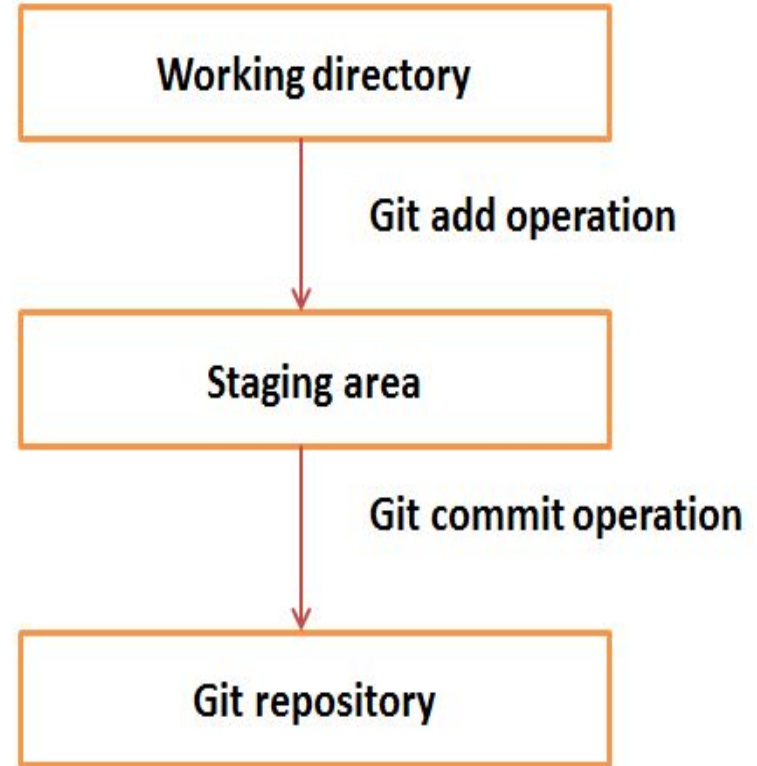
Git repository

Git Setup - Local Repository

- You can take a **local directory** that is currently not under version control, and turn it into a Git repository.If you have a project directory that is currently not under version control and you want to start controlling it with Git, you first need to go to that project's directory.
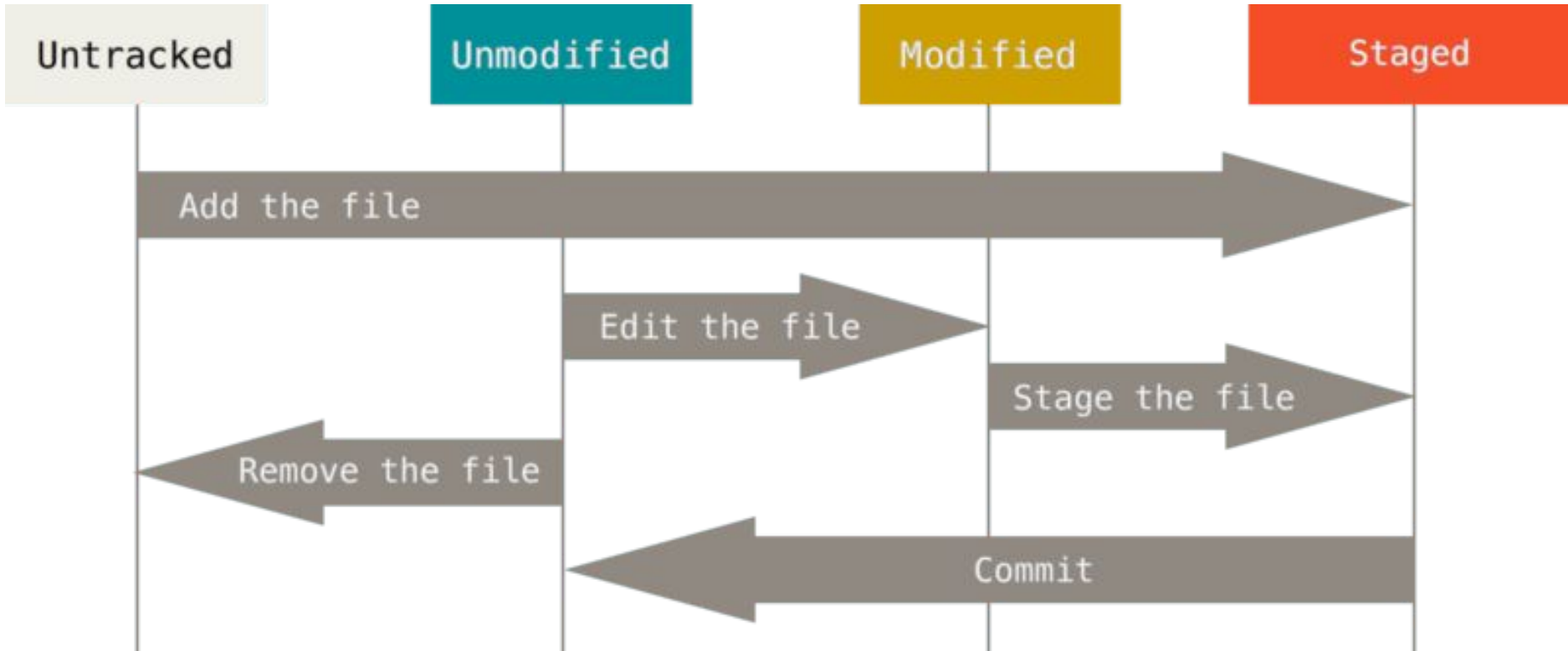
$ git init

- Makes any directory into a Git repository.
- You can **clone** an existing Git repository from elsewhere.

$ git clone *<GIT_REPO_URL>*

- Clone URL is of two type:
    - **https** : https://github.com/cloudmlops/git-demo-practical.git
        - Requires Username and Password
    - **ssh** : git@github.com:cloudmlops/git-demo-practical.git
        - Requires SSH Keys (Public Key and Private Key)
- A clone url will always have **.git**

Changes to a Repository

Basic git commands

- git init -> Initialize a directory as a local repository

- git add -> Add file from working area to staging area

- git status -> current state of files in the directory

- git commit -> commits changes present in staging area.

- git log -> Display commit information

- **git clone https://github.com/boto/boto3.git ->** This will clone remote repository into local.

git working

- **origin** refers to the source repository from where it was cloned.
- **HEAD** is a reference to the last commit in the currently checked-out branch.

- <u>git diff</u> : View difference between Stage and Working Directory
- <u>git diff --staged</u> : View difference between HEAD and Staged
- <u>git diff HEAD</u> : View difference between HEAD and Working Directory

- Staged and index both are same
- Unstaged changes exist in our Working Directory, but Git hasn't recorded them into its version history yet.
- Staged changes are a lot like unstaged changes, except that they've been marked to be committed the next time you run git commit
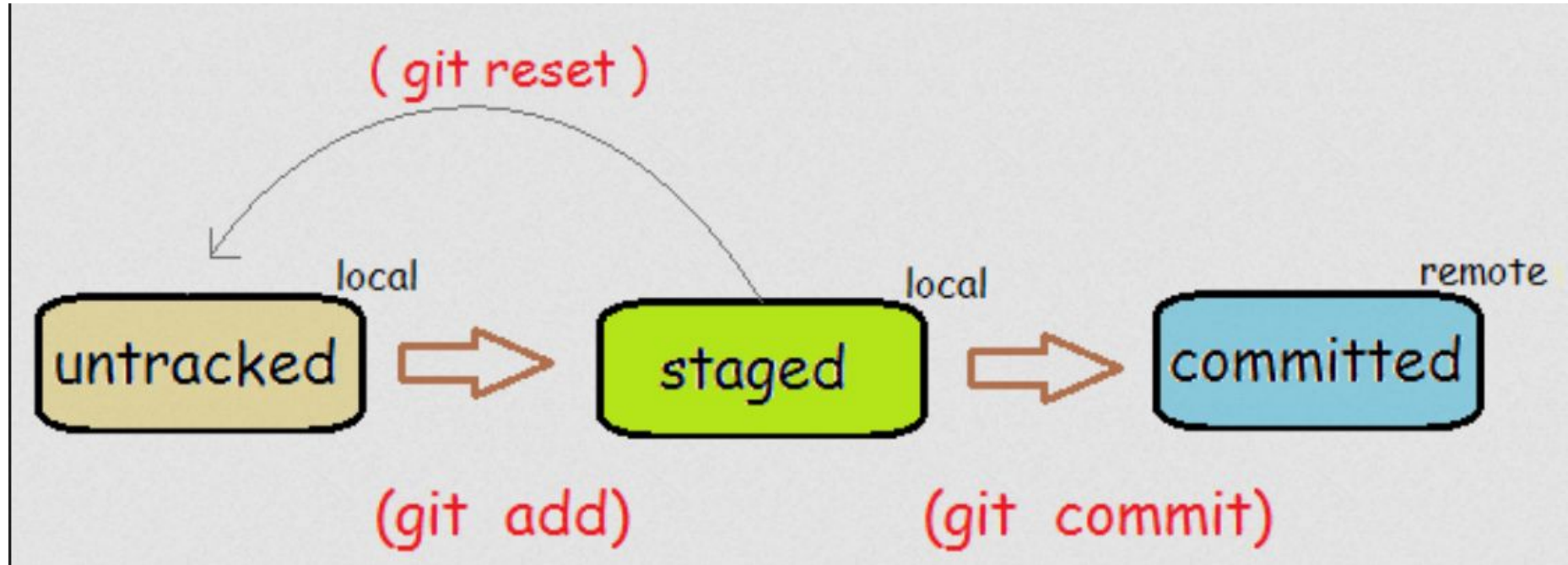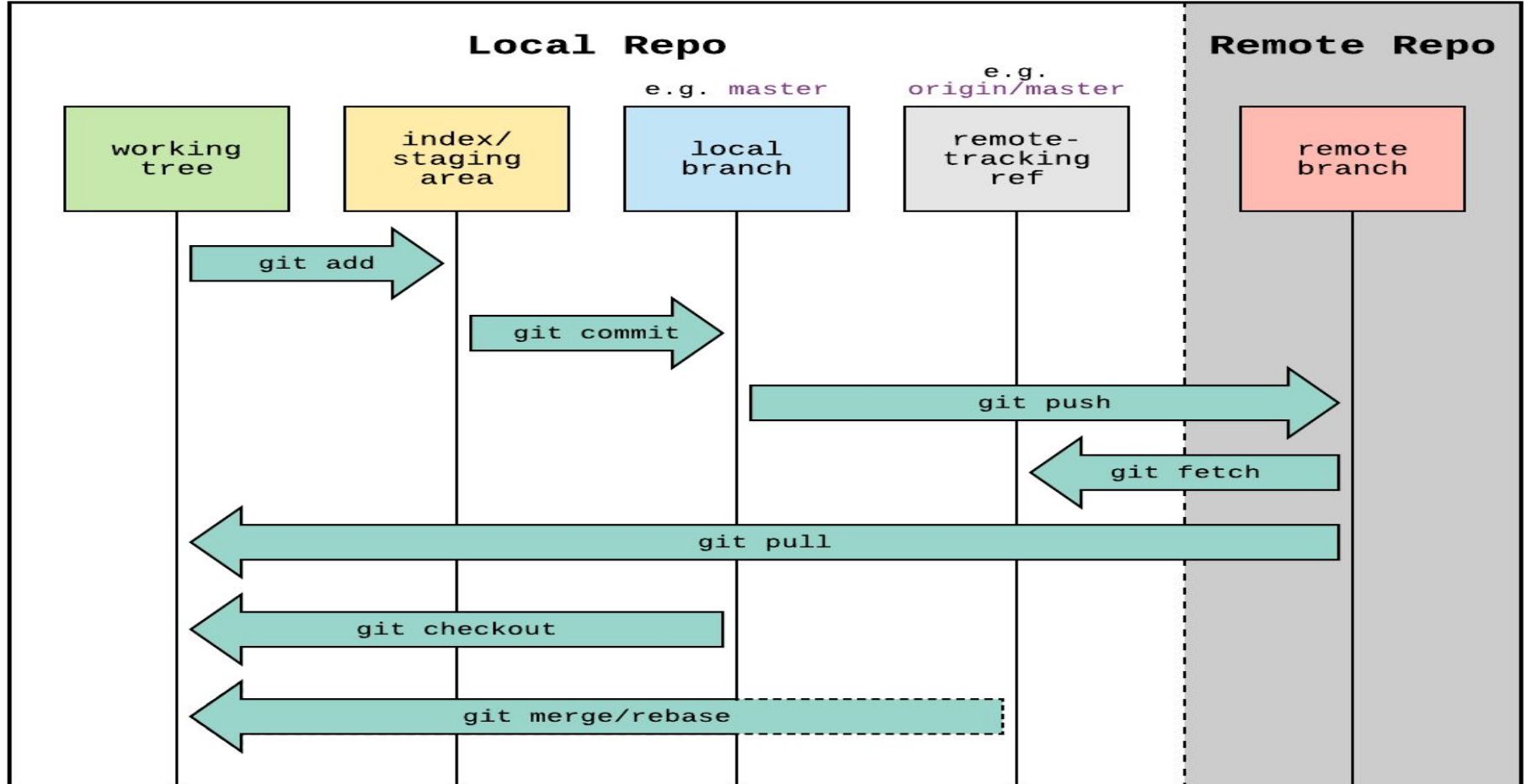
git working
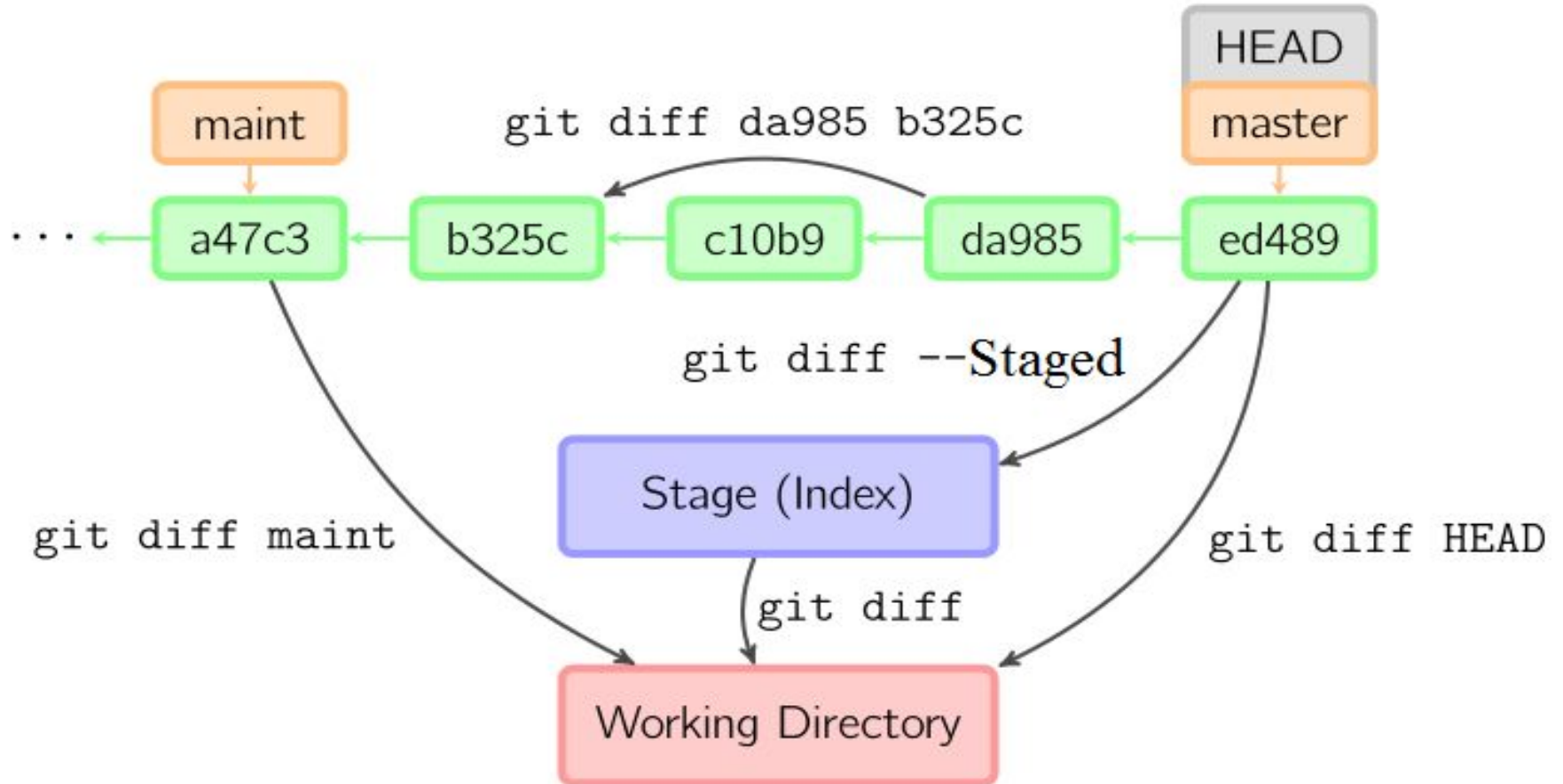
Git commands Workflow
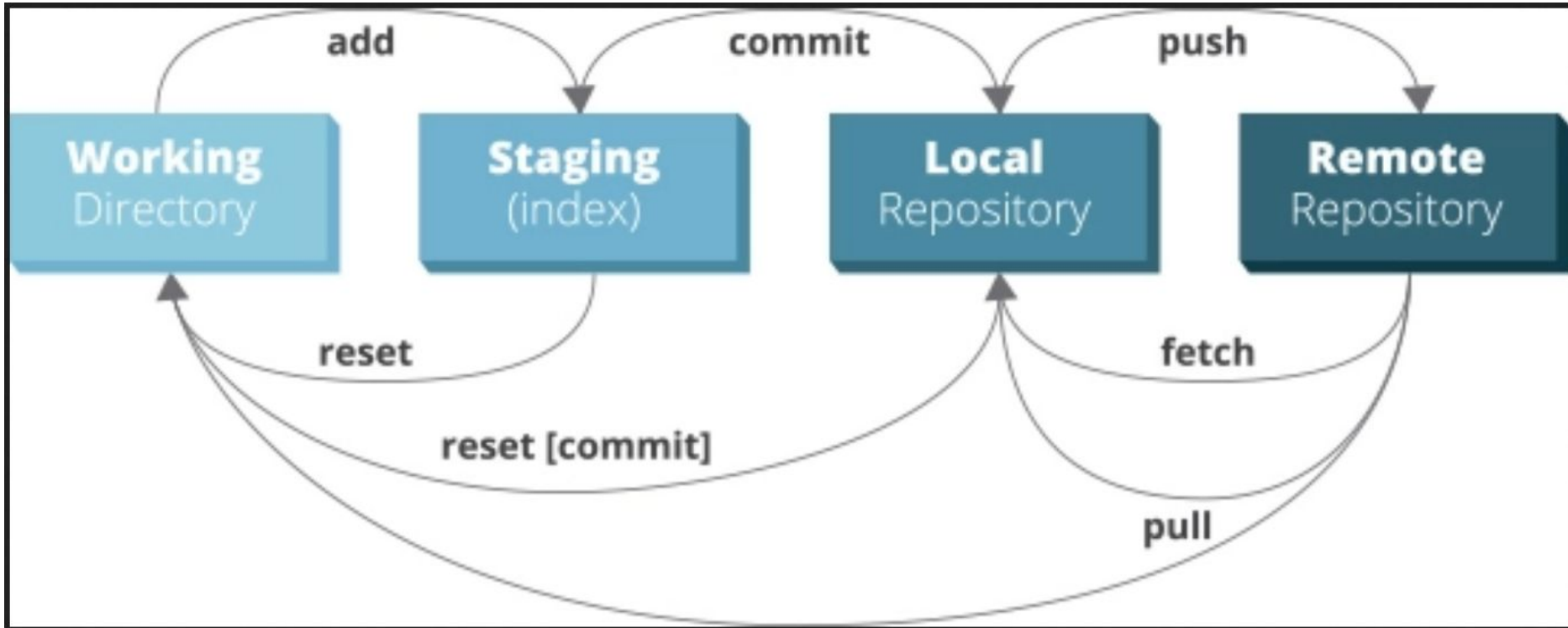
Git commands Workflow

Git commands Workflow

.git directory structure

- ├──.git
- ├── HEAD/ (A pointer to your current branch)
- ├── config/ (contains all configuration preferences)
- ├── description/(description of your project)
- ├── Index/ (is used as staging area between working directory and repo)
- ├──logs/ (keeps records to changes that are made in ref)
- ├──objects/ (all data are stored here: commits, trees and tags)
- ├── hooks/ (shell scrips that are invoked after executing a command)
- ├──refs/ (holds your local branch remote branch and tags)

A typical Git branching Model
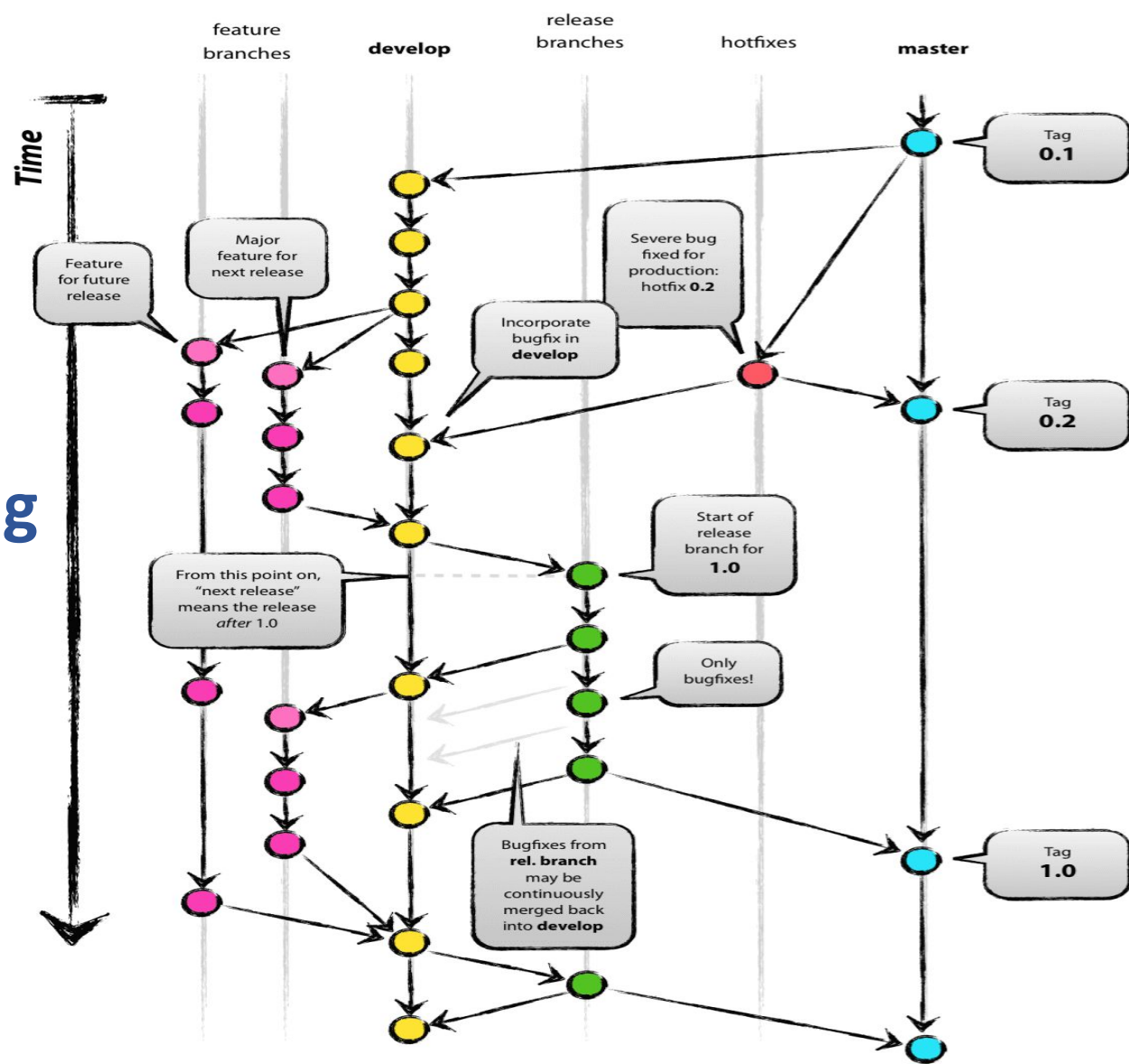
Git Remote

- A Git remote is a connection to another repository.

- When you set a remote, you can push code to and pull code from a remote repository.

- A remote repository could be hosted on GitHub, BitBucket, GitLab, or another version control platform.

Working with Git Remote
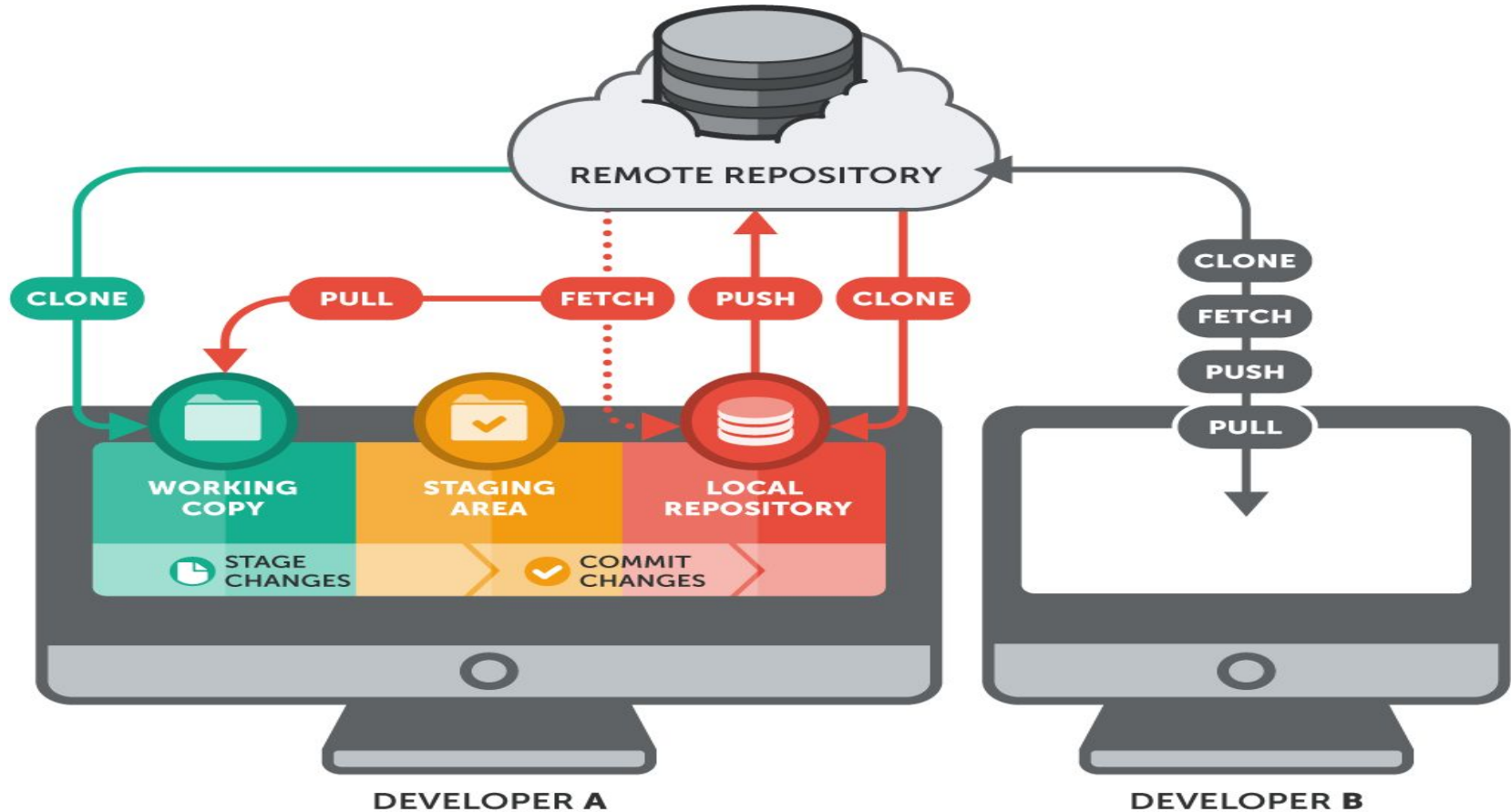
What is Pull Request?

- Pull requests require two branches: a **source branch** that contains the code you want reviewed, and a **destination branch**, where you merge the reviewed code.
- Pull requests let you tell others about changes you've pushed to a GitHub repository.
- Once a pull request is created, required users can review the set of changes, discuss potential modifications, and even push follow-up commits if necessary.

- After the changes have been reviewed and all approval rules on the pull request have been satisfied, you can merge a pull request using Github UI.

Pull Request Workflow

- **Pull** the changes to your local machine (get the most recent base of stable branch)

- 1. Create a new branch **feature_change** (version)

- 2. Commit the changes

- 3.a Push your changes

- 3.b Open a "Pull Request"(propose changes)(To merge feature_change into base branch)

- 4. Discuss and Review your code

- 5. Approval on Pull Request to merge.

- 6. "Merge" your branch to the master branch

Github Collaboration

- **Inviting collaborators to a personal repository**
  - Check for Github Username of the person you're inviting as a collaborator.
  - On GitHub, navigate to the main page of the repository.
  - Under your repository name, click **Settings** > click **Manage access** on left sidebar.
  - Click **Invite a collaborator.**
  - Enter username and Add NAME to REPOSITORY
  - The user will receive an email inviting them to your repository. Once they accept your invitation, they will have collaborator access to your repository.

Git Best Practices

- To avoid branch conflicts always raise a PR with the base branch that you have checked out from.
- There should be no changes made in feature branches once it has merged to the default branch i.e develop/master/main.
- A good commit message explains what the change is and why it matters.

Distributed Version Control Systems - Features

- The user can copy an existing repository. This copying process is typically called **cloning** and the resulting repository can be referred to as a clone.
- Providing the ability for collaborators to work offline and commit incrementally.
- Allowing a collaborator to determine when his/her work is ready to share.
- Offering the collaborator access to the repository history when offline.
- Allowing the managed work to be published to multiple repositories, potentially with different branches or granularity of changes visible.
- Every clone contains the full history of the collection of files and a cloned repository has the same functionality as the original repository

Why use SCM?

When working as a development team, SCM allows you:

- To collaborate efficiently on a single codebase

    Helps resolve code conflicts

    Makes it easy to share contents

- To track every change; SCM acts as a Single Source of Truth

    Provides a complete modification history

    Allows easy rollback to an earlier version

Git Terminology

- **git clone [url] :** Clone (download) a repository that already exists on GitHub, including all of the files, branches, and commits.
- **git status** : shows you what branch you're on, what files are in the working or staging directory and other important information.
- **git remote -v** : Show the associated remote repositories and their stored name, like **origin**.
- **git push** : Uploads all local branch commits to the remote.
- **git pull:** Update your local working branch with commits from the remote, and update all remote tracking branches.
- **git log** : Browse and inspect the evolution of project files.

Git Terminology

Term	Definition
Branch	A *branch* is a named pointer to a commit. Selecting a branch in Git terminology is called *to checkout* a branch.You can create a new branch from an existing one and change the code independently from other branches. One of the branches is the default (typically named *main*). The default branch is the one for which a local branch is automatically created when cloning the repository.
Commit	When you commit your changes into a repository this creates a new *commit object* in the Git repository. This commit object uniquely identifies a new revision of the content of the repository.
HEAD	*HEAD* is a symbolic reference most often pointing to the currently checked out branch.
Repository	A *repository* contains the history, the different versions over time and all different branches and tags. In Git each copy of the repository is a complete repository.

Git Terminology

Term	Definition
Staging area	The staging area is the place to store changes in the working tree before the commit. The staging area contains a snapshot of the changes in the working tree (changed or new files) relevant to create the next commit.
Working tree	The working tree contains the set of working files for the repository. You can modify the content of files, add files into staging area and commit the changes as new commits to the repository.
Index	Index is an alternative term for the staging area.
Tag	A *tag* points to a commit which uniquely identifies a version of the Git repository. With a tag, you can have a named point to which you can always revert to. You can revert to any point in a Git repository, but tags make it easier. The benefit of tags is to mark the repository for a specific reason, e.g., with a release. Branches and tags are named pointers, the difference is that branches move when a new commit is created while tags always point to the same commit. Tags can have a timestamp and a message associated with them.

In case of fire

1. git commit
2. git push
3. leave building