Welcome to GitHub

SCM Introduction

- Source code management (SCM) is used to track modifications to a source code repository.
- SCM tracks a running history of changes to a code base and helps resolve conflicts when merging updates from multiple contributors.
- SCM is also synonymous with Version control.
- As software projects grow in lines of code and contributor head count, the costs of communication overhead and management complexity also grow.
- SCM is a critical tool to alleviate the organizational strain of growing development costs.
- When multiple developers are working within a shared codebase it is a common occurrence to make edits to a shared piece of code.
- Separate developers may be working on a seemingly isolated feature, however this feature may use a shared code module. Therefore developer 1 working on Feature 1 could make some edits and find out later that Developer 2 working on Feature 2 has conflicting edits.

SCM Introduction

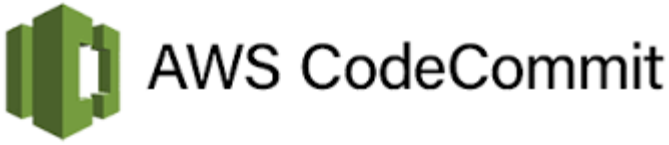
- Before the adoption of SCM this was a nightmare scenario. Developers would edit text files directly and move them around to remote locations using FTP or other protocols.
- Developer 1 would make edits and Developer 2 would unknowingly save over Developer 1’s work and wipe out the changes. SCM’s role as a protection mechanism against this specific scenario is known as Version Control.

Version Control System

- Version Control is the management of changes to documents, computer programs, large websites and other collection of information.
- Version control helps developers track and manage changes to a software project's code. As a software project grows, version control becomes essential.
- It records all the changes made to a file or set of files, so a specific version may be called later if needed.
- The system makes sure that all the team members are working on the latest version of the file.
- Helps in managing and protecting the source code.
- Comparing earlier versions of the code
- **Repository** – A directory or storage space where your projects can live. It can be local to a folder on your computer, or it can be a storage space on GitHub or another online host. You can keep code files, text files, image files, you name it, inside a repository.

Top Version Control System

Beanstalk

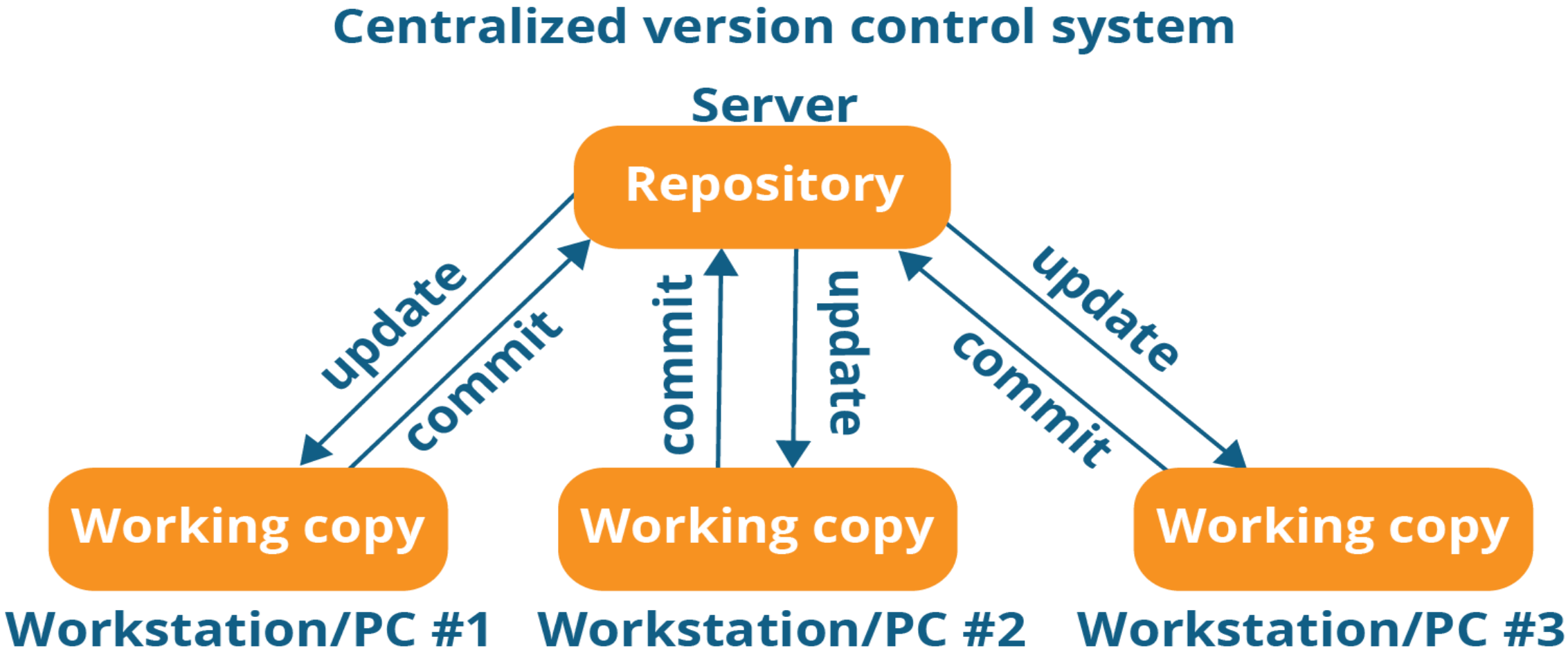


Version Control System

- There are two types of VCS:
  - Centralized Version Control System (CVCS)
  - Distributed Version Control System (DVCS)

Centralized Version Control System

- Uses a central server to store all files and enables team collaboration. It works on a single repository to which users can directly access a central server.

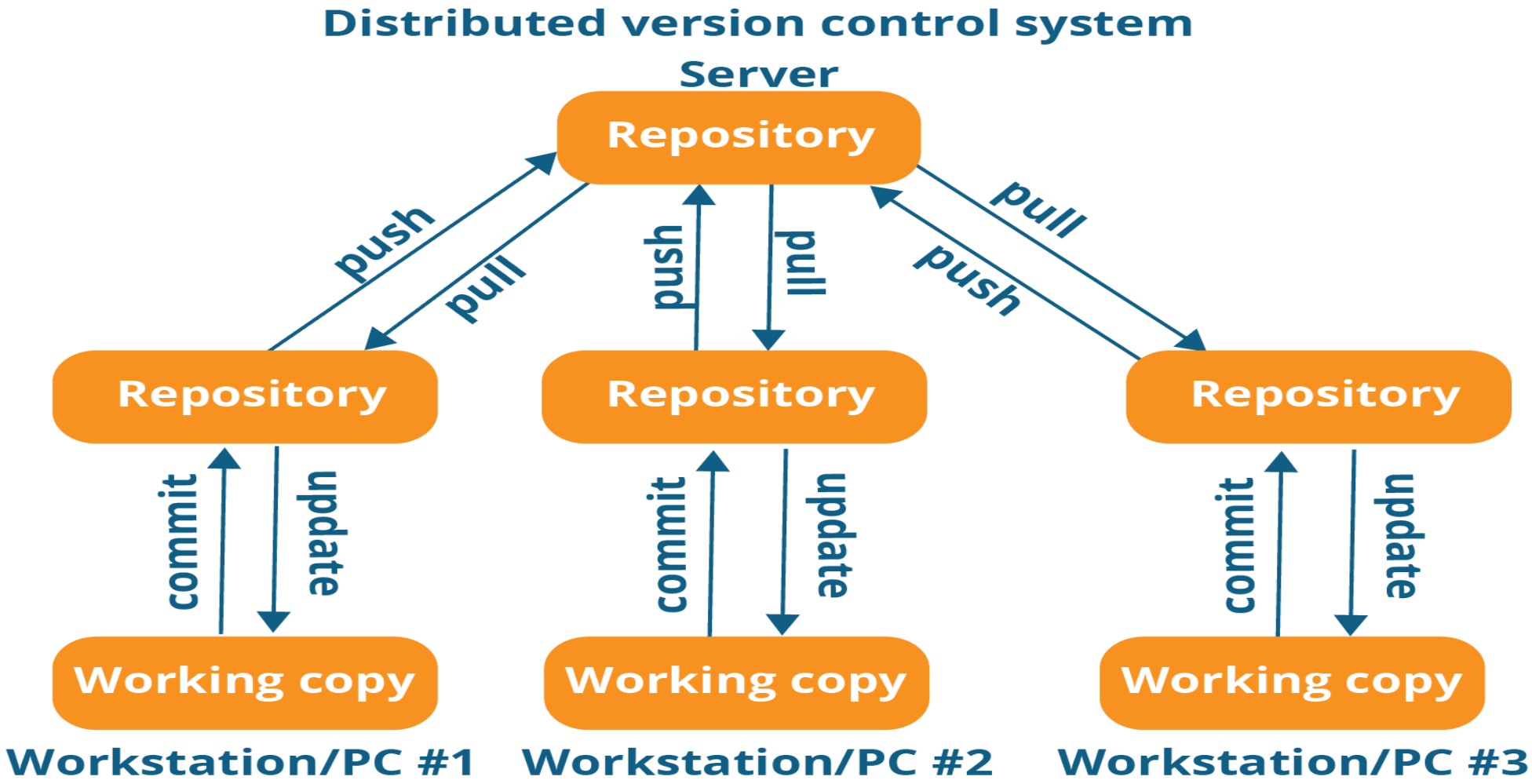


Centralized Version Control System

- CVS
- Subversion
- TFS
- Perforce

Decentralized Version Control System

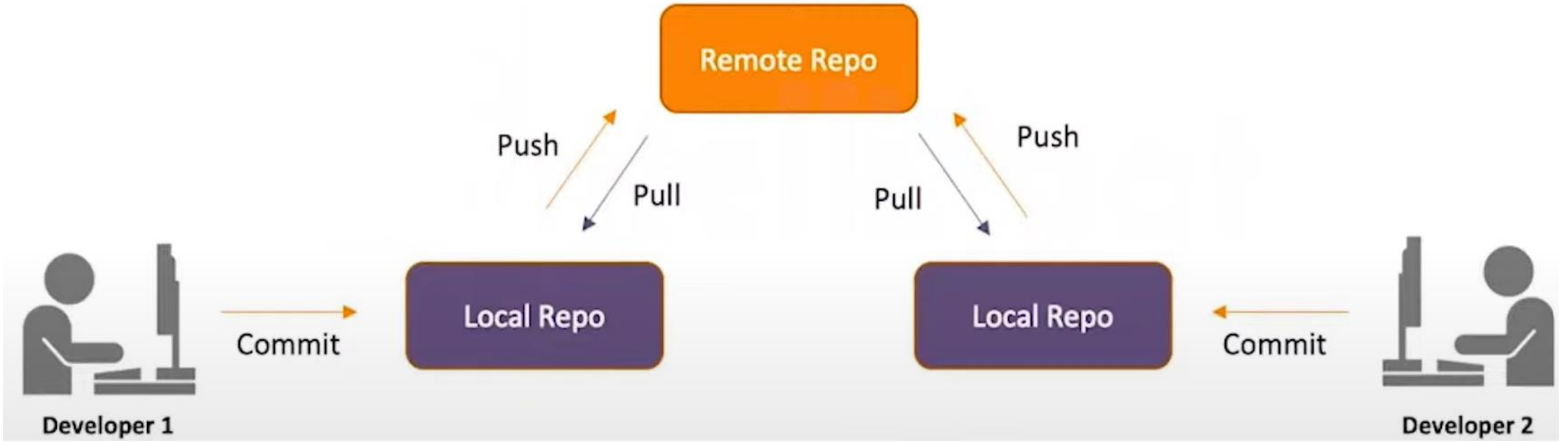
- Every contributor has a local copy or "clone" of the main repository i.e. everyone maintains a local repository of their own which contains all the files and metadata present in the main repository.



Decentralized Version Control System

- Git
- Mercurial
- AWS Code Commit
- BitKeeper
- Bazaar

Decentralized Version Control System



What is GitHub

- At a high level, GitHub is a website and cloud-based service that helps developers store and manage their code, as well as track and control changes to their code. To understand exactly what GitHub is, you need to know two connected principles:
  - Version control
  - Git

What is Git

- Git is a free, open source distributed version control system tool designed to handle everything from small to very large projects with speed and efficiency.
- It was created by Linus Torvalds in 2005 to develop Linux Kernel.
- Git has the functionality, performance, security and flexibility that most teams and individual developers need.
- It also serves as an important distributed version-control ***DevOps tool***.
- Git records the current state of the project by creating a tree graph from the index. It is usually in the form of a Directed Acyclic Graph (DAG).

Features of Git

Economical

Released under GPL's license. It is free & open source.

Non-Linear

Supports non-linear development of software.

Snapshots

Records changes made to a file rather than file itself

Distributed

Every user has his own copy of the repository data stored locally.

Speed

Speed offered by Git is lightening fast compared to other VCSs.

Robustness

Nearly every task in Git is undo-able.

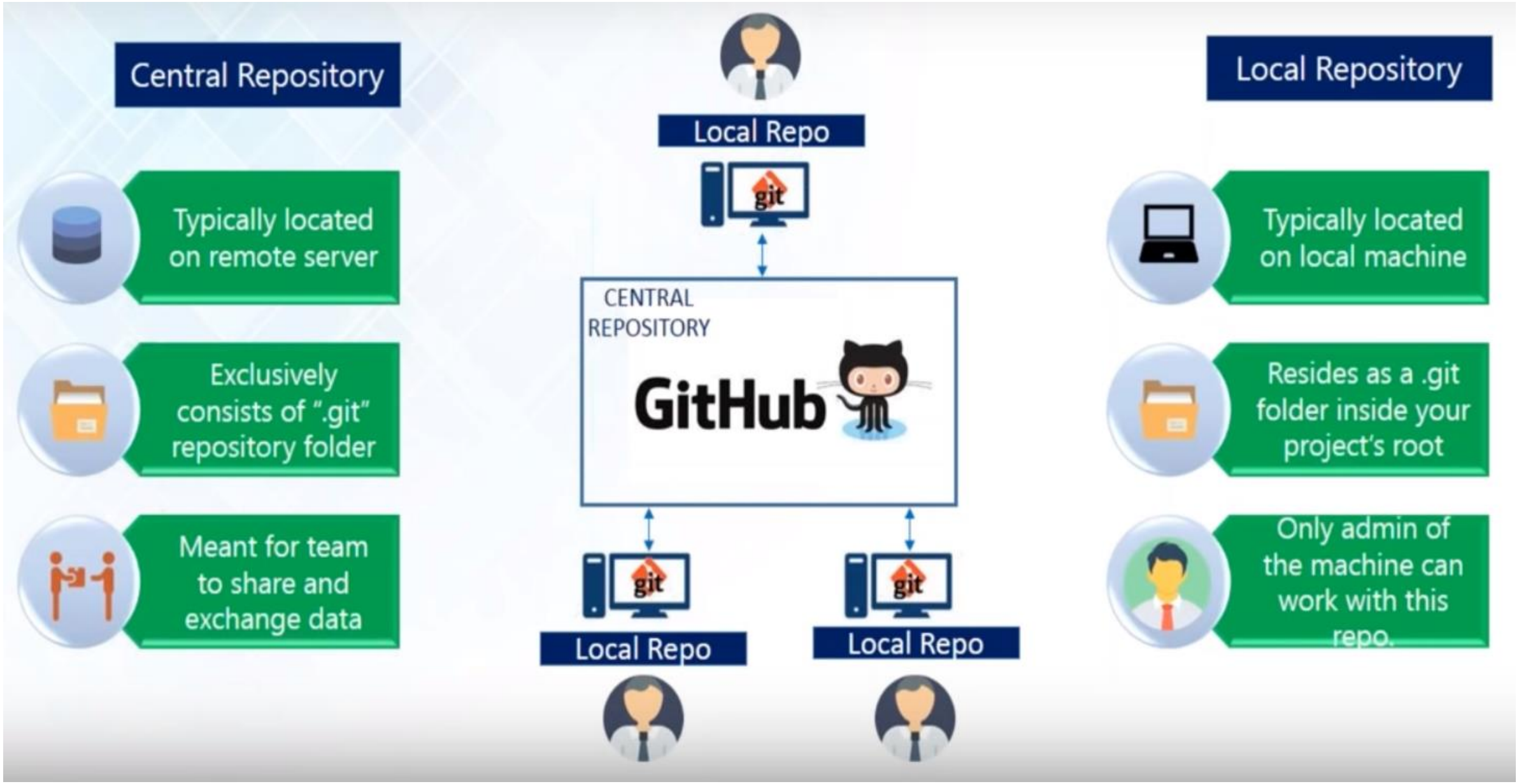
Integrity

No changes can be made without Git recording it.

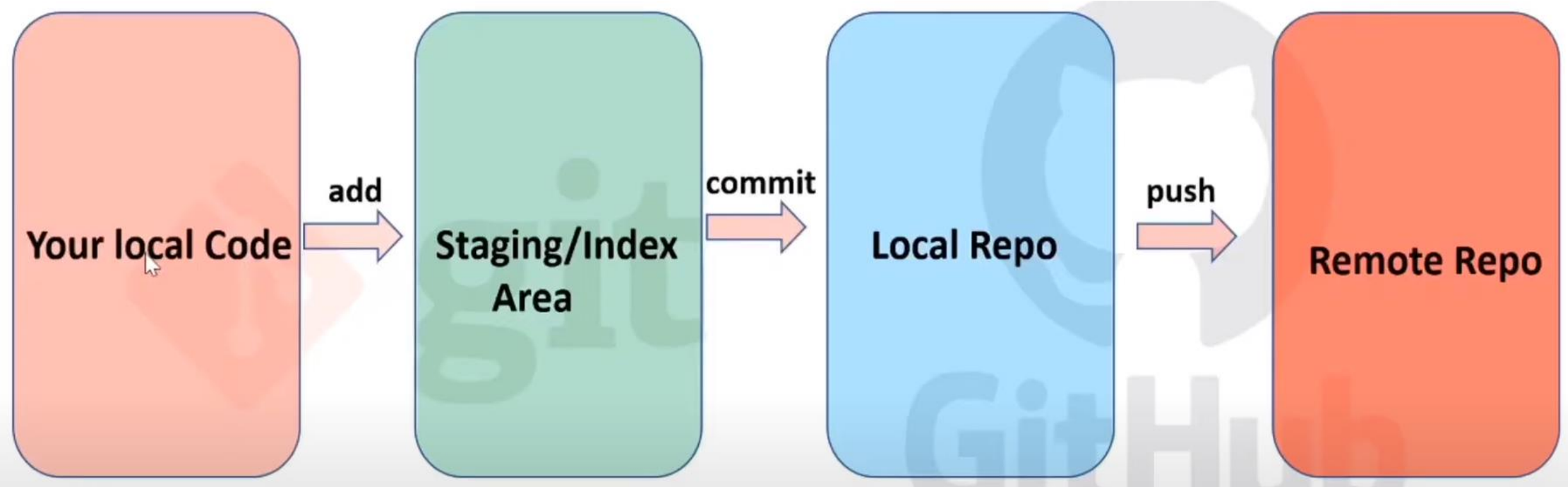
Branching

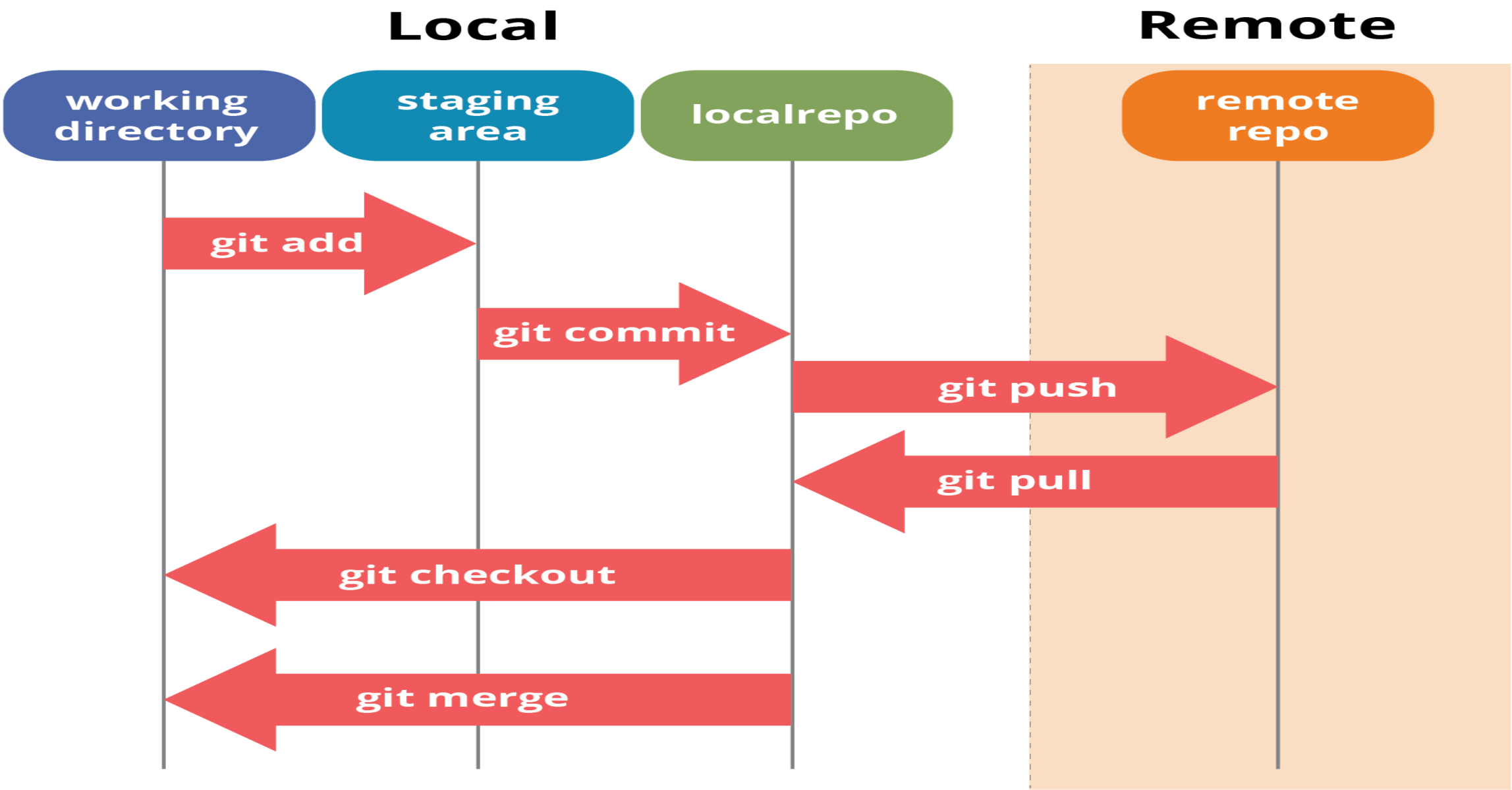
Every collaborator's working directory is a branch by itself.

Git Repository (Local vs Remote)



GitHub Ecosystem




Local
Remote
working directory
staging area
localrepo
remote repo
git add
git commit
git push
git pull
git checkout
git merge

Exploring GitHub Repository

- http://github.com

Git vs Github

Git is a revision control system, a tool to manage your source code history

Installed and maintained in your local system

Git is the **tool**

GitHub is a hosting service for Git repositories

Exclusively cloud-based

GitHub is the **service for projects that use Git**

- Git pushes or pulls data from the central server
- Github is a core hosting platform for version control collaboration.
- Github is a company that allows you to host a central repository in a remote server.

Git Environment Setup

- Git supports a command called **git config** that lets you get and set configuration variables that control all facets of how Git looks and operates. It is used to set Git configuration values on a global or local project level.
- Setting **user.name** and **user.email** are the necessary configuration options as your name and email will show up in your commit messages.
  - git config --global user.name "Himanshu Dubey"
  - git config --global user.email "himanshudubey481@gmail.com"
- You can set the default text editor when Git needs you to type in a message. If you have not selected any of the editors, Git will use your default system's editor.
- To select a different text editor, such as Vim,
  - git config --global core.editor Vim

Git Environment Setup

- You can check your configuration settings; you can use the git config --list command to list all the settings that Git can find at that point.
  - git config –list

Git configuration levels

- The git config command can accept arguments to specify the configuration level. The following configuration levels are available in the Git config.
  - local
  - global
  - system

Git Environment Setup – Configuration levels

--local

- It is the default level in Git. Git config will write to a local level if no configuration option is given. Local configuration values are stored in .git/config directory as a file.

--global

- The global level configuration is user-specific configuration. User-specific means, it is applied to an individual operating system user.
- Global configuration values are stored in a user's home directory.
  - ~ /.gitconfig on UNIX systems and
  - C:\Users\\.gitconfig on windows as a file format.

Git Environment Setup – Configuration levels

--system

- The system-level configuration is applied across an entire system. The entire system means all users on an operating system and all repositories.
- The system-level configuration file stores in a gitconfig file off the system directory. $(prefix)/etc/gitconfig on UNIX systems and C:\ProgramData\Git\config on Windows.

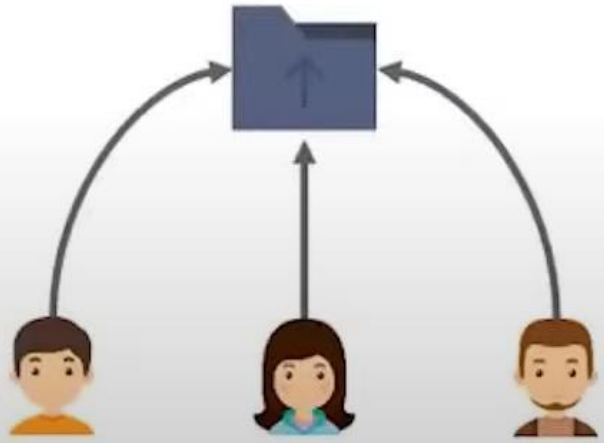
- The order of priority of the Git config is local, global, and system, respectively.
- It means when looking for a configuration value, Git will start at the local level and bubble up to the system level.

Git Workflow

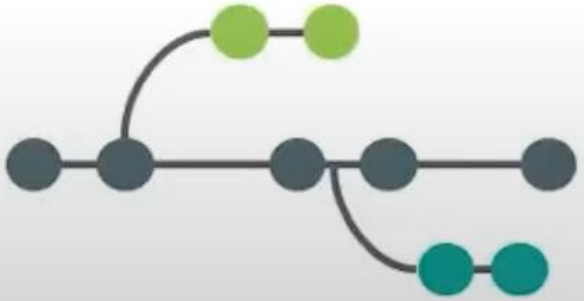
- A Git workflow is a recipe or recommendation for how to use Git to accomplish work in a consistent and productive manner.
- Git workflows encourage users to leverage Git effectively and consistently.
- There are the three popular workflows which are accepted and are followd by various tech companies:

Git Workflow

Centralized WorkFlow



Feature Branch WorkFlow



GitFlow WorkFlow

Git Workflow – Centralized WorkFlow

- This Workflow does not require any other branch other than Master.
- All the changes are directly made in the master, and finally merged on the remote master, once work is finished.
- Before Pushing changes, the master is rebased with the remote commits.
- Results in a clean, linear history.
- Ideally for a Prototype kind of applications
- If one or two developers are working on a project and no code review is needed

Git Terminology

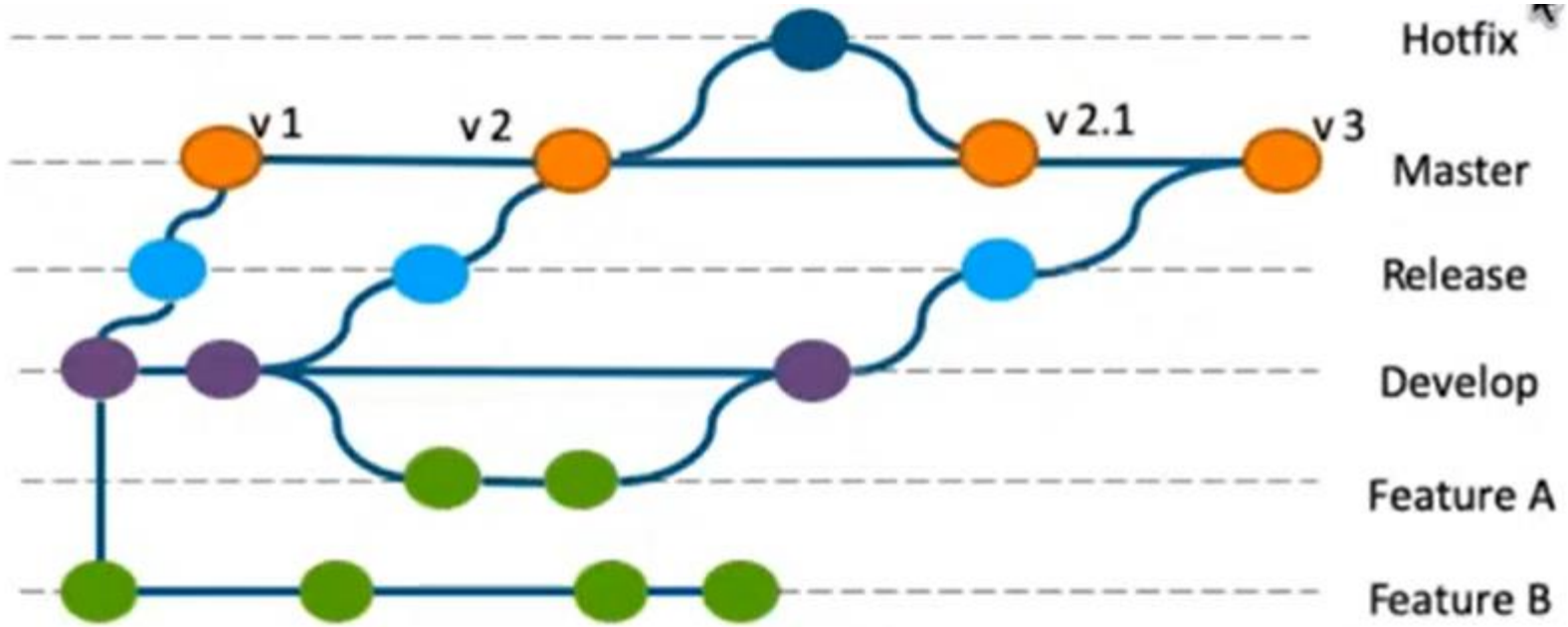
- **Repository** – A folder or directory where all your project code lives in. if this repository is located outside your system then we call this as Remote Repository. If this is located in your system then we call this as Local repository
- **Clone** – Creating Local Repository from Remote Repository
- **Promote/Push/Commit/Add** – Pushing Changes done to local repository into Remote Repository
- **Pull/Get/Update** – Pulling the Changes from remote Repository to Local repository

Git Workflow – Feature Branching

- Master only contains the production ready code.
- Any development work, is converted into a feature branch.
- There can be numerous feature branches, depending on the application's development plan.
- Once the feature is complete, the feature branch is merged with the master

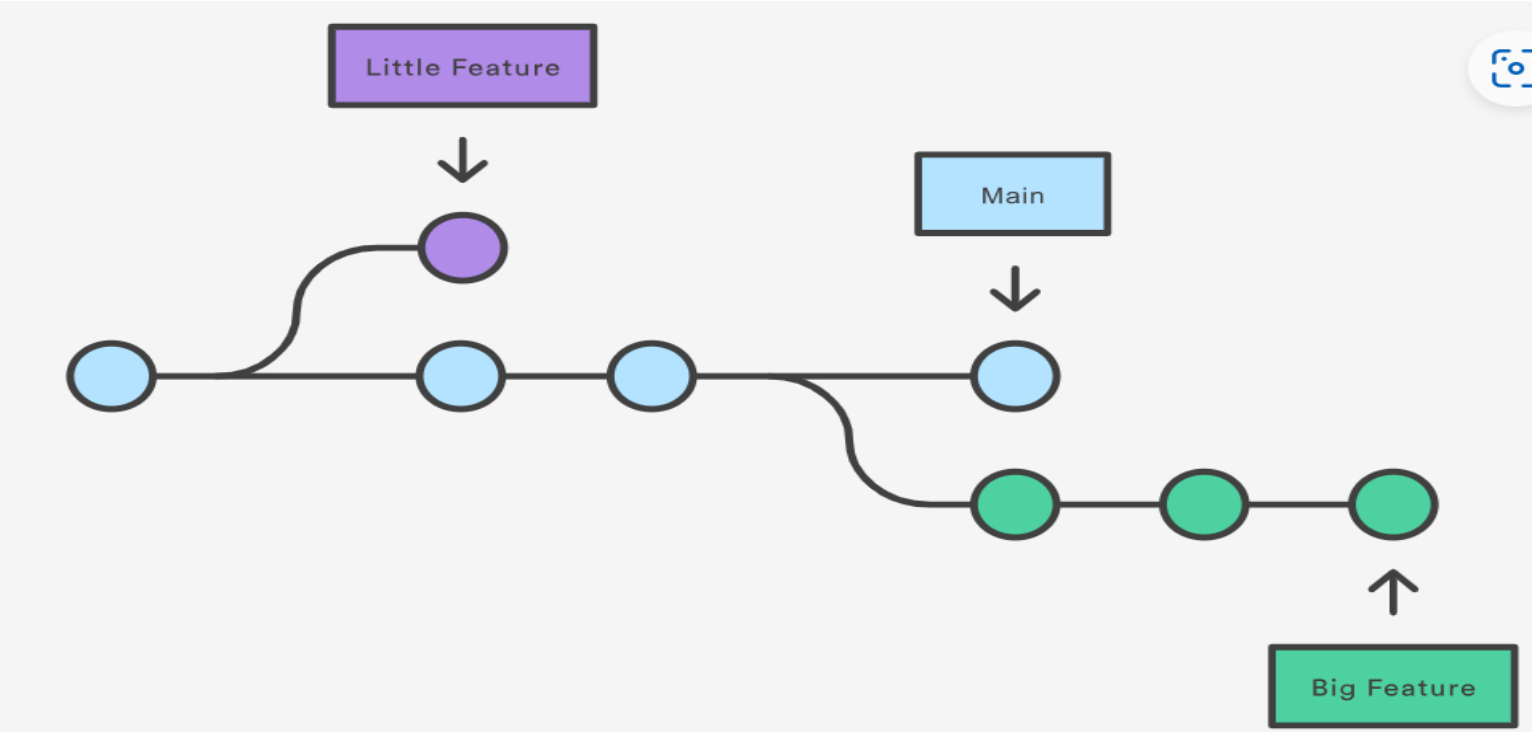
Git Workflow – Gitflow Workflow

- The Feature branches are never merged directly with master.
- Once the features are ready, commit is merged with Develop.
- When there are enough commits on Develop, we merge the Develop branch with Release branch, only read me files or License files are added after this commit on Release.
- Any quick fixes which are required, are done on the Hotfix branch, this branch can directly be merged with Master.



Git Branching

- Branches are pointers to a specific commit.
- Git branches are effectively a pointer to a snapshot of your changes.
- When you want to add a new feature or fix a bug—no matter how big or how small—you spawn a new branch to encapsulate your changes.
- This makes it harder for unstable code to get merged into the main code base, and it gives you the chance to clean up your future's history before merging it into the main branch.



Git Branching Commands

- **git branch** -- List all of the branches in your Repository
- **git branch branch1** -- Creates a new branch called branch1
- **git push --set-upstream origin <branch1>** -- To Map the newly created branch in local to Remote
- **git branch –d branch1** -- Deletes the branch1 if it is merged to its parent
- **git branch –D branch1** -- Force delete the specified branch, even if it has unmerged changes.
- The previous commands will delete a local copy of a branch. The branch may still exist in remote repos. To delete a remote branch execute the following.
- **git push origin --delete branch1**
- **git branch –m branch1** -- Rename the current branch to ＜branch＞.
- **git branch –a** -- List all remote branches.
- **git branch –merged**
- **git branch –no-merged**

Pull Request

- Notifying developers about changes you have pushed to a branch in a repository.
- Acknowledge and review changes.
- You can create a pull request on GitHub in 2 ways
  - From Forked repository
  - Within Repository

Git Ignore

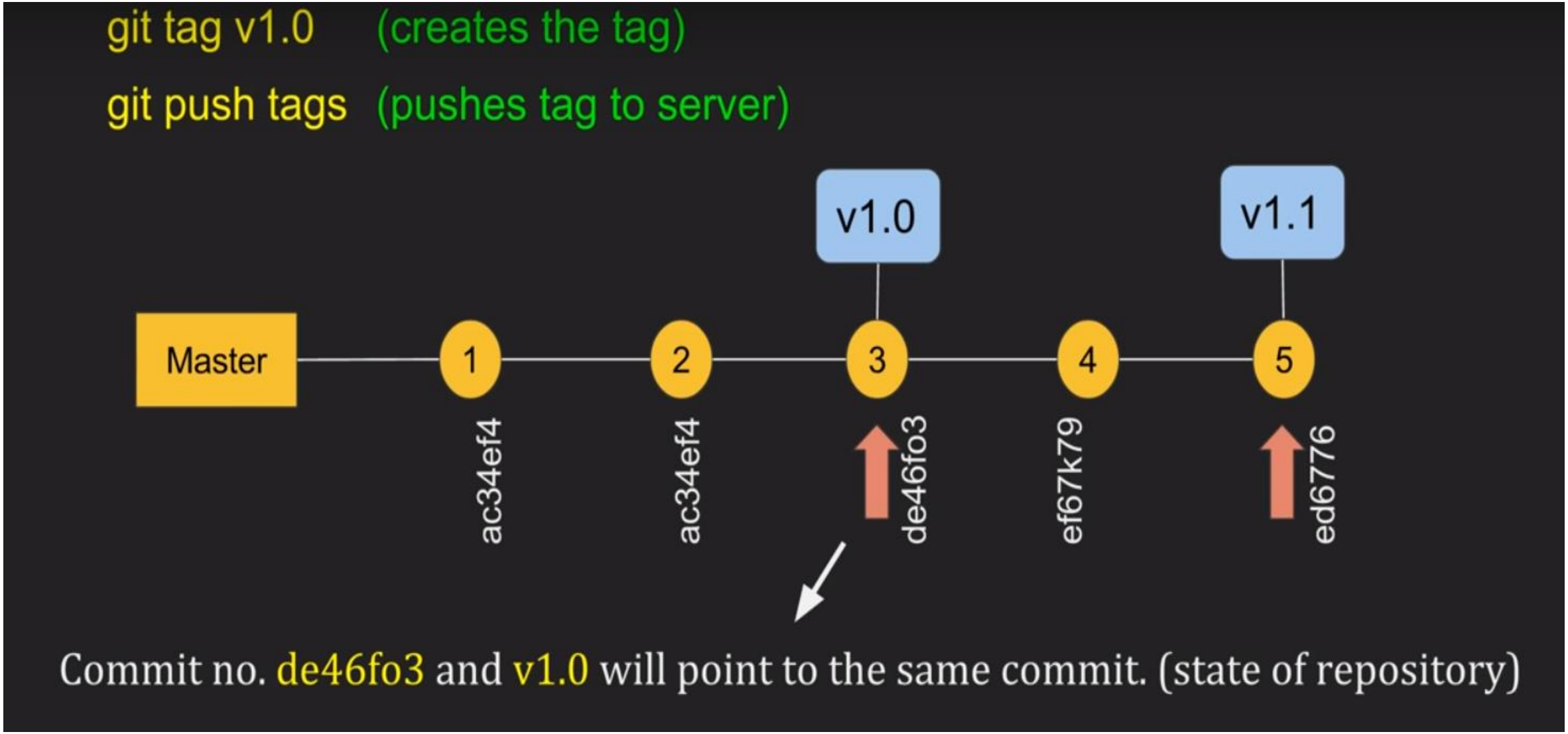
- A gitignore file specifies intentionally untracked files that Git should ignore. Files already tracked by Git are not affected.

- Each line in a gitignore file specifies a pattern. When deciding whether to ignore a path, Git normally checks gitignore patterns from multiple sources,

Git Tagging

- Git has the ability to tag specific points in a repository’s history as being important. Typically, people use this functionality to mark release points (v1.0, v2.0 and so on).
- A tag is like a branch that doesn’t change. Unlike branches, tags, after being created, have no further history of commits.

Git Tagging


git tag v1.0 (creates the tag)
git push tags (pushes tag to server)
v1.0
v1.1
Master
1
2
3
4
5
ac34ef4
ac34ef4
de46fo3
ef67k79
ed6776
Commit no. de46fo3 and v1.0 will point to the same commit. (state of repository)

Git Tagging

- Commands
  - Git tag <tagname> -- Creates the Tag from latest Commit ID
  - Git tag <tagname> <commitID> -- Creates the Tag from Commit ID
  - Git tag or git show <tagname> -- Displays the list of available Tags
  - Git push origin <tagname> or git push –tags -- Push the tags to Remote Repository
  - Git tag –d <tagname> OR git push origin –d <tagname> -- To delete the Tags
  - Git checkout –b <branchname> <tagname> -- Create the Branch from the Tag

Forking in GitHub

- A fork is a copy of a repository.
- Forking a repository allows you to freely experiment with changes without affecting the original project.
- Most commonly forks are used to either suggest changes to someone else's project or to use someone else's project as a starting point for your own idea.
- Forking is a GitHub concept, it has nothing to do with Git software.
- It is used to copy someone else's repository to your own repo in GitHub.
- The changes made to a forked repository are not reflected in the parent repository.
- If one wants to suggest any changeto the parent repository from the forked repository.

Forking Workflow

Git init

- The **git init** command creates a new Git repository.
- It can be used to convert an existing, unversioned project to a Git repository or initialize a new, empty repository.
- Most other Git commands are not available outside of an initialized repository, so this is usually the first command you'll run in a new project.
- Executing **git init** creates a .git subdirectory in the current working directory, which contains all of the necessary Git metadata for the new repository.
- This metadata includes subdirectories for objects, refs, and template files. A HEAD file is also created which points to the currently checked out commit.
- Commands
  - git init
  - git init <Directory-Name>

Q & A