Welcome to GitHub

Session-1
(07 June 2023)

What is GitHub

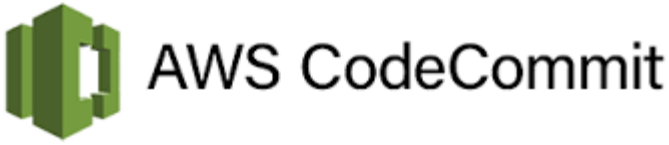
- At a high level, GitHub is a website and cloud-based service that helps developers store and manage their code, as well as track and control changes to their code. To understand exactly what GitHub is, you need to know two connected principles:
  - Version control
  - Git

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

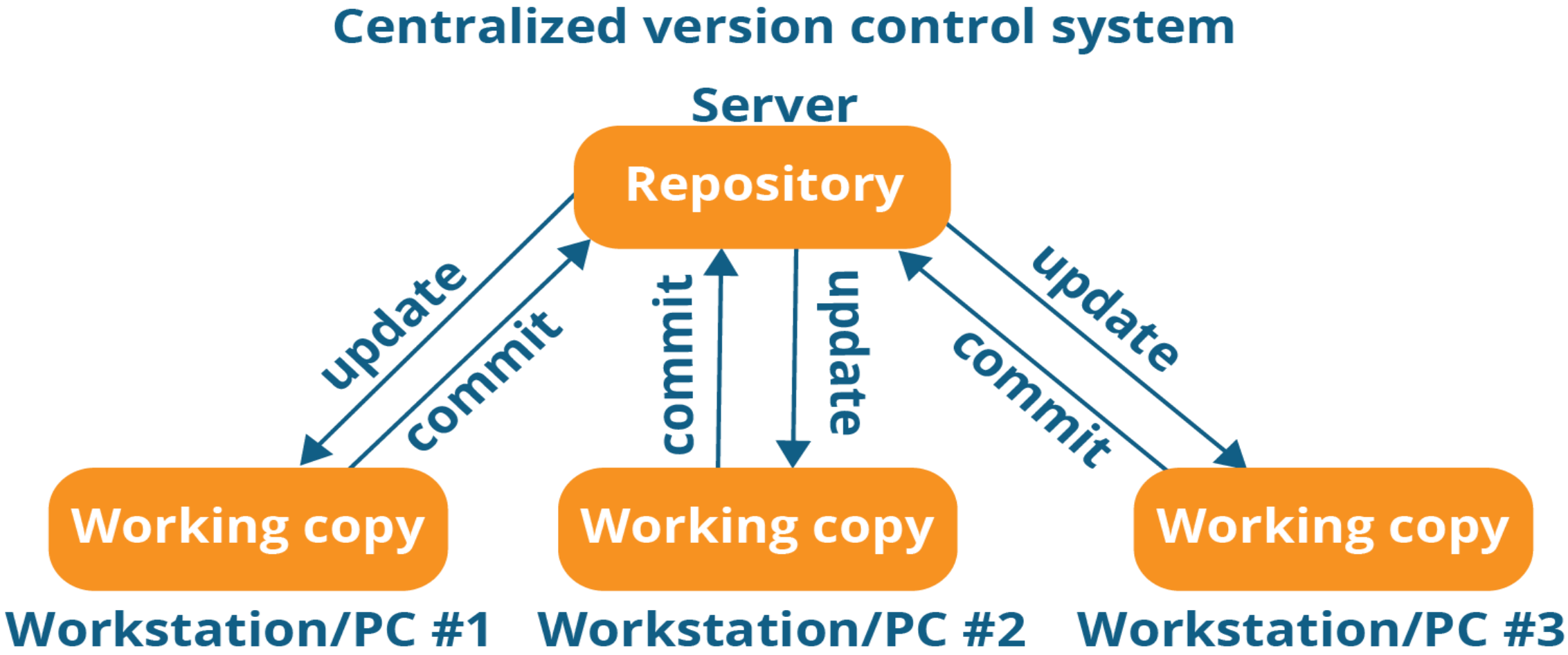


Version Control System

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

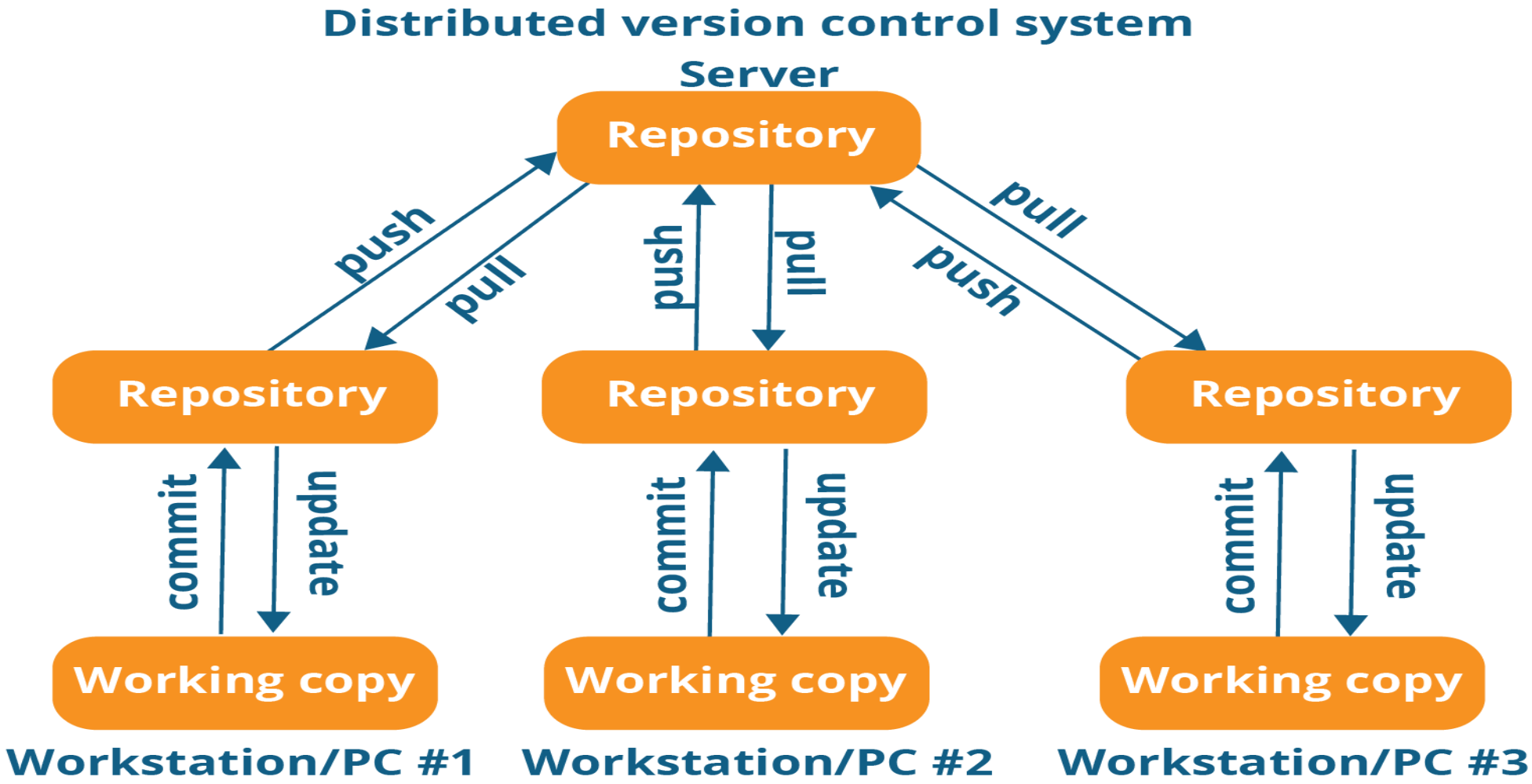
Centralized Version Control System

- 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.

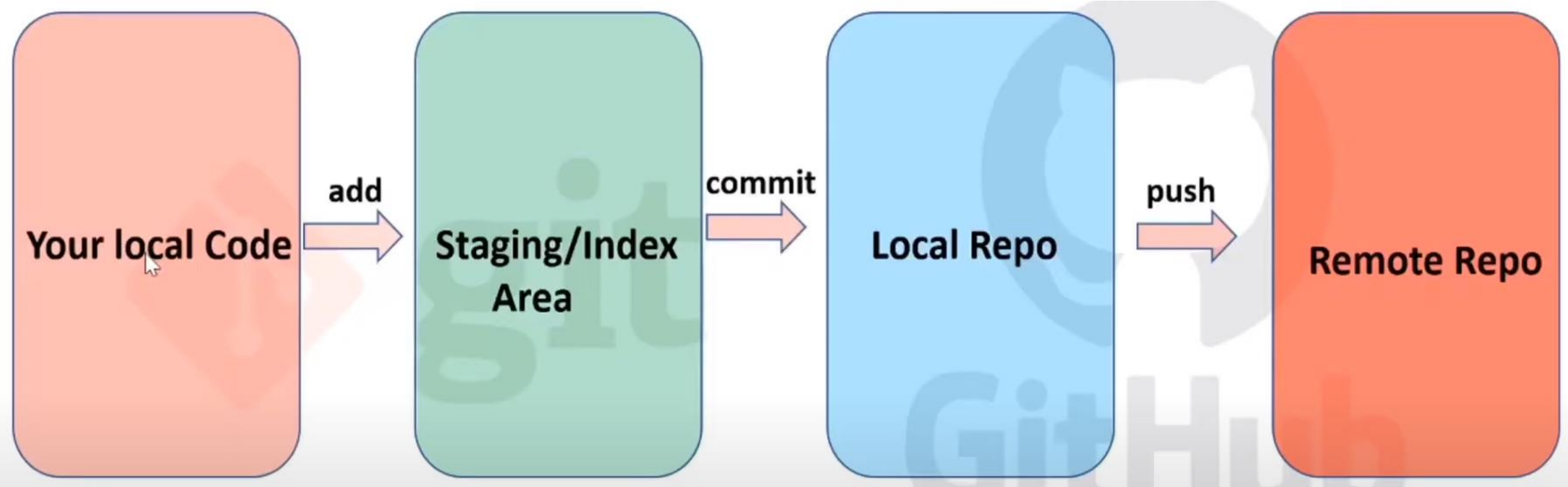


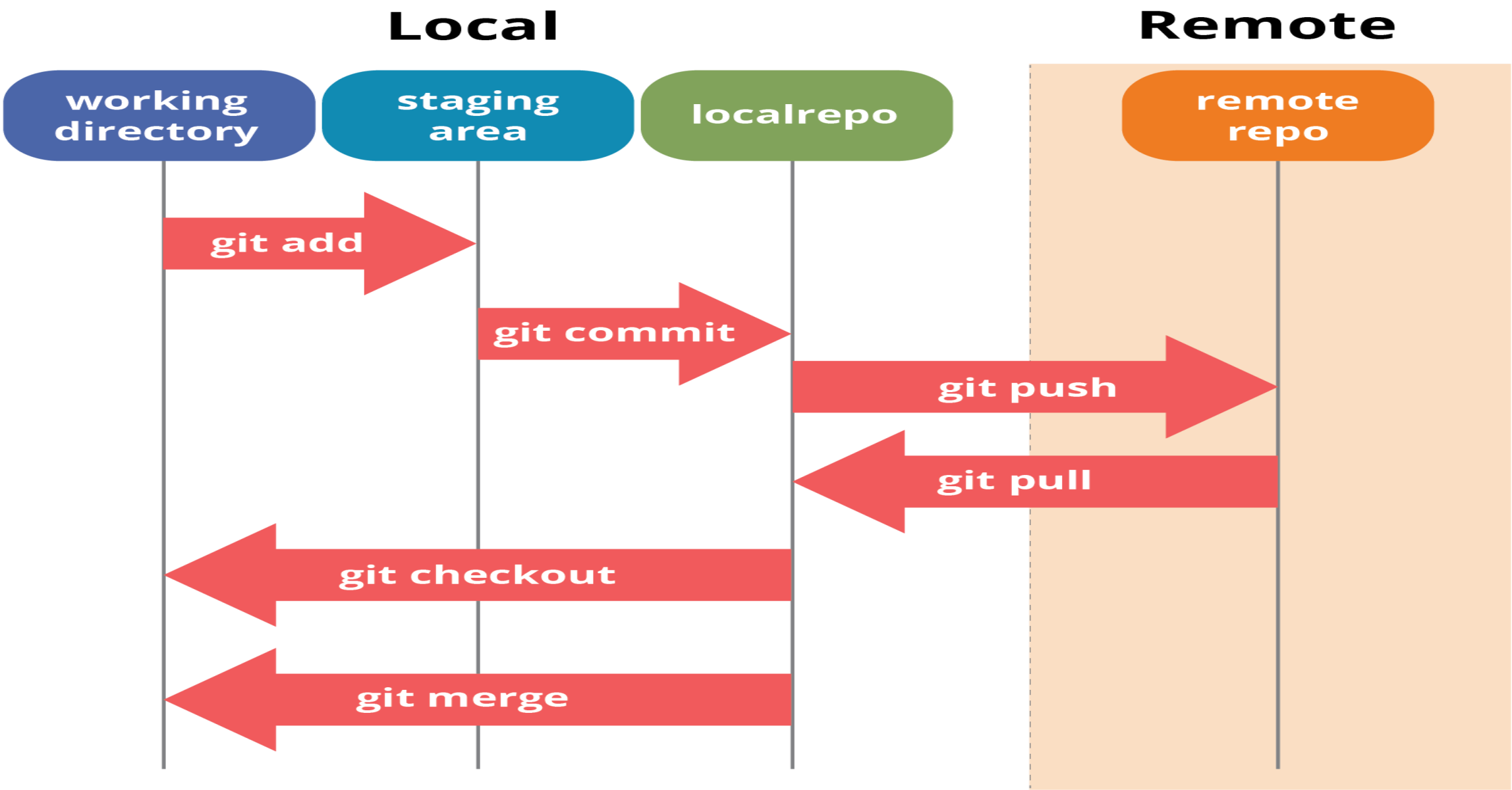
Decentralized Version Control System

- 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.



GitHub Ecosystem




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

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).

Exploring GitHub Repository

- http://github.com

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:
  - Centralized workflow
  - Feature Branching
  - GitFlow Workflow

Git Workflow – Centralized WorkFlow

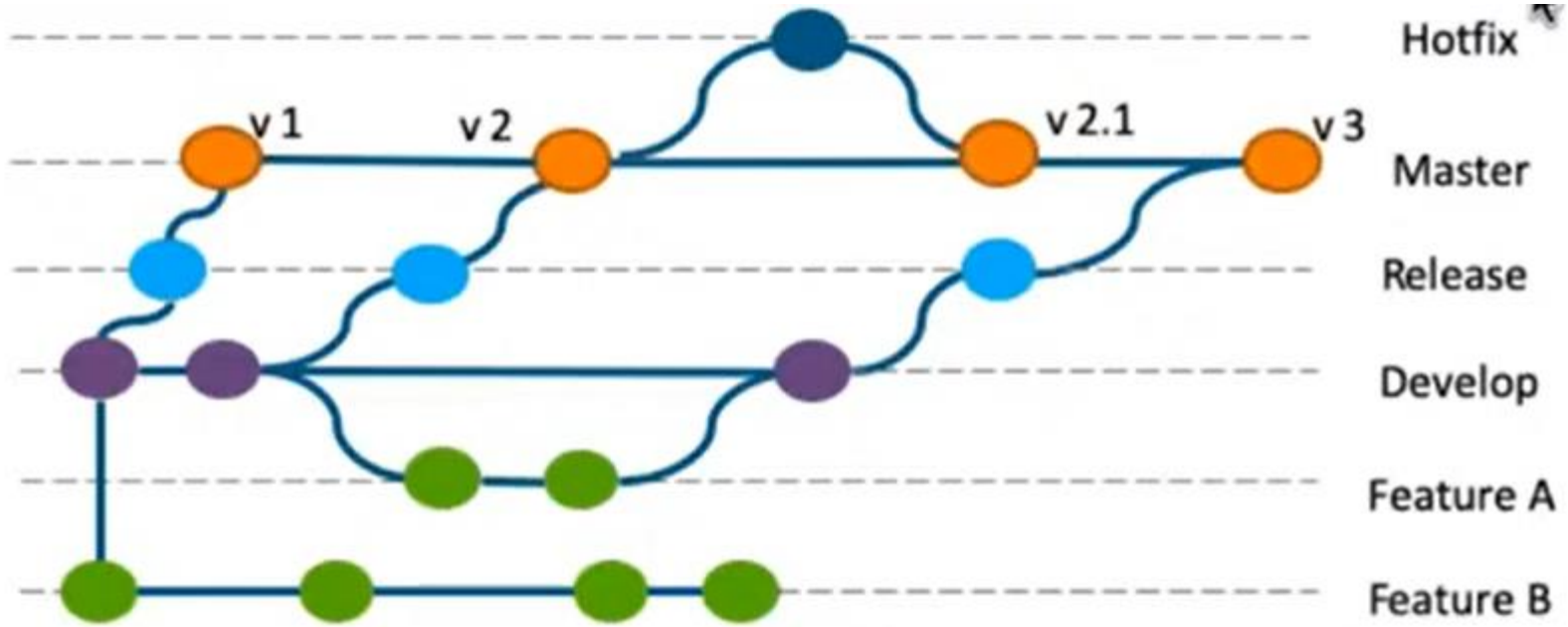
- This Workflow does not require any other branch other than Master.
- All the changes are directly made in the master, and finally merged on the remote master, once work is finished.
- Before Pushing changes, the master is rebased with the remote commits.
- Results in a clean, linear history.
- Ideally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

Git 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.



Session-2
(09 June 2023)

GitHub Issues

- Issues let you track your work on GitHub, where development happens.
- When you mention an issue in another issue or pull request, the issue's timeline reflects the cross-reference so that you can keep track of related work.
- To indicate that work is in progress, you can link an issue to a pull request.
- When the pull request merges, the linked issue automatically closes.
- You can organize and prioritize issues with projects. To track issues as part of a larger issue, you can use task lists. To categorize related issues, you can use labels and milestones.
- To stay updated on the most recent comments in an issue, you can subscribe to an issue to receive notifications about the latest comments. To quickly find links to recently updated issues you're subscribed to, visit your dashboard.

Git Autocrlf

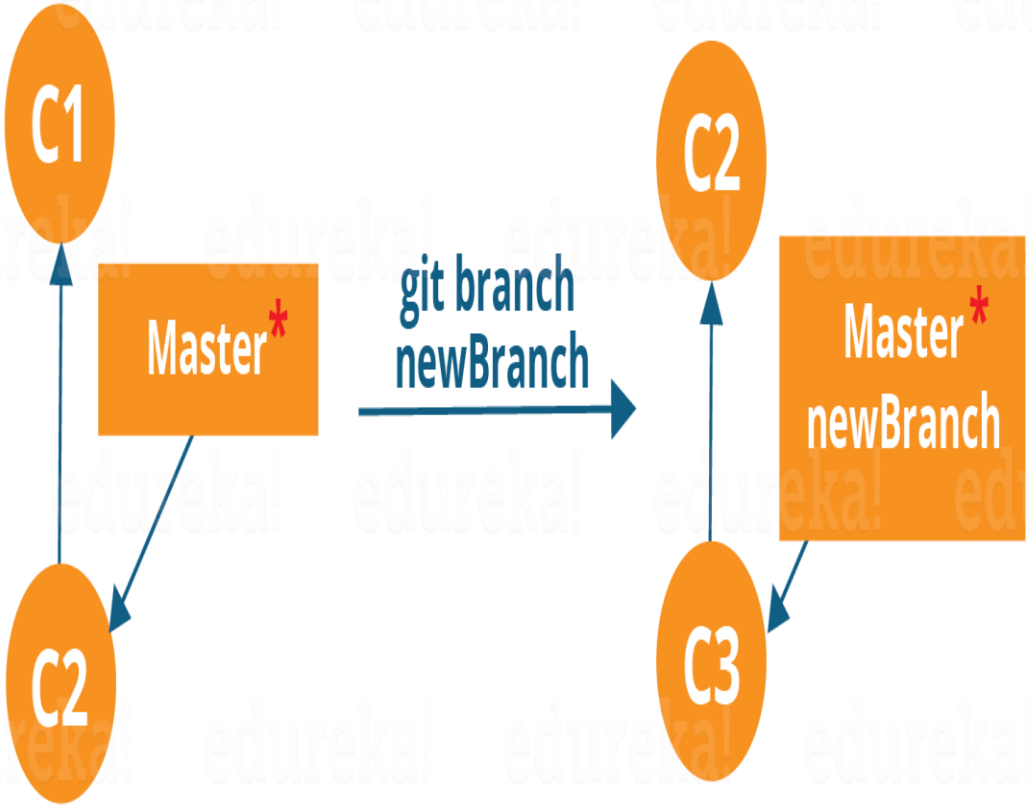
- Every time you press return on your keyboard you insert an invisible character called a line ending. Different operating systems handle line endings differently.
- When you're collaborating on projects with Git and GitHub, Git might produce unexpected results if, for example, you're working on a Windows machine, and your collaborator has made a change in macOS.
- You can configure Git to handle line endings automatically so you can collaborate effectively with people who use different operating systems.

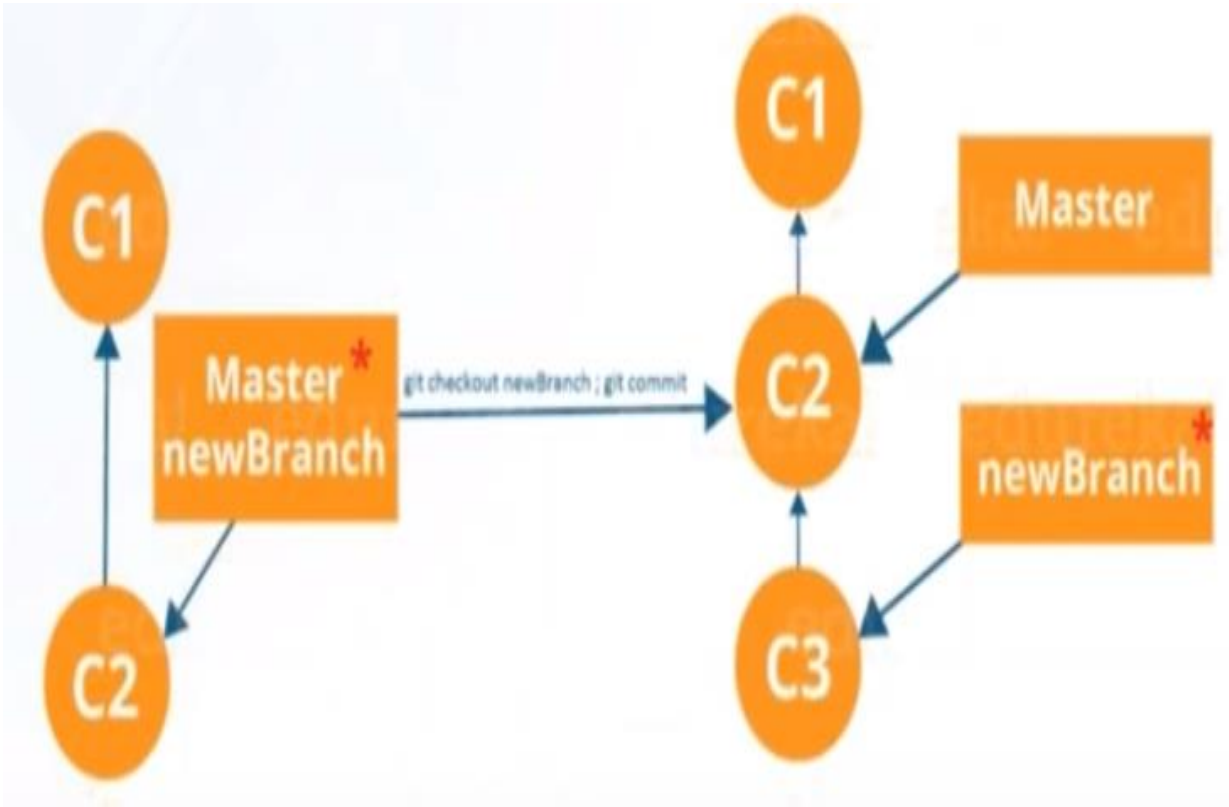
Git Autocrlf

- The git config core.autocrlf command is used to change how Git handles line endings. It takes a single argument.
  - $ git config --global core.autocrlf false

Branches and Switching Branches

- Branches are pointers to a specific commit.





Pull Request

- Creating a Pull Request Exploring a Pull Request and Code Review
- Merging Pull Requests

Editing Files on GitHub

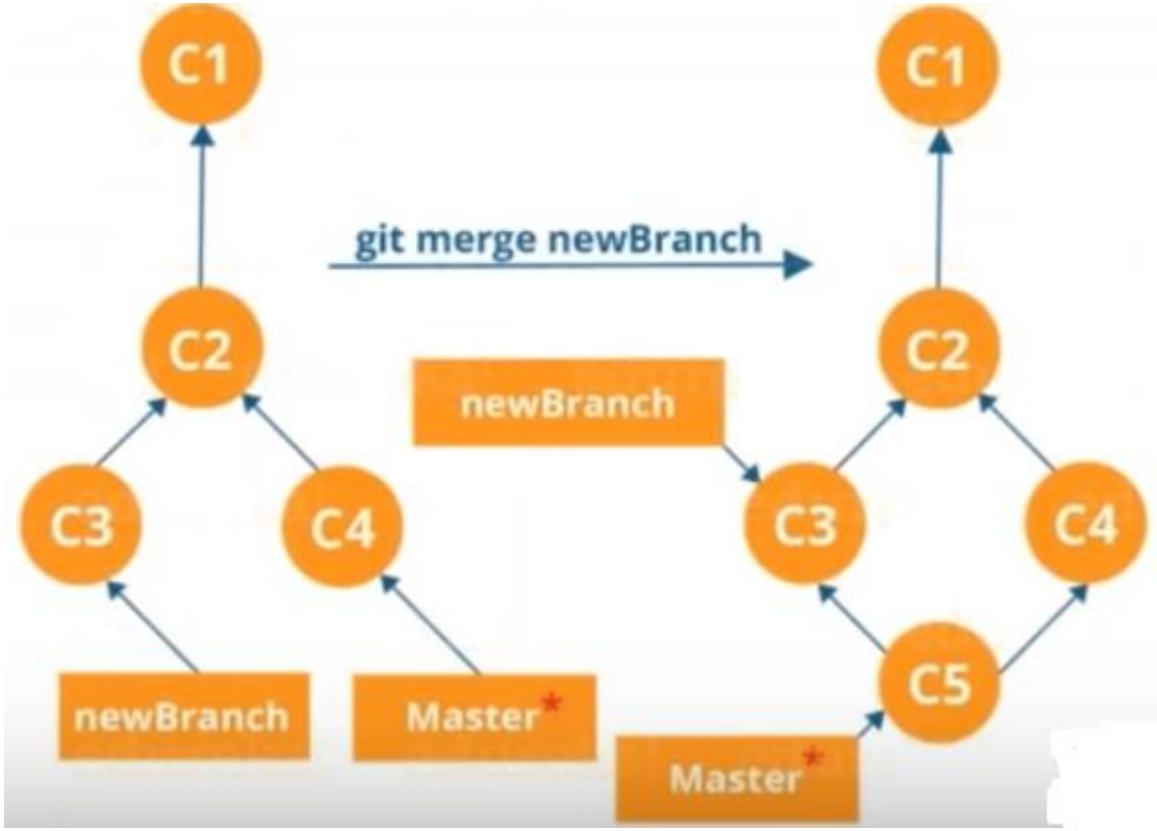
- You can edit files directly on GitHub in any of your repositories using the file editor.

Commiting Changes in Github

- Similar to saving a file that's been edited, a commit records changes to one or more files in your branch. Git assigns each commit a unique ID, called a SHA or hash, that identifies: The specific changes. When the changes were made.

Git Merging

- It is a way to combine the work of different branches together.
- Allows to branch off, develop a new feature & combine it back in.



Cleaning Up Branches

- When working with Git, it is quite usual to **accumulate** many different branches for the different features we are working on.
- However, when merged with our master branch, you may want **to clean up unused branches** in order for your Git workspace to be **more organized**.
- First of all, you want to check which branches have already been merged with your current branch.
- To check merged branches, use the “git branch” command with the “–merged” option.
  - git checkout master
  - git branch –merged
- Delete the branches that are already merged with
  - git branch -d <branch>

Cleaning Up Branches

- The other way of cleaning up local branches on Git is to use the “git branch” command with the “-D” option.
- In this case, the “-D” option stands for “–delete -force” and it is used when your local branches are not merged yet with your remote tracking branches.
  - git branch -D <branch>
- If there are any differences between the branches, you will have to use the “-D” option to delete the branch locally.
- Once local branch is deleted then to delete mapping remote branch
  - git push origin -d <branchname>

Cleaning Up Remote Tracking Branches

- A tracking-branch is a local branch set to track changes done on the remote branch of your Git server.
- Those tracking branches are created in order to track changes but they may become obsolete if remote branches were deleted on the server.
- In this case, let's say that you have a local "feature" branch, a remote-tracking branch named "origin/feature", but the "feature" branch has been deleted on the remote.
- In order to clean up remote tracking branches, meaning deleting references to non-existing remote branches, use the "git remote prune" command and specify the remote name.
  - git remote -v
    - origin https://gitserver.com/user/repository.git (fetch)
    - origin https://gitserver.com/user/repository.git (fetch)
- In order to delete remote tracking branches, we would then execute
  - git remote prune origin

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 change to the parent repository from the forked repository.

Forking Workflow

README

- A Readme file is used to communicate important information about a repository including
    - What the project does
    - How to run the project
    - Why it is noteworthy
    - Who maintains the Project
- If you put a Readme in the root of your project, Github will recognize it and automatically display it in the repo's home page.
- This is like entry point to learn more about the project or application.
- README's are markdown files, ending with the .mdextension. Markdown is a convenient syntax to generate formatted text.

README

- markdown-it demo

Session-3
(12 Jun 2023)

Git Bisect

- The git bisect command is used to discover the commit that has introduced a bug in the code. It helps track down the commit where the code works and the commit where it does not, hence, tracking down the commit that introduced the bug into the code.
- Suppose there are many commits, and we have to find the commit that introduced the bug.
- The normal way to do this would be to check out each commit one by one and build it until we reach the commit where the code does not compile.
- This is essentially a linear search and may take a long time depending on the number of commits.
- git bisect helps to find the faulty commit by performing a binary search on the commits to reduce the time taken to find the faulty commit.

Git Bisect

git bisect performs a binary search to find the faulty commit.

- At a high level, the way bisect works is that it lets you mark commits as “good” or “bad” until it can figure out the specific commit that caused the repository to flip from good to bad. To minimize the number of commits you have to inspect, it tries to stick to a binary search as much as possible.

Rename or Move files in GIT

- Git keeps track of changes to files in the working directory of a repository by their name.
- When you move or rename a file, Git doesn't see that a file was moved
- it sees that there's a file with a new filename, and the file with the old filename was deleted (even if the contents remain the same).
- renaming or moving a file in Git is essentially the same operation; both tell Git to look for an existing file in a new location.
- Moving and renaming files in version control systems rather than deleting and re-creating them is done to preserve their history.
  - **git mv <Oldfilename> <Newfilename>**
- If the filename you move to already exists, you'll need to use the following command
  - **git mv –f <Oldfilename> <Newfilename> OR**
  - **git mv –force <Oldfilename> <Newfilename>**

Staging Hunks of Changes

- To keep a track of modifications or changes in the file we have to bring that changes to our staging area which we bring by using **Staging**.
- While **Hunk** means a piece of changes.
- For example, we have written 4 lines of text in a file and modified 4 lines of text in that file that is known as a hunk which you can consider as a piece of change.
- command we use to add all the modified data while working on git is
  - **git add .**
- We can add changes by Hunk by using **git add -p <FileName>** command which will open an interactive prompt that will ask the user whether we want to Stage all the Hunks or any particular Hunk.

Staging Hunks of Changes

- **git add -p <FileName>** is the command which provides us an interactive interface along with various commands in that interface.
  - Select **s** to split the Changes
  - Press **n** to go to next hunk
  - When you decided to stage that Hunk, Press **y**

Git Diff

- Git diff is a command-line utility. It's a multiuse Git command. When it is executed, it runs a diff function on Git data sources. These data sources can be files, branches, commits, and more. It is used to show changes between commits, commit, and working tree, etc.
- However, we can also track the changes with the help of git log command with option -p. The git log command will also work as a git diff command.

Scenerio1: Track the changes that have not been staged.

- The usual use of git diff command that we can track the changes that have not been staged.
- Suppose we have edited the file1.txt file. Now, we want to track what changes are not staged yet. Then we can do so from the **git diff** command

Git Diff

Scenerio2: Track the changes that have staged but not committed:

- The git diff command allows us to track the changes that are staged but not committed. We can track the changes in the staging area.
- To check the already staged changes, use the --staged option along with git diff command.
  git diff --staged

Scenerio3: Track the changes after committing a file:

- Git, let us track the changes after committing a file. Suppose we have committed a file for the repository and made some additional changes after the commit. So we can track the file on this stage also.
- Now, we have changed the file1.txt file again as "Changes are made after committing the file." To track the changes of this file, run the git diff command with HEAD argument. : **git diff HEAD**

Git Diff

Scenario4: Track the changes between two commits:

- We can track the changes between two different commits.
- Git allows us to track changes between two commits, whether it is the latest commit or the old commit. But the required thing for this is that we must have a list of commits so that we can compare.
- The usual command to list the commits in the **git log** command.
- Suppose, we want to track changes of a specified from an earlier commit. To do so, we must need the commits of that specified file. To display the commits of any specified, run the git log command as:

git log -p follow file1.txt

git diff <commit1> <commit2>

- The above command will display the changes between two commits.

Git Diff Branches

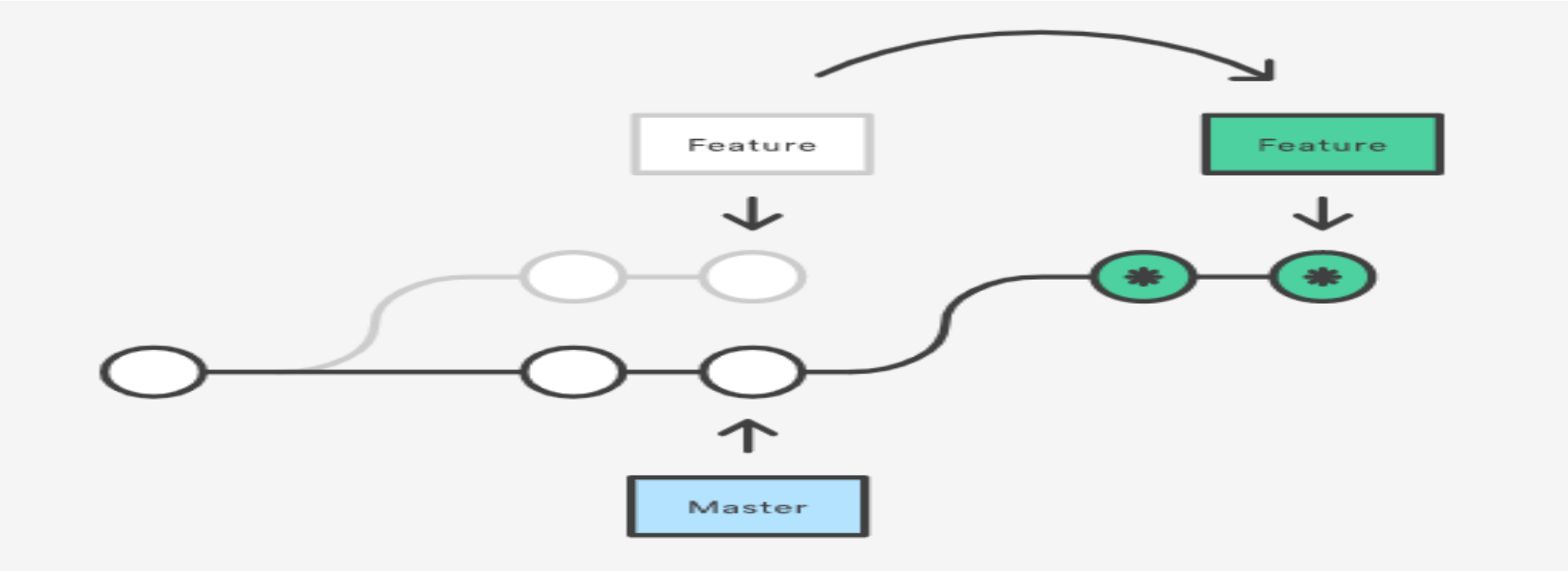
- Git allows comparing the branches. If you are a master in branching, then you can understand the importance of analyzing the branches before merging. Many conflicts can arise if you merge the branch without comparing it. So to avoid these conflicts, Git allows many handy commands to preview, compare, and edit the changes.
- The git diff command is a widely used tool to track the changes.
- The git diff command allows us to compare different versions of branches and repository. To get the difference between branches, run the git diff command as follows:

git diff <branch 1> < branch 2>

* The git diff command is giving a preview of both branches. So, it will be helpful to perform any operation on branches.

Git Rebasing

- Rebasing is the process of moving or combining a sequence of commits to a new base commit.
- Changing the base of your branch from one commit to another making it appear as if you'd created your branch from a different commit.
- The primary reason for rebasing is to maintain a linear project history.



Git Rebasing

- Generally, it is an alternative of git merge command. Merge is always a forward changing record.
- Comparatively, rebase is a compelling history rewriting tool in git. It merges the different commits one by one.
- Branch will take the updated version of its parent branch
- You will lose the history

Git Rebasing vs Merge

- Both of these commands are designed to integrate changes from one branch into another branch—they just do it in very different ways.
- The major benefit of rebasing is that you get a much cleaner project history. First, it eliminates the unnecessary merge commits required by git merge. Second, rebasing also results in a perfectly linear project history
- Git checkout feature
- Git merge master
- Git rebase master

Git Rebasing vs Merge LAB

- Create Repository
- Clone it to Local Repository
- Add Couple of Commits
- Create Two Branches from main Branch
- Add two more commits to main branch
- Add two more commits in each of the branches created from main
- From one Branch execute
  - Git merge master
- From another Branch
  - Git rebase master
- Go to two branches and check git log --oneline

Q & A