**GIT**

**Version control With GIT**

**Agenda**

* 1. What is Version Control?
  2. Types of Version Control System
  3. Introduction to GIT
  4. GIT Lifecycle
  5. Common GIT commands
  6. Merging in GIT
  7. Resolving Merge Conflicts
  8. Git Workflow

What is Version Control?

* Version control is a system that records/manages changes to documents,

computer programs etc over time. It helps

us tracking changes when multiple people work on the same project.

Advantages of Version Control:

* Versioning is Automatic
* Team collaboration is simple
* Easy access to previous versions
* Only modified code is stored across different versions, hence saves storage

Types of Version Control System:

1. Central VCS
2. Distributed VCS

Centralized VCS:

* Centralized Version Control system has one single copy of code in the central server.
* If the central server crashes, all data from the project will be lost.
* Developers will have to “commit” their changes in code to this central server.
* “Committing” a change is to recording the change in the central system.

Distributed VCS:

* In this VCS, one does not necessarily rely on a central server to store all the versions of a projects file.
* There is not threat, even if the server crashes.
* Every developer “clones” a copy of the main repository on their local system.
* This also copies, all the past versions of the code on the local system too.
* Therefore, the developer need not be connected to the internet to work on the code.

Introduction to GIT:

Why Git?

* It ensures there are no code conflicts between developers.
* Developers can revert their project to the older versions of the code.

What is Git?

* Git is a version control system that is used to manage & keep track of code.
* Everytime we commit or save, git basically takes a pictures of what files look like at that moment & stores a reference to that snapshot.

Git Lifecycle:

The lifecycle stages are as follows;



1. Working Directory:

* The place where your project resides in your local disk.
* The project can be tracked by git, using the command “git init” & it automatically creates a hidden .git folder.

1. Staging Area:

* Staged files are the files that are ready to be committed to the repository we are working on.
* To get Git to track them, we need to stage them, or add them to the staging environment. Using command “git add --all”.

1. Commit:

* Once the files are selected & are ready in staging area, they can now be saved in repository.
* Saving file is known as doing a commit.
* The command is “git commit –m ‘message’”.

Common Git Commands:

* **git init:** To create new repository & to initialize a git repository on local system.
* **git clone [url]:** To download the remote repository to our local system.
* **git status:** To check status of the files.
* **git add:** Add changes to the staging area.
* **git commit:** Commit changes to the local repository.(git commit –m “message”)
* **git remote:** Connecting local repository to the remote repository.(git remote add origin “<url>”) .Here remote repository is on github.
* **git push:** To push the changes to your repository(git push origin <branch-name>). If the branch is master, hence use command (git push origin master).
* **git pull:** To pull changes from remote repository(github).
* **git branch:** Branches allow you to work on different parts of projet without impacting the main branch. Also we can work on different projects without interfering with each other.

(git branch <new branch name>)

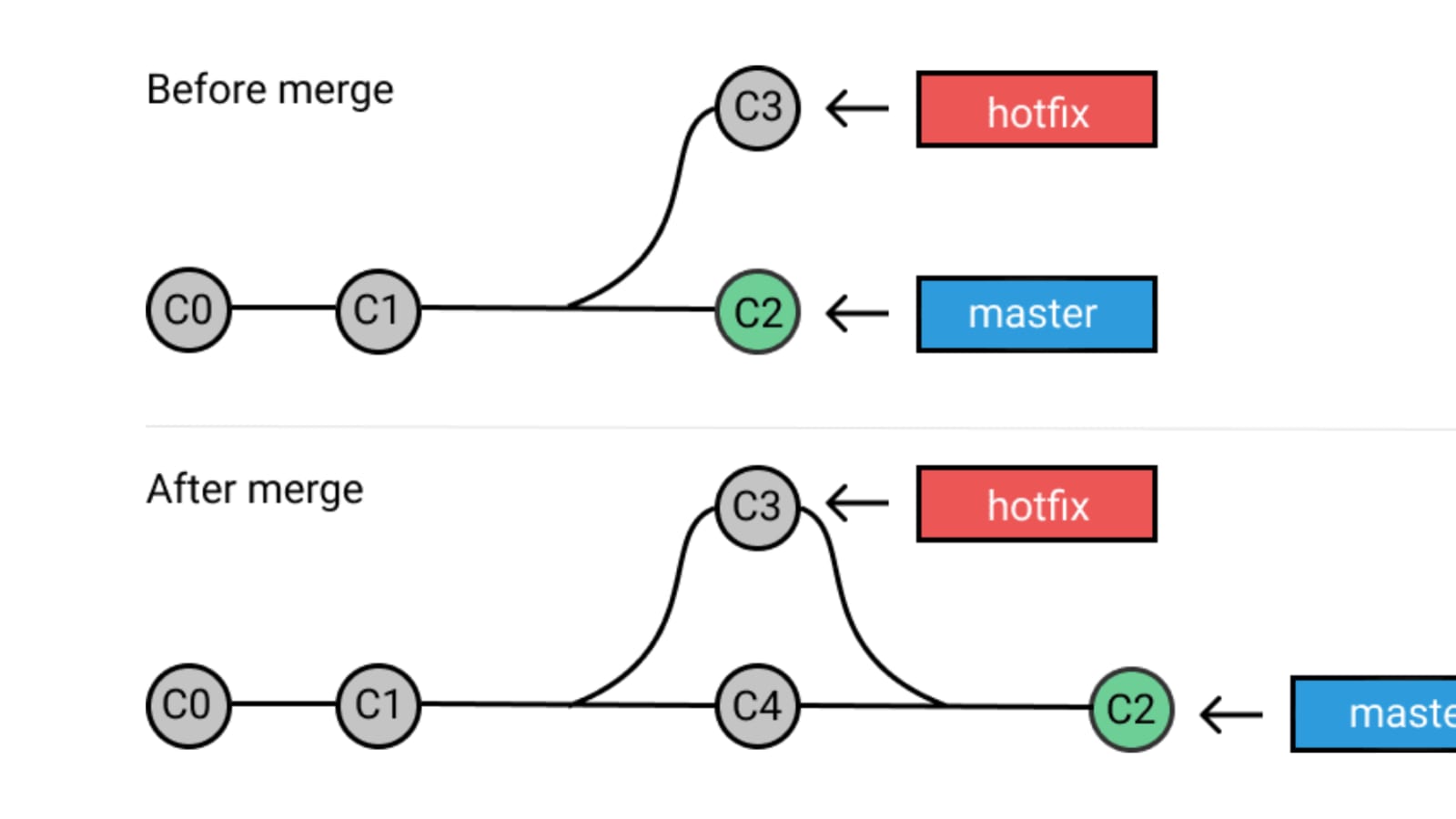
* **git checkout <branch name>:** It is used to switch to the new branch.
* **git log:** It is used to view the history of commits for a repo.
* **git Stash:** Used when we want to save our work without committing the code. It takes our modified tracked files & saves them on stash of unfinished changes, that we can reapply at any time.
* **git revert<commit-id>:** It reverts the commit that has already been pushed & made public.
* **git diff:** It helps in checking the difference between two versions of a file.

(git diff <commit-id of version x> <commit-id of version y>)

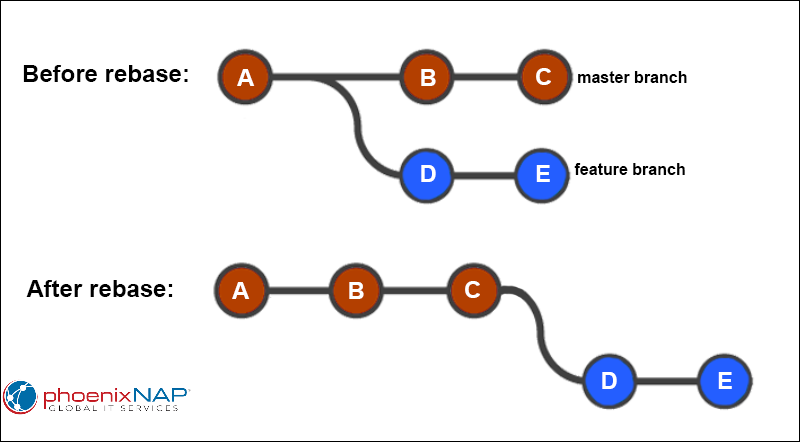
Merging in Git:

* If the developer has finished his code on his branch, then code will have to be combined with master branch. This can be done using two ways.

1. Git Merge
2. Git Rebase

Git Merge: If we want to apply changes from one branch to another branch, then we use merge command. “git merge <source branch name>”

* It’s preferred to switch to the master branch before any branch wants to merge with it.
* Should be used on remote branches, since history does not change.

Git Rebase: It’s an alternative to git merge command. And should be used on local branches, since history does change.

“git rebase <source branch>”

Merge Conflicts:

* Merge conflicts occur when changes are made to the same line of a file. For example if one person edits a file & another person deletes the same file.

Resolving Merge Conflicts:

To resolve merge conflict use the following commands.

* git status: To identify conflicted files.
* git log --merge: It produces the list of commits that are in conflict with source branch.
* git pull --rebase origin master: To know where conflict is.
* git diff: It is used for comparing branches, commits, and files.
* git mergetool: To solve conflict.
* git rebase --continue : To continue next.
* git merge --abort: To abort the conflict.