Git Branches

Objectives

- Branches
- Merges
- Conflicts

1 Recap- Git Workflow

Recap-What is Git?

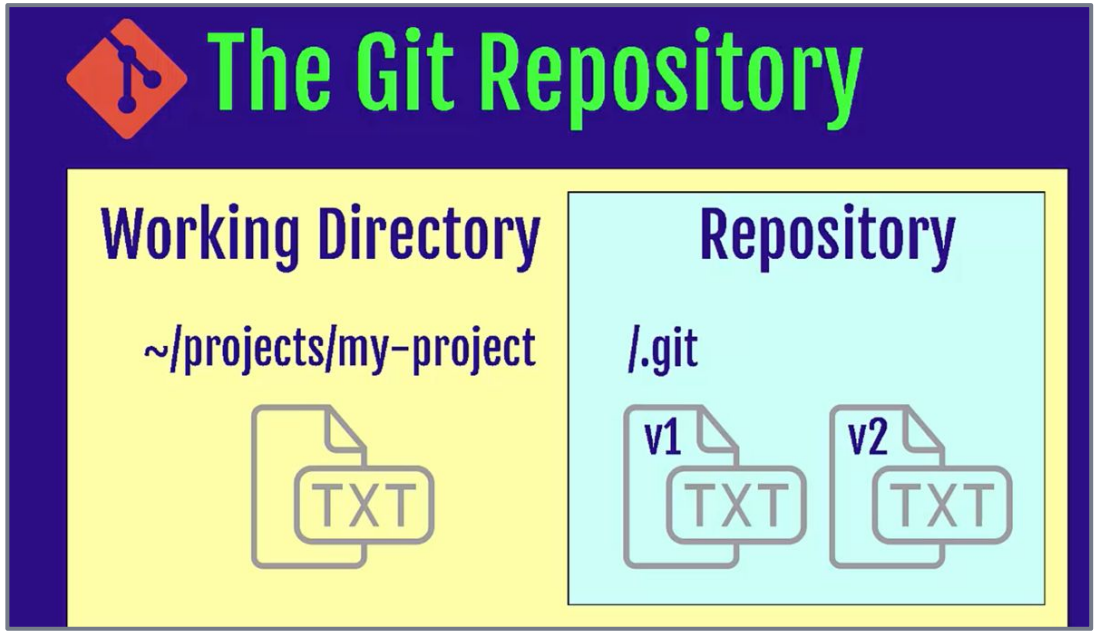
- **Git** is an **open source distributed version control system**
- **Tracks** and **records** changes to files over time (**versioning**)
- Can **retrieve** previous version of files at any time (**time travel**)
- Can be used **locally**, or **collaboratively** with others (**teamwork**)
- Contains extra information such as **date**, **author**, and **a message explaining the change**
- **Compare** and **Blame**
  - What changed
  - When it changed
  - Why it changed
  - Who changed it

Recap-Git Repository

What is a repository

- A directory or storage space where your projects can live.

- Local Repository
- Remote Repository (Central Repository)



Recap-Workflow-Git's "three trees"

Working Directory

Where you work.Create new files, edit files delete files etc.

Staging Area (Index)

Before taking a snapshot, you're taking the files to a stage. Ready files to be committed.

Repository (Commit Tree)

Committed snapshots of your project will be stored here with a full version history.

Recap-Basic Commands

git help

git init

git status

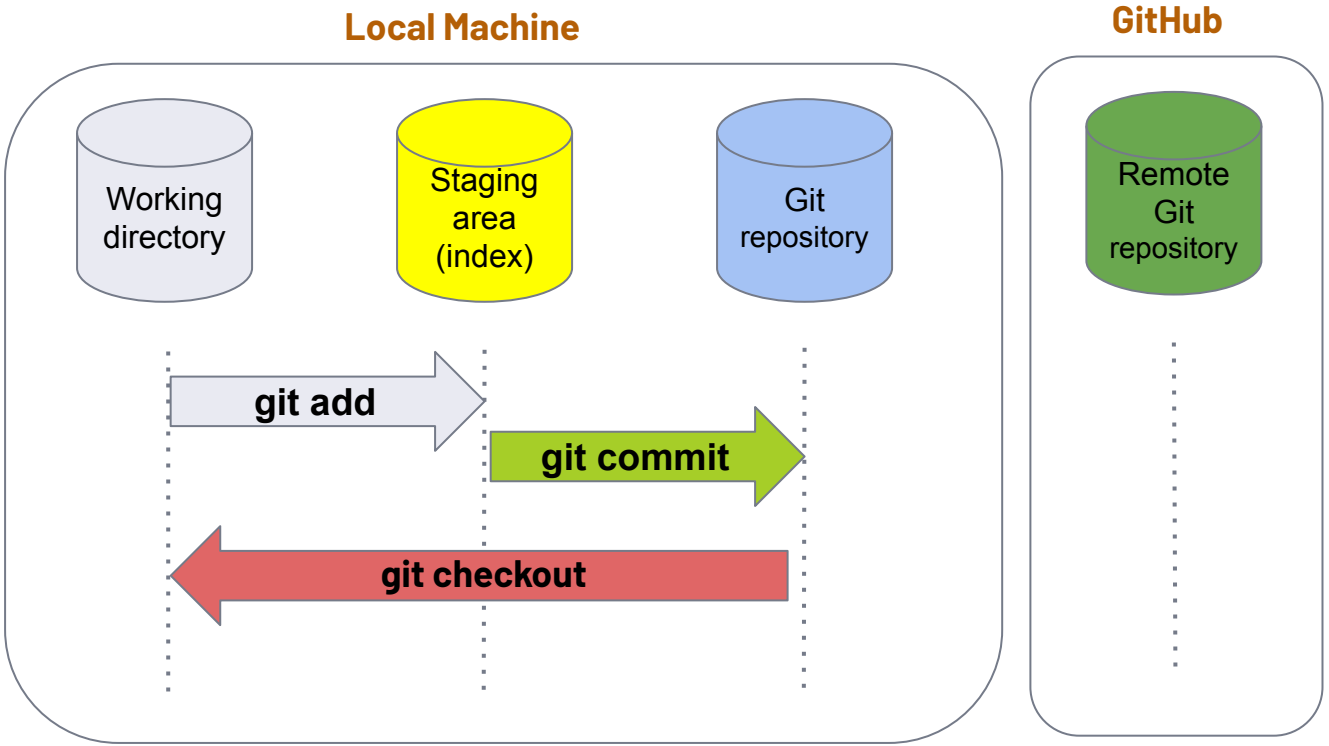
git add .

git rm --cached

git commit -m “abc”

git log

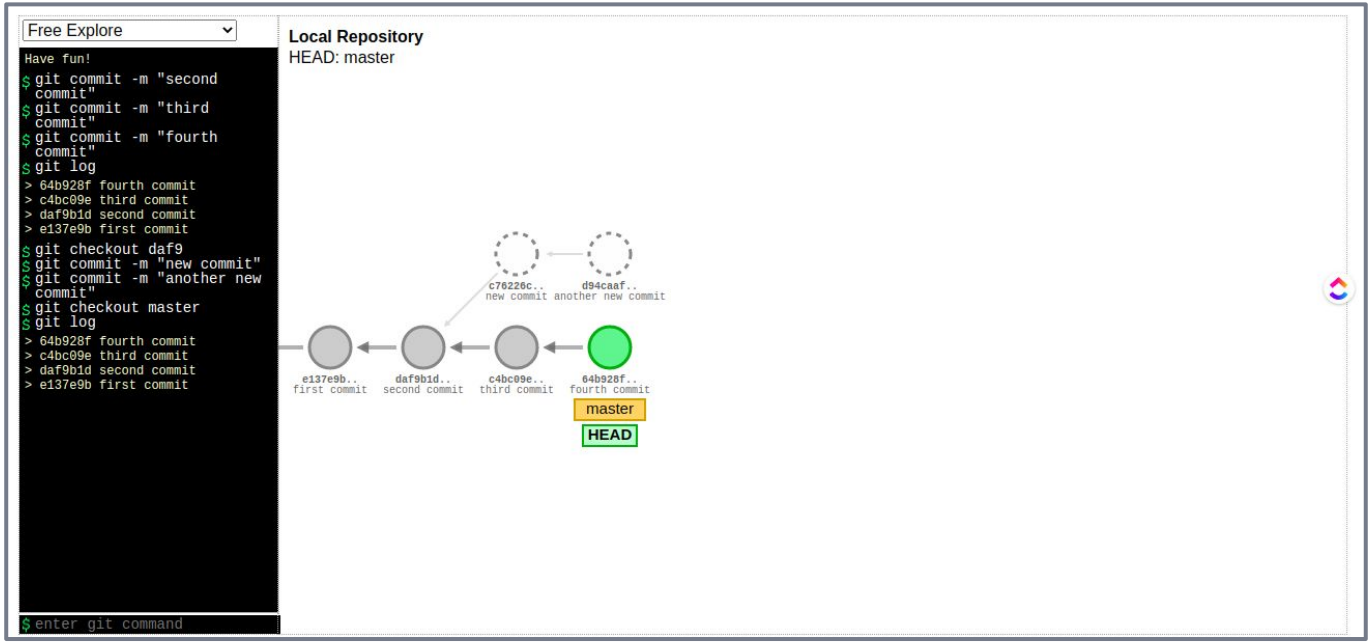
git checkout commitID



Recap-TimeTravel

https://git-school.github.io/visualizing-git/

Visualize Git illustrates what's going on underneath the hood when you use common Git operations. You'll see what exactly is happening to your commit graph.

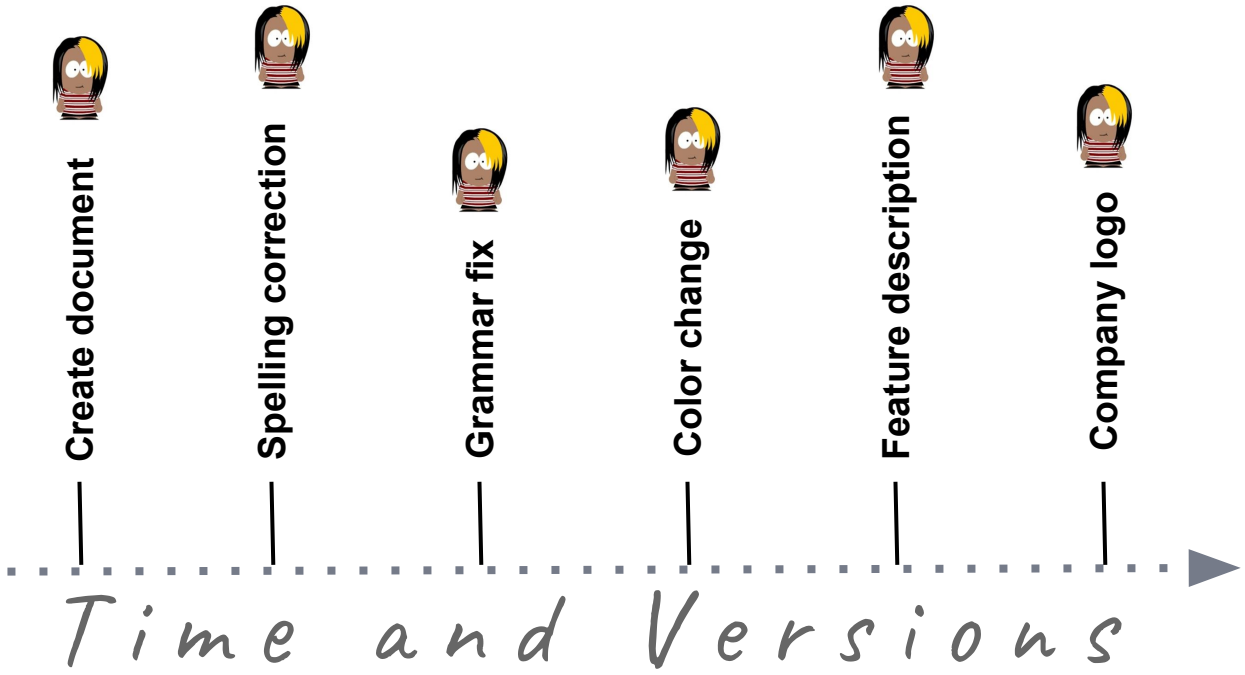


Branch, Head

What comes to you your mind when you hear this?

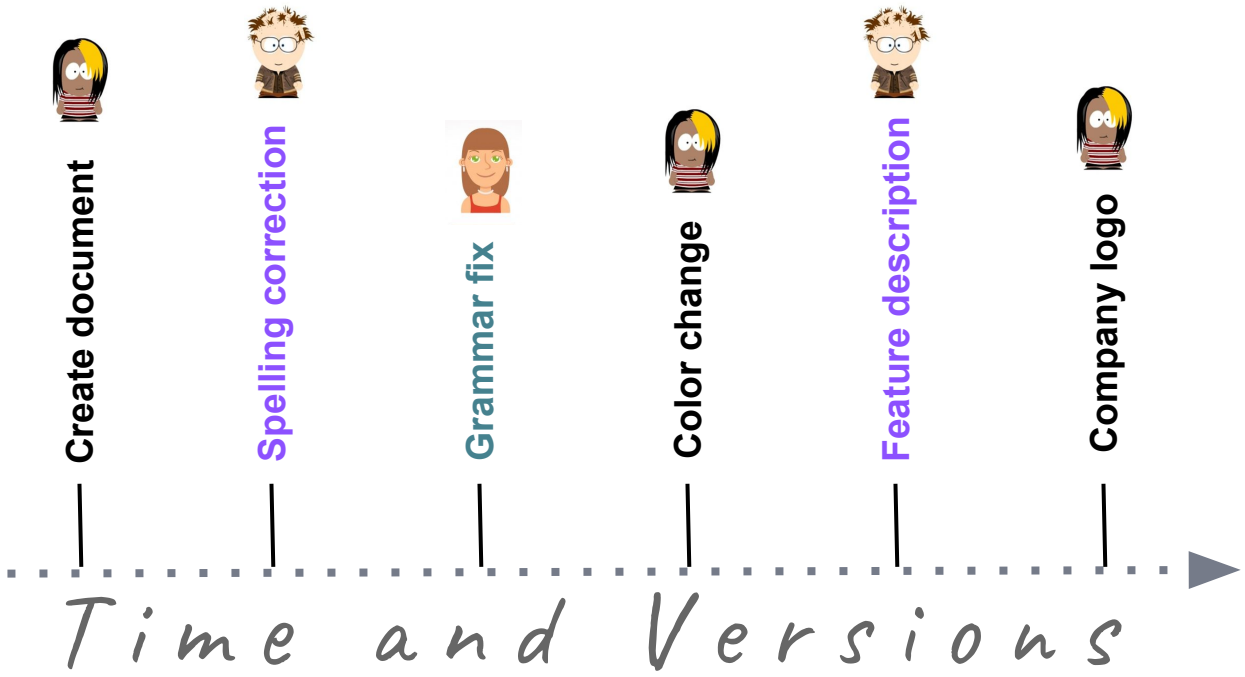
Git Branches

History Tracking



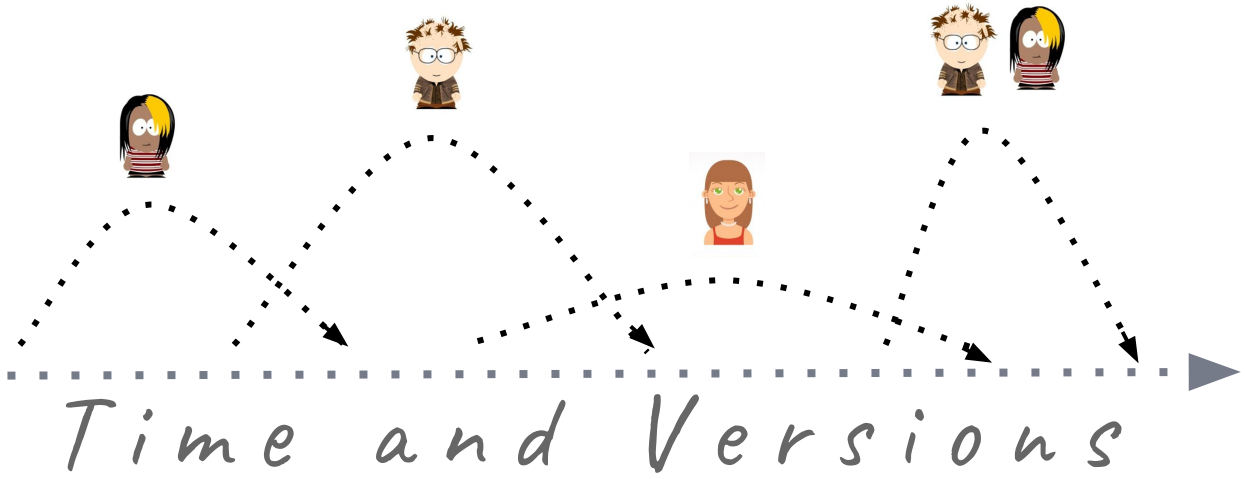
Git Branches

Collaborative History Tracking



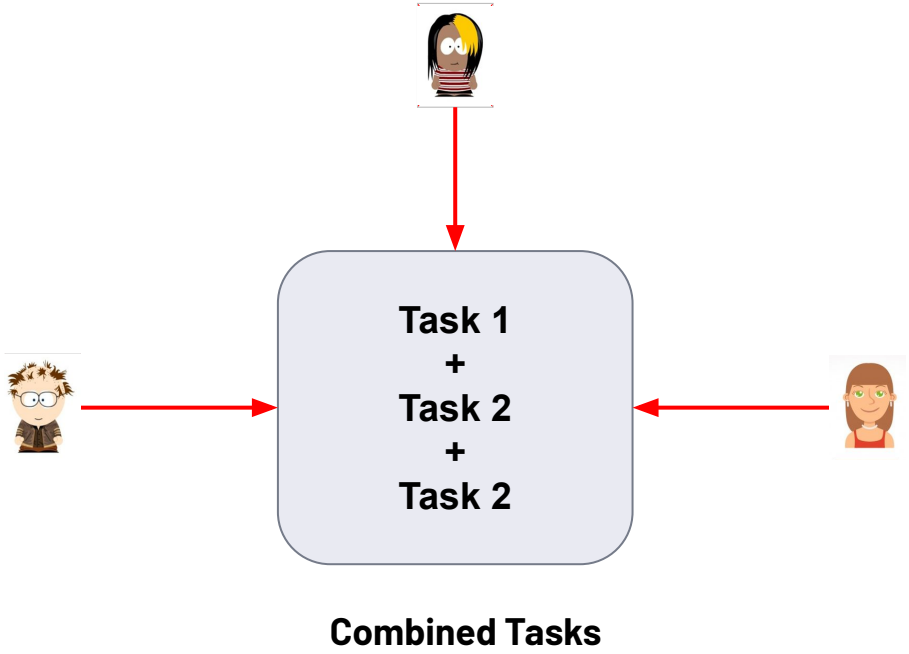
Git Branches

Collaborative History Tracking



Git Branches

Collaboration

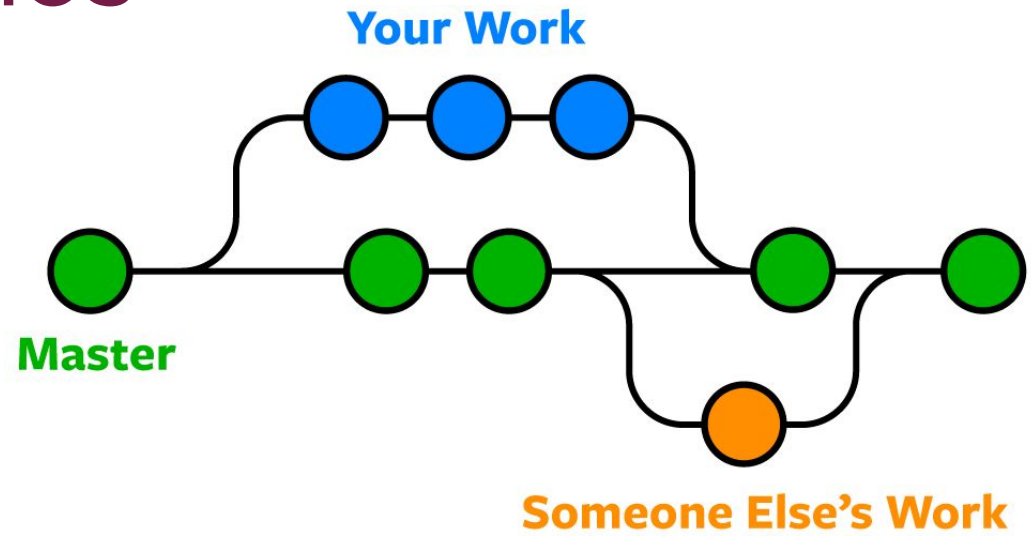


Branches

What is a Branch in Git?

- Branch in Git is similar to the branch of a tree. Analogically, a tree branch is attached to the central part of the tree called the trunk.
- While branches can generate and fall off, the trunk remains compact and is the only part by which we can say the tree is alive and standing.
- Similarly, a branch in Git is a way to keep developing and coding a new feature or modification to the software and still not affecting the main part of the project. We can also say that branches create another line of development in the project.
- The primary or default branch in Git is the master/main branch (similar to a trunk of the tree). As soon as the repository creates, so does the main branch (or the default branch).

Branches



- Production of the project lives on master/main branch
- Branches are reference to a commit

```
Erics-Mac:project eric$ git branch
* master
```

Branches

Git Refs and Heads

- In Git, a **ref** is a human readable name that references a Git commit ID. A ref is essentially a pointer to a commit. Examples of refs are Git branch names such as master and dev.

- In Git, a **head** is a ref that points to the tip (latest commit) of a branch. You can view your repository's heads in the path .git/refs/heads/. In this path you will find one file for each branch, and the content in each file will be the commit ID of the tip (most recent commit) of that branch.

Git refs and Git heads are simply pointers to commits, in the form of text files where the file name represents the name of the ref/head and the content is the commit ID that the ref points to.

Branches

- to see local branches

```
git branch
```

- to see remote branches

```
git branch -r
```

- to see all branches

```
git branch -a
```

Creating/switching branches

- create a new branch

```
git branch Branch name
```

- switch to a branch

```
git checkout Branch name
```

- create a new branch and switch to that branch

```
git checkout -b Branch name
```

Deleting Branches

Delete a local branch

```
git branch -d <BranchName>
```

```
git branch -D <BranchName>
```

The -d option only deletes the branch if it has already been merged. The -D option is a shortcut for --delete --force, which deletes the branch irrespective of its merged status.

Deleting a Branch REMOTELY

Here's the command to delete a branch remotely: git push <remote> --delete <branch>.

For example: `git push origin --delete front-end`

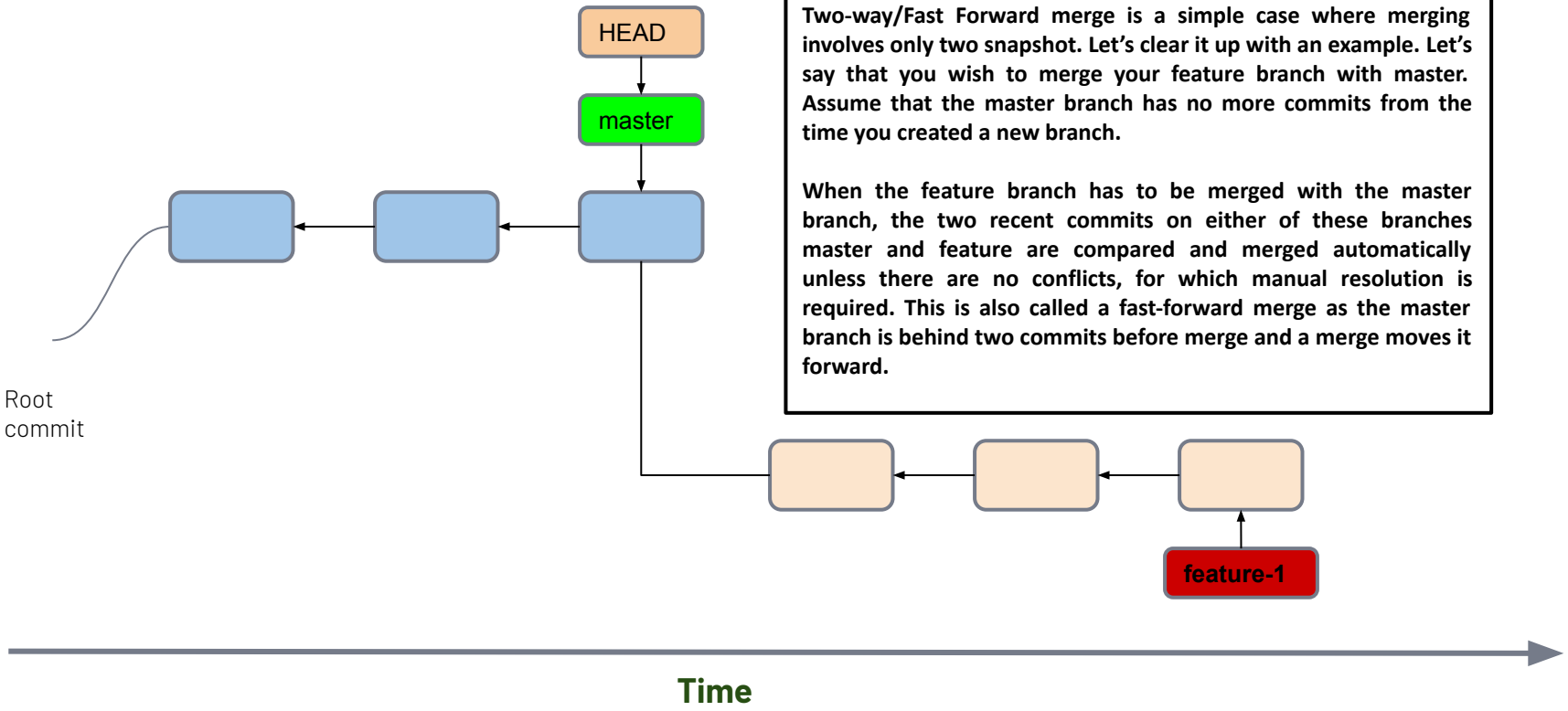
The branch is now deleted remotely.

Git Merge

What is git merge?

Git merge is Git's way of combining two separate development histories into one. In other words, you can merge code from a separate branch into the main branch to integrate the changes.

Fast forward merge

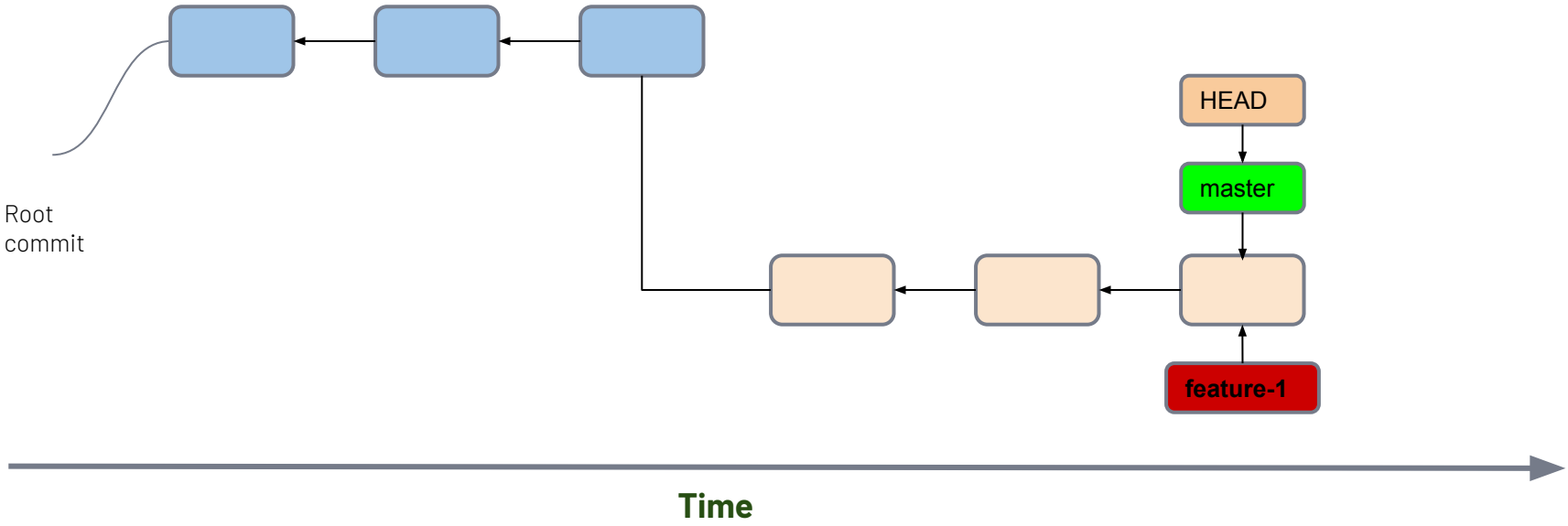


Two-way/Fast Forward merge is a simple case where merging involves only two snapshot. Let's clear it up with an example. Let's say that you wish to merge your feature branch with master. Assume that the master branch has no more commits from the time you created a new branch.

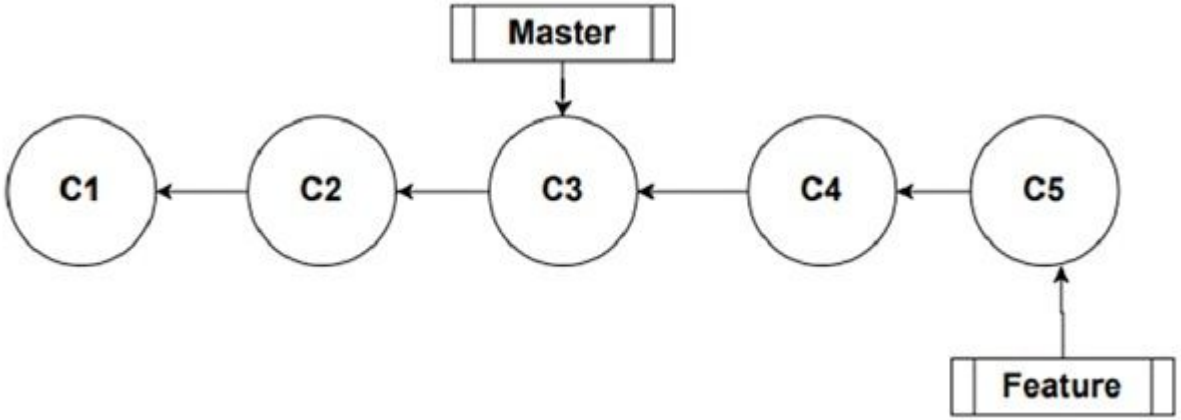
When the feature branch has to be merged with the master branch, the two recent commits on either of these branches master and feature are compared and merged automatically unless there are no conflicts, for which manual resolution is required. This is also called a fast-forward merge as the master branch is behind two commits before merge and a merge moves it forward.

Fast forward merge

git merge <feature-branch>

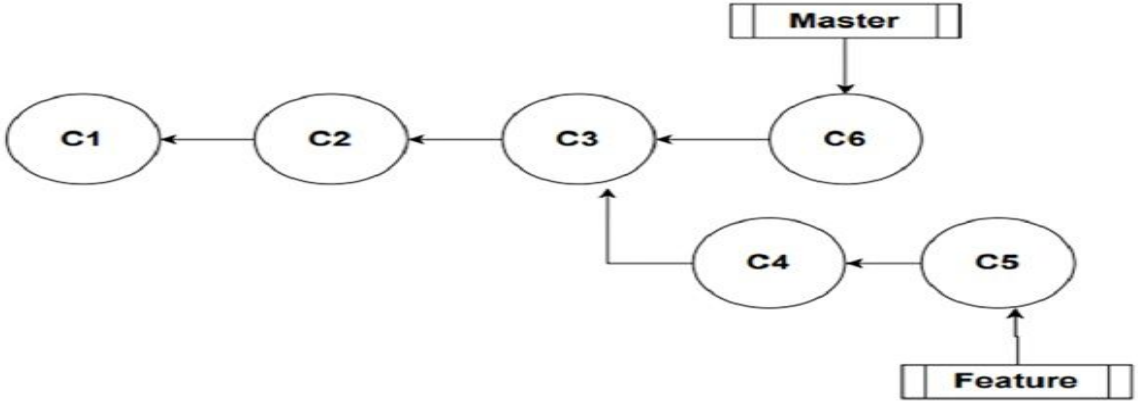


3-way merge



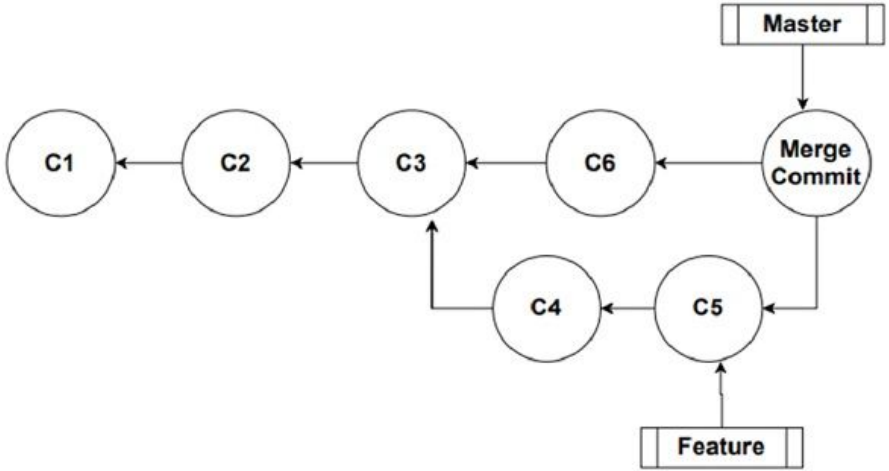
Let us look at an example of a 3-way merge. In this example, the Feature branch is two commits ahead of the Master branch.

3-way merge



- Due to the commit performed on the Master branch, both our branches Master and Feature are now diverged.
- This means we have some changes in the Master branch that is not present in the Feature branch. If we perform a merge in this case, Git cannot move the master pointer towards the Feature branch.
- If git simply moves the Master pointer to the Feature pointer, then the latest commit C6 performed on the Master branch will be lost.
- So how do we perform a merge if the branches are diverged?
- When we want to merge the branches that are diverged, Git creates a new commit (Merge Commit) and combines the changes of these two branches as shown in the next diagram.

3-way merge



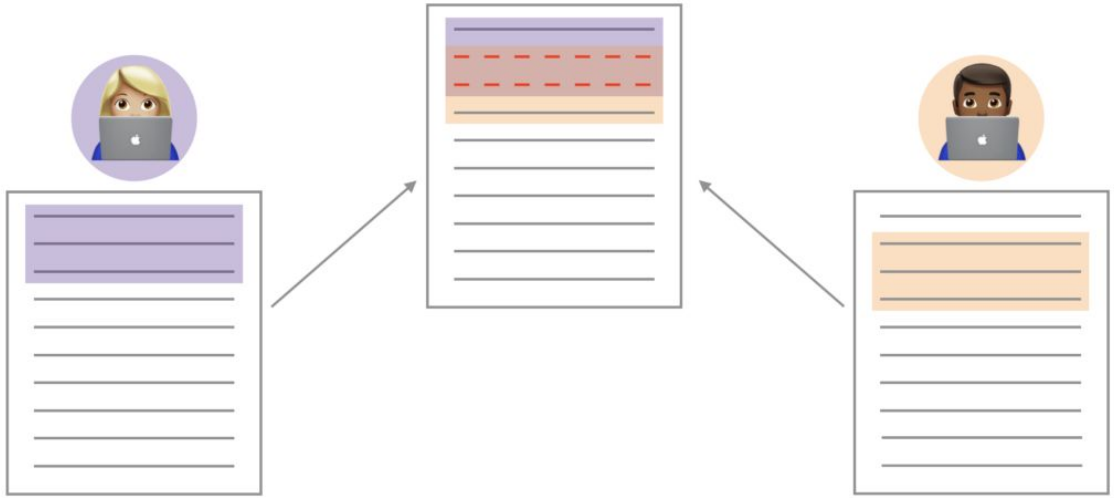
The reason it is called a 3-way merge is because the Merge Commit is based on 3 different commits.

- The common ancestor of our branches, in this case commit number C3. This commit contains code before we diverge into different branches.
- The tip of the Master branch, that is the last commit performed on the Master branch - C6
- The tip of the Feature branch, the last commit performed on the Feature branch - C5

To merge the changes from both the branches, Git looks at the three different snapshots - the before snapshot and the after snapshots. Based on these snapshots, Git combines the changes by creating the new commit called the Merge Commit.

Github - Merge Conflict

➔ **Merge conflicts** happen when you merge branches that have competing commits, and Git needs your help to decide which changes to incorporate in the final merge.



Git fork

Most commonly, forks are used to either propose changes to someone else's project to which you do not have write access, or to use someone else's project as a starting point for your own idea.

You can fork a repository to create a copy of the repository and make changes without affecting the upstream repository.

Git fork vs. clone: What's the difference?

Git clone vs. fork

- **Git clone** is primarily used to make a clone or copy of that repo to a new location.
- When a Git repository is cloned, the target repository remains shared amongst all of the developers who had previously contributed to it.
- Other developers who had previously contributed to that codebase will continue to push their changes and pull updates from the cloned repository.
- In contrast to a clone, a **Git fork** operation will create a completely new copy of the target repository. The developer who performs the fork will have complete control over the newly copied codebase.
- Developers who contributed to the Git repository that was forked will have no knowledge of the newly forked repo.
- Previous contributors will have no means with which they can contribute to or synchronize with the Git fork unless the developer who performed the fork operation provides access to them.

THANKS!

Any questions?

You can find me at:

- sumod@clarusway.com
- Slack - @sumod