Git Branches

CLARUSWAY©
WAY TO REINVENT YOURSELF

Objectives

- Branches
- Merges
- Conflicts

1

Recap- Git Workflow

Recap-What is Git?

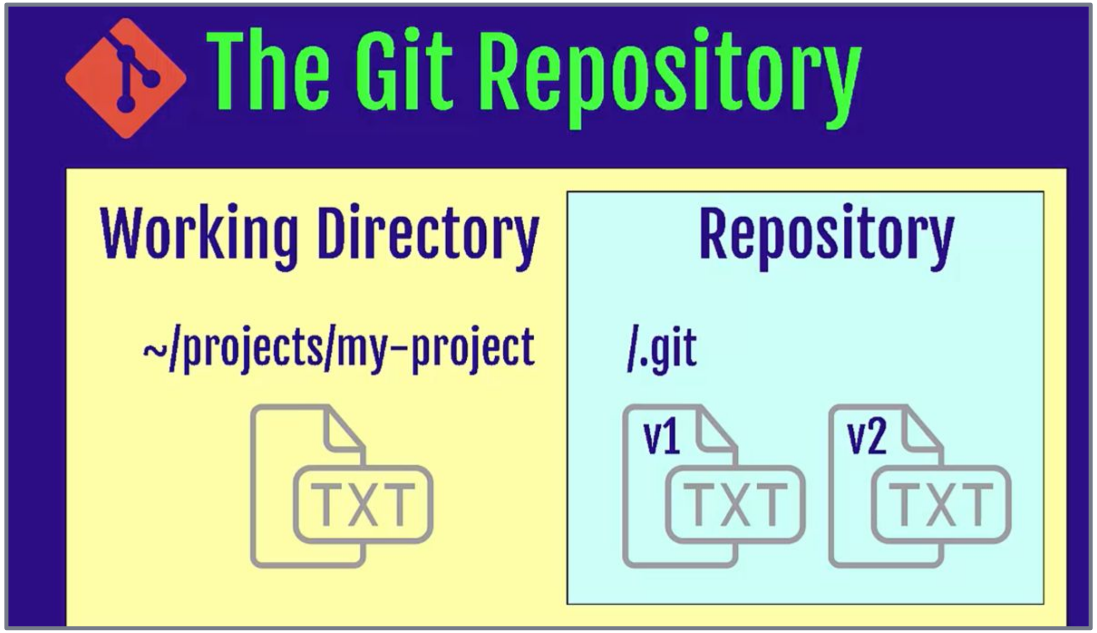
- **Git** is an **open source distributed version control system**
- **Tracks** and **records** changes to files over time (**versioning**)
- Can **retrieve** previous version of files at any time (**time travel**)
- Can be used **locally**, or **collaboratively** with others (**teamwork**)
- Contains 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)



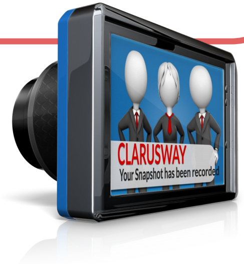
Recap-Workflow-Git's "three trees"

Working Directory

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

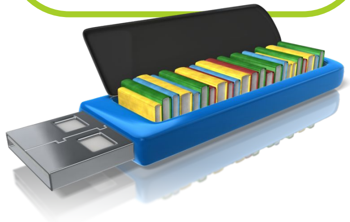
Staging Area (Index)

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

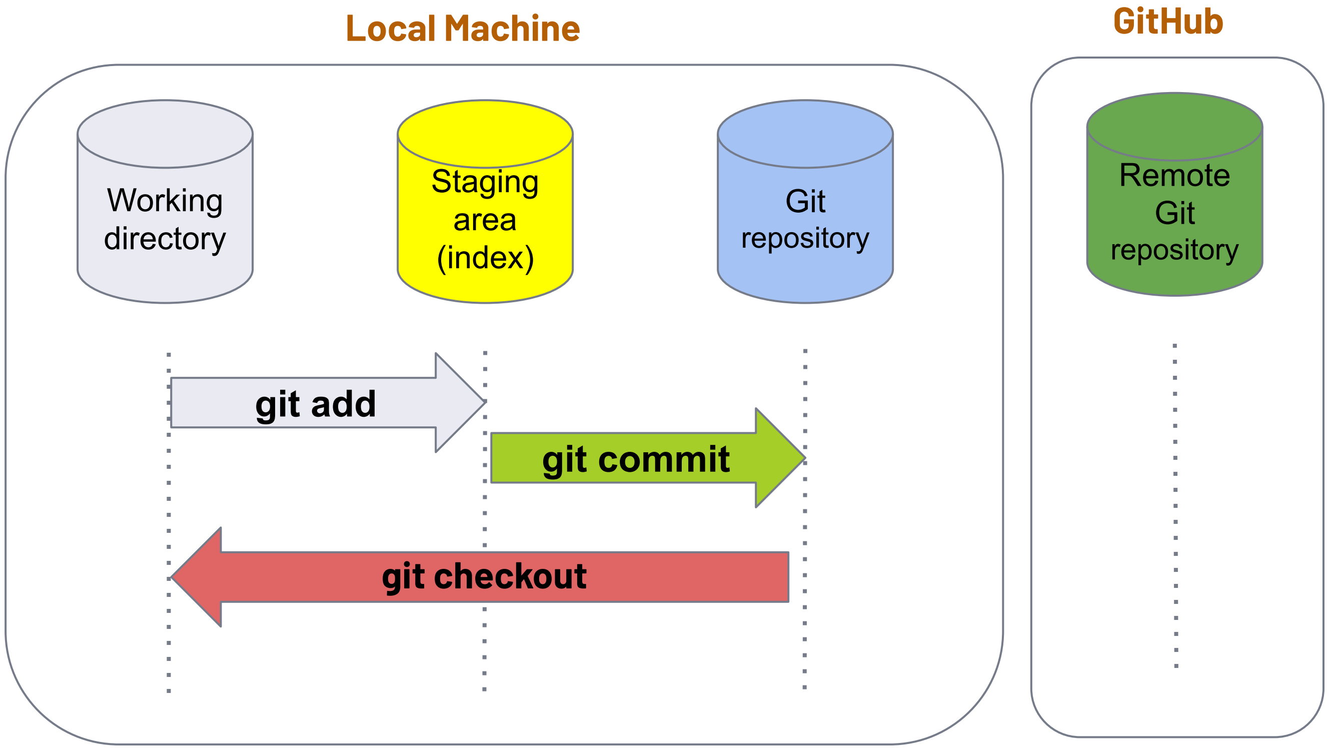
git add .

git rm --cached

git commit -m “abc”

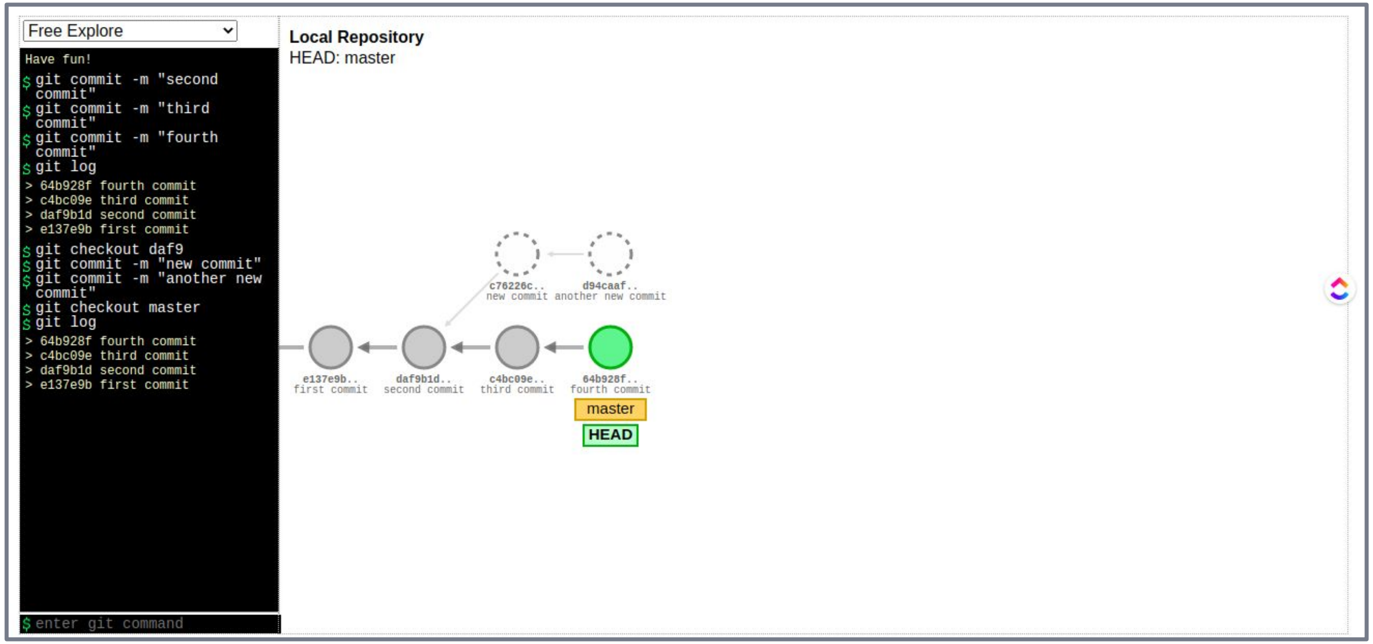
git log

git checkout commitID



Recap-Timetravel

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



```
Have fun!
$ git commit -m "second
  commit"
$ git commit -m "third
  commit"
$ git commit -m "fourth
  commit"
$ git log
> 64b928f fourth commit
> c4bc09e third commit
> daf9b1d second commit
> e137e9b first commit
$ git checkout daf9
$ git commit -m "new commit"
$ git commit -m "another new
  commit"
$ git checkout master
$ git log
> 64b928f fourth commit
> c4bc09e third commit
> daf9b1d second commit
> e137e9b first commit
```

CLARUSWAY©
WAY TO REINVENT YOURSELF

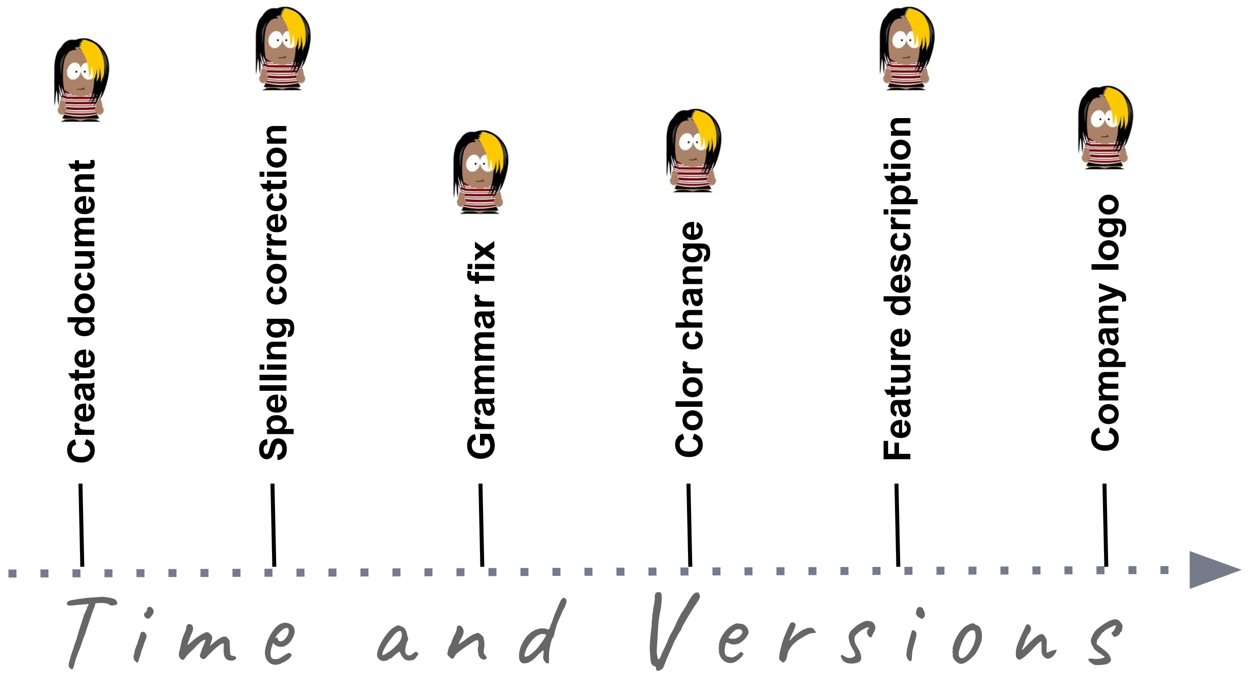
Git

Branch, Head

What comes to you your mind when you hear this?

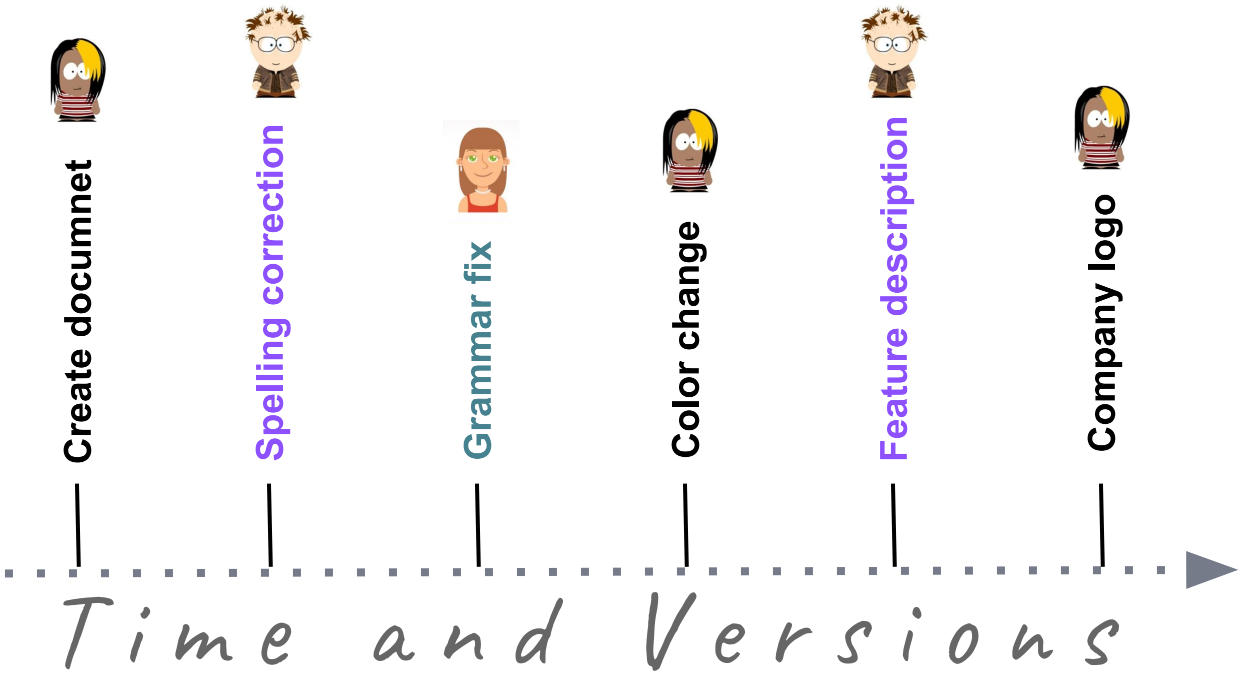
Git Branches

History Tracking



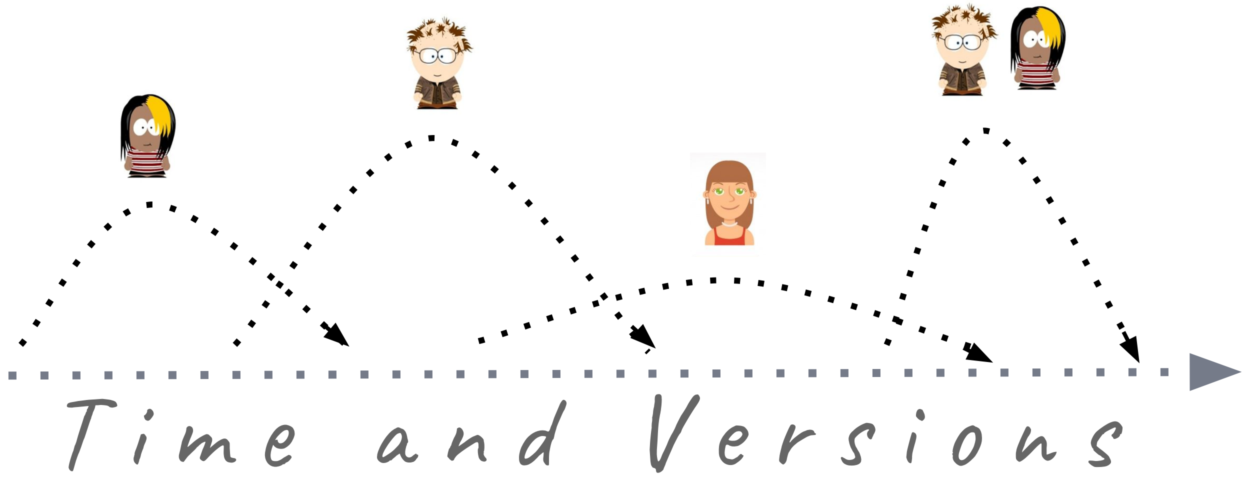
Git Branches

Collaborative History Tracking



Git Branches

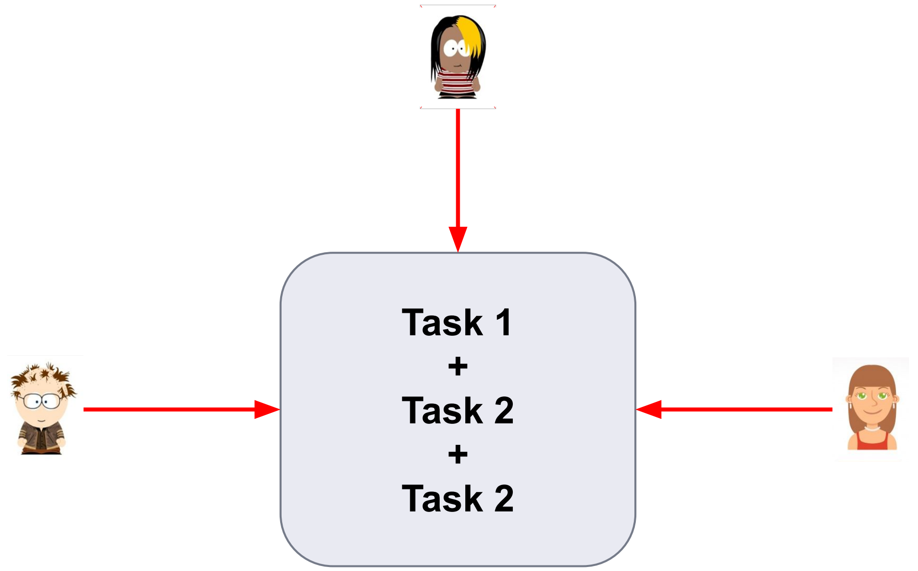
Collaborative History Tracking



CLARUSWAY©
WAY TO REINVENT YOURSELF

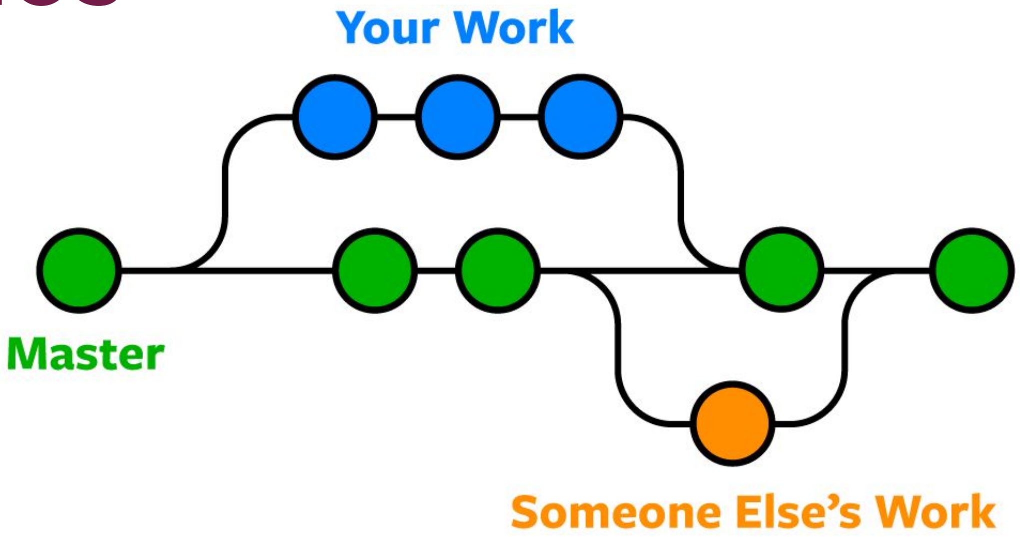
Git Branches

Collaboration

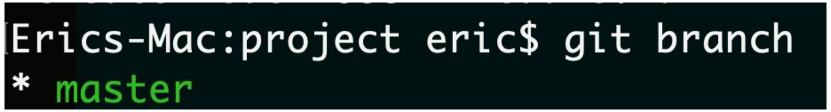


Combined Tasks

Branches



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



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

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

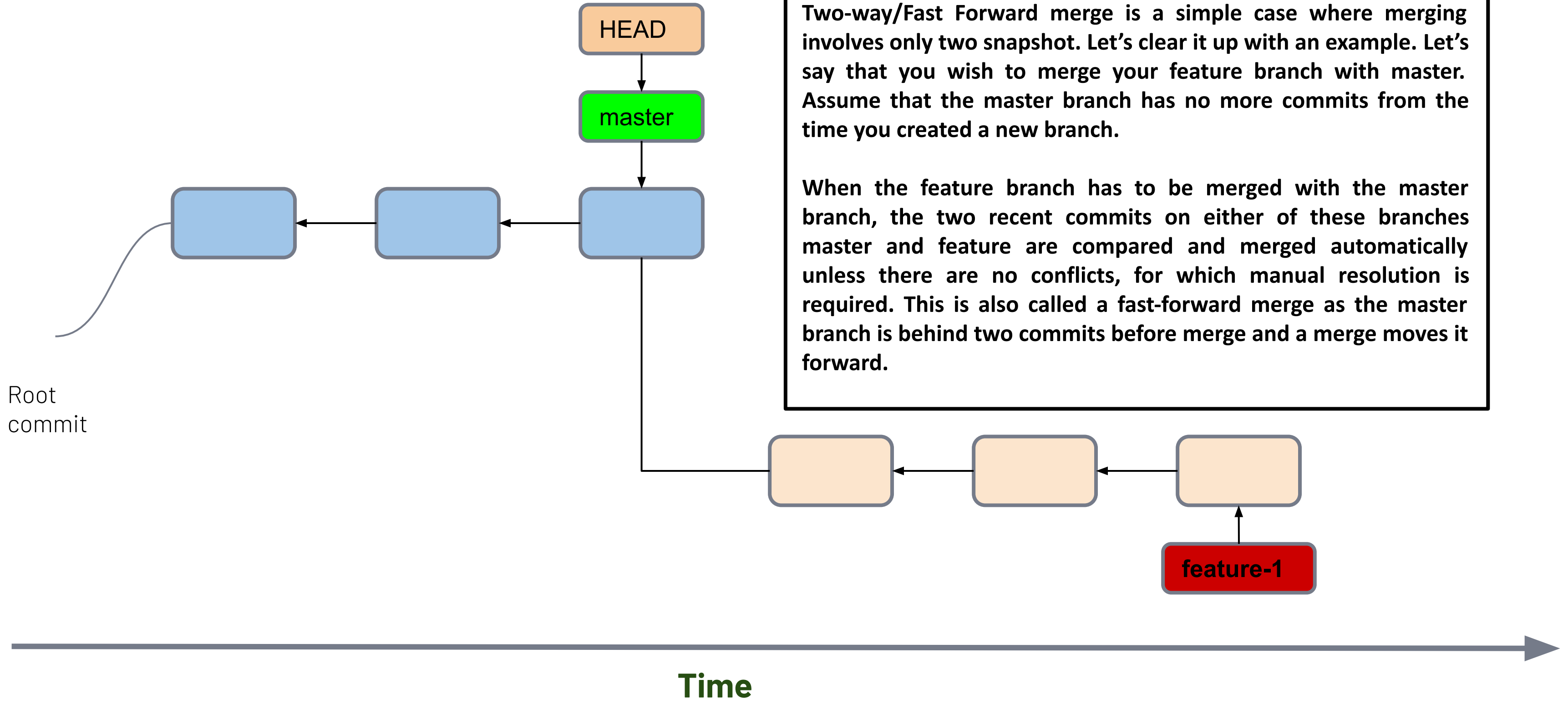
→ delete a local branch

```
git branch -d Branch name
```

```
git branch -D Branch name
```

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.

Fast forward merge

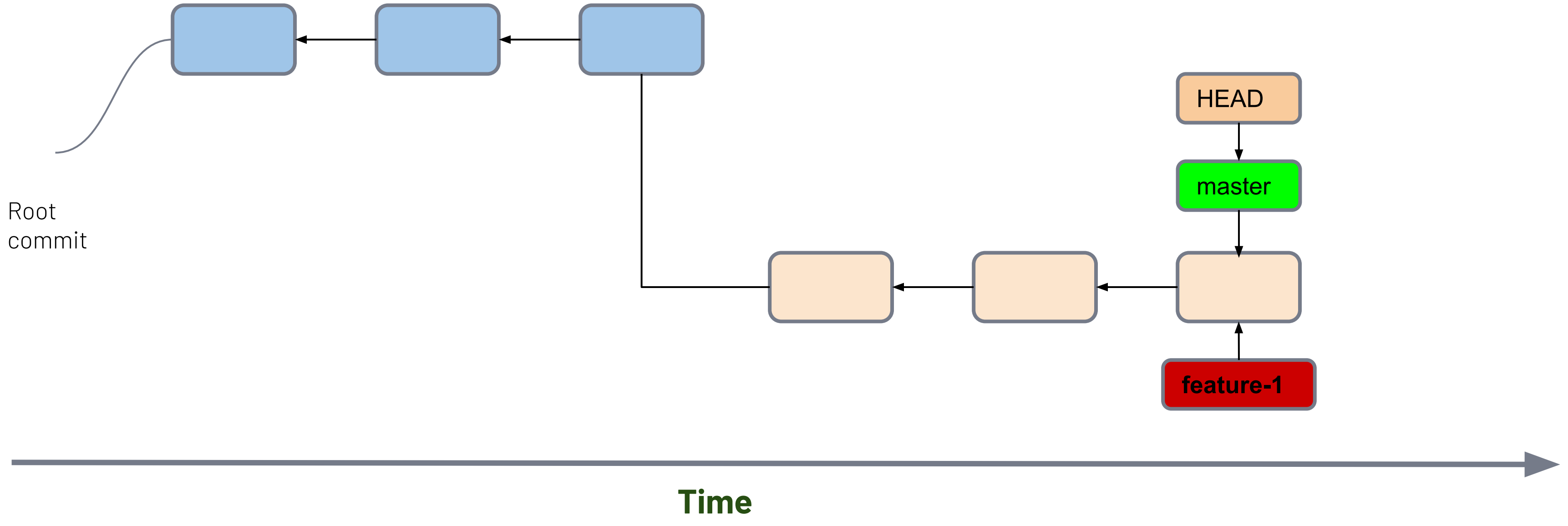


Two-way/Fast Forward merge is a simple case where merging involves only two snapshot. Let's clear it up with an example. Let's say that you wish to merge your feature branch with master. Assume that the master branch has no more commits from the time you created a new branch.

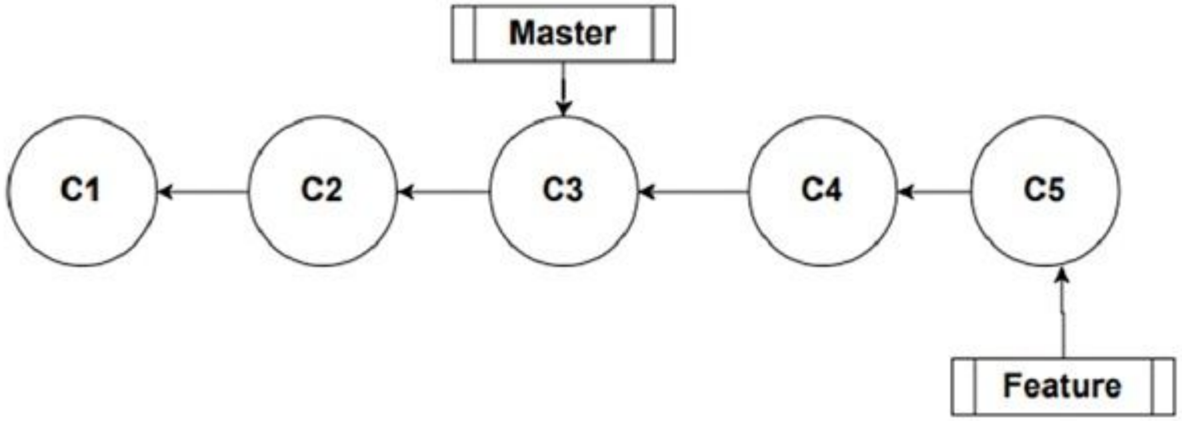
When the feature branch has to be merged with the master branch, the two recent commits on either of these branches master and feature are compared and merged automatically unless there are no conflicts, for which manual resolution is required. This is also called a fast-forward merge as the master branch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

Fast forward merge

```
git merge <feature-branch>
```



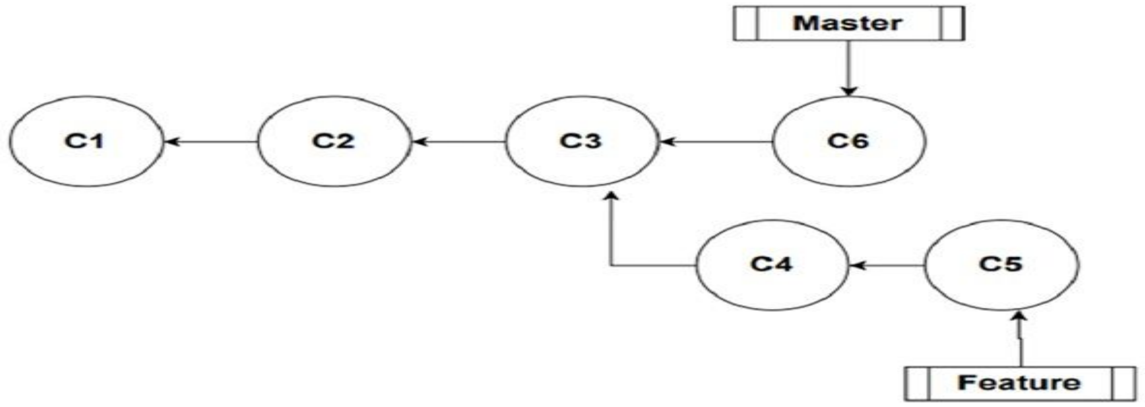
3-way merge



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.

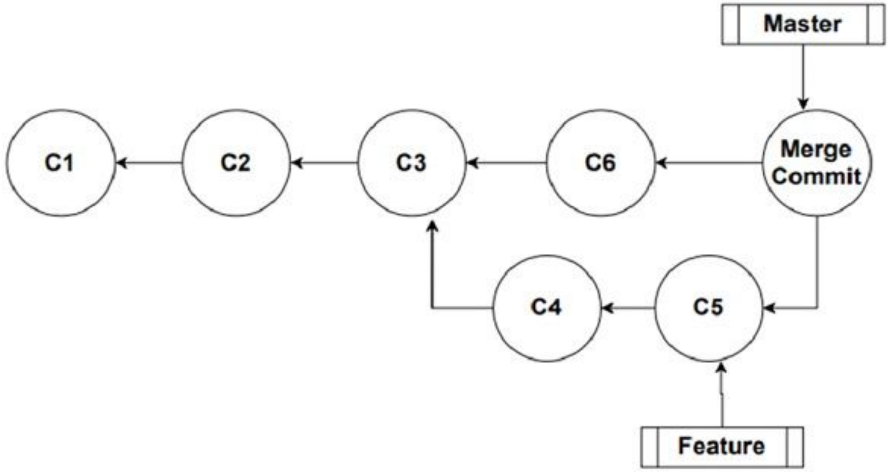
3-way merge

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.



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 below diagram.

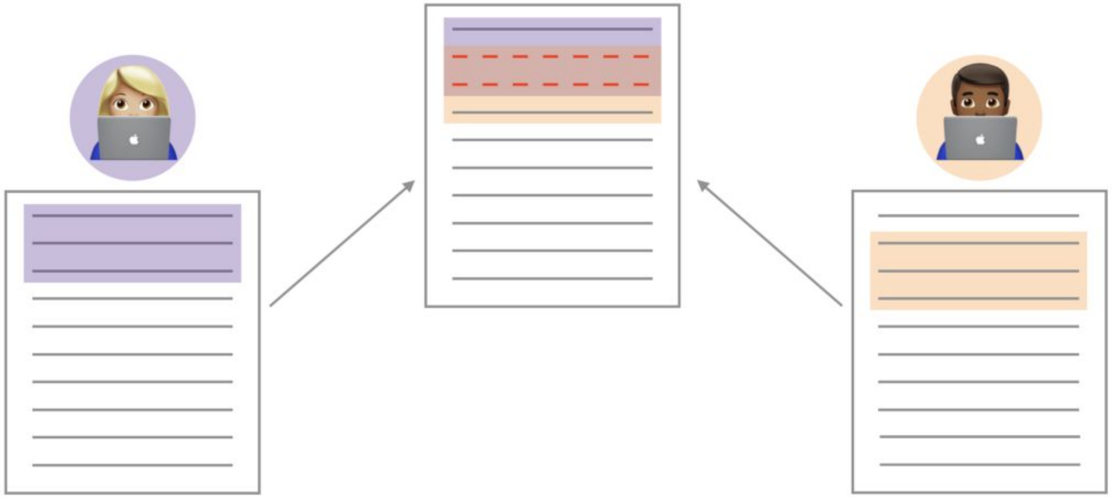
Merge Conflicts



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.



THANKS!

Any questions?

You can find me at:

- sumod@clarusway.com
- Slack - @sumod