# Introduction to Git

# Core idea

General take outs of this talk

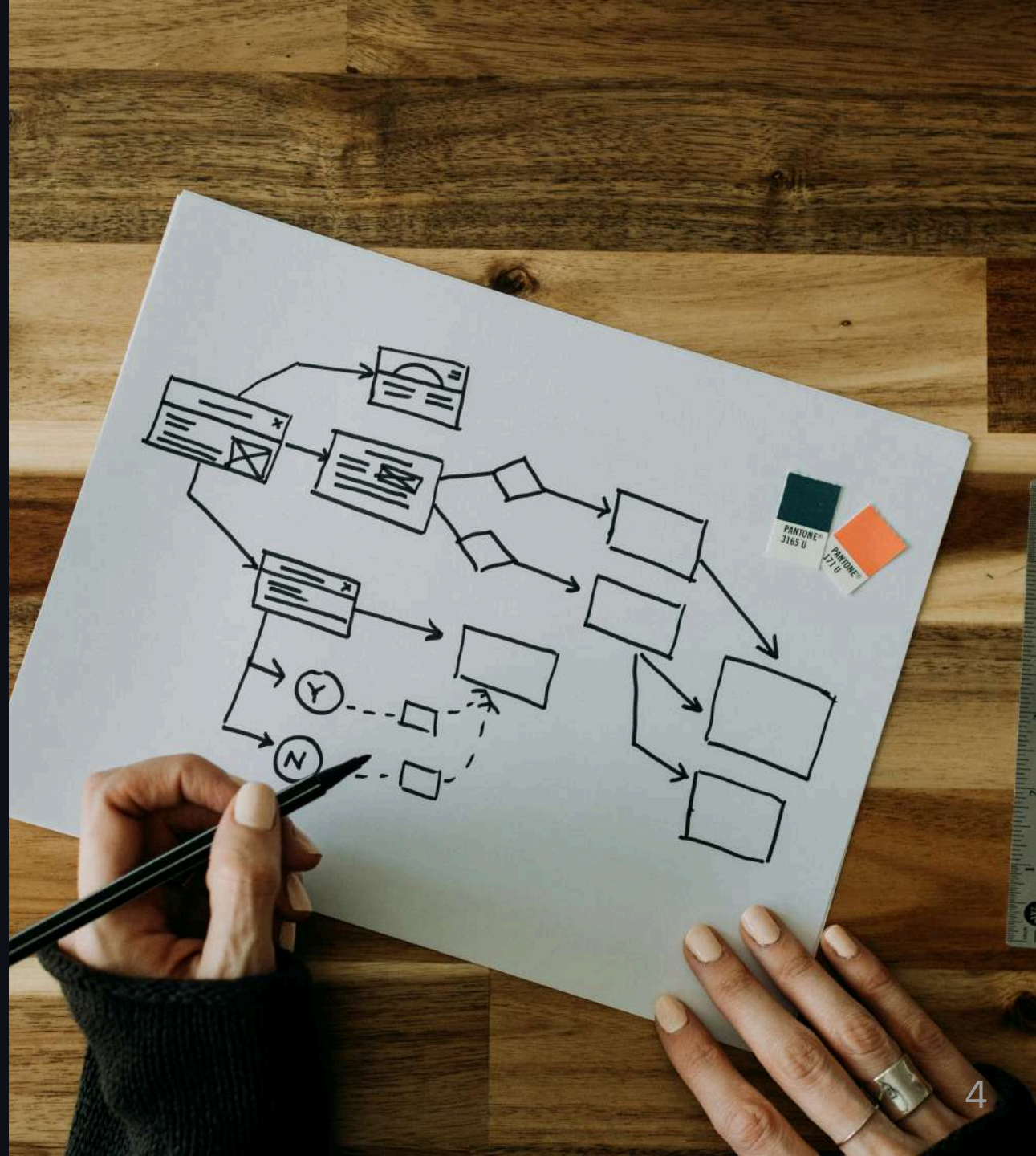
1. Common knowledge on Version Control System
2. Basic Local Git usage
3. Branches / Tags / Merge
4. Common conflicts
5. Remote Git usage
6. Git tools

## Main objective of this talk

You should have an understanding of the basics concepts of Git philosophy

Giving you the tools to read various tutorials that can be found online and not feel lost

# Introduction to VCS



## What is a VCS ?

- **V**ersion **C**ontrol **S**ystem (VCS)
- Track specific changes over time
- Collaborative Development : Team Projects (through branches, conflict resolutions)

**Helps coordinating parallel work and managing project among individual and teams.**

## Why would we need a versioning system

A bit of context :
Imagine a code with 500k lines, if we wanted to print it, assuming we put 100 lines of code / page, this would take 20 books of 600 pages !

- Collaborative edition
- Collaborative development
- Duplicates non-modified parts of the code
- Impossible to track bugs and correct them for everyone.

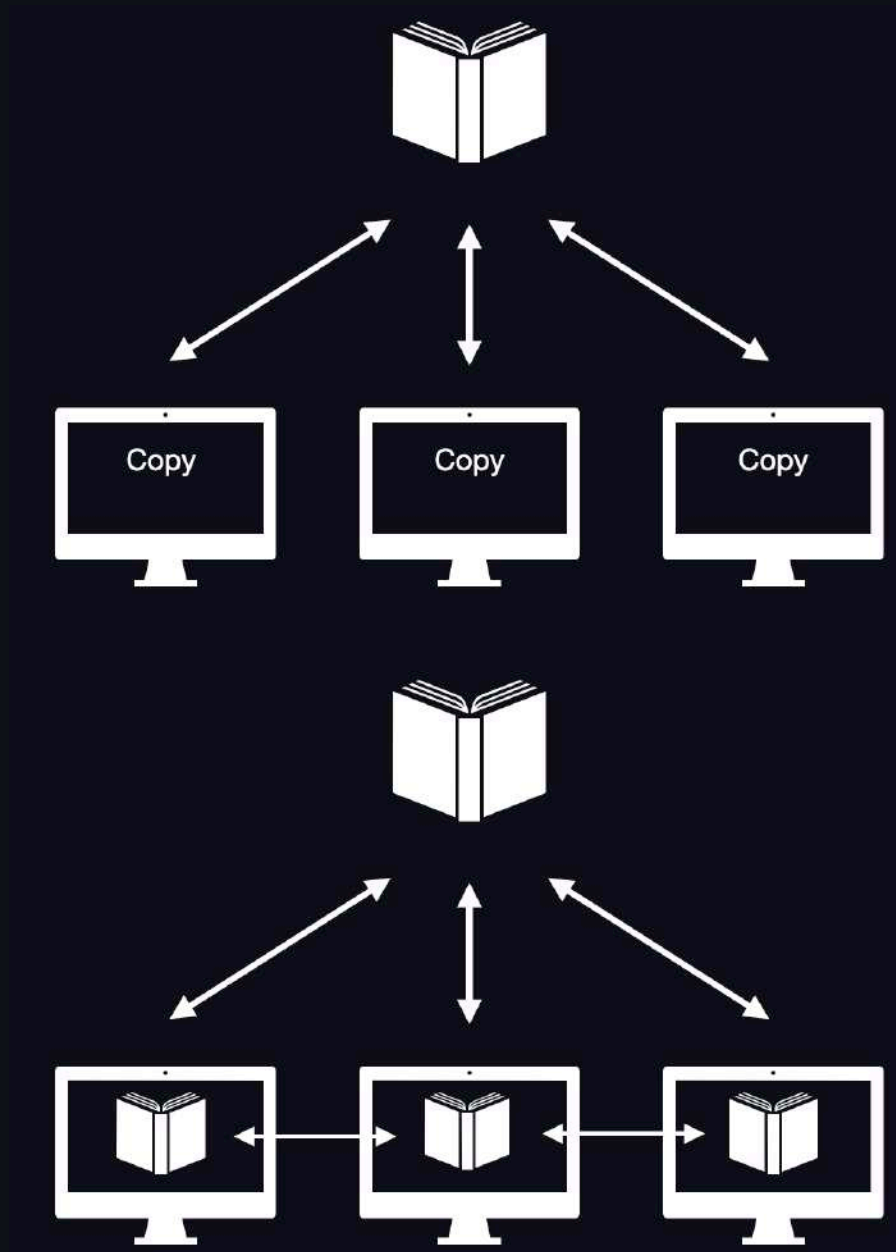
# Why use Git ?

CVS or SVN

- Centralized : one server
- Slow for big changes

GIT

- Distributes the project amongst each users with its historic
- Offline access
- User versioning possible
- Allow you to work without interfering with others work



# Disclaimer

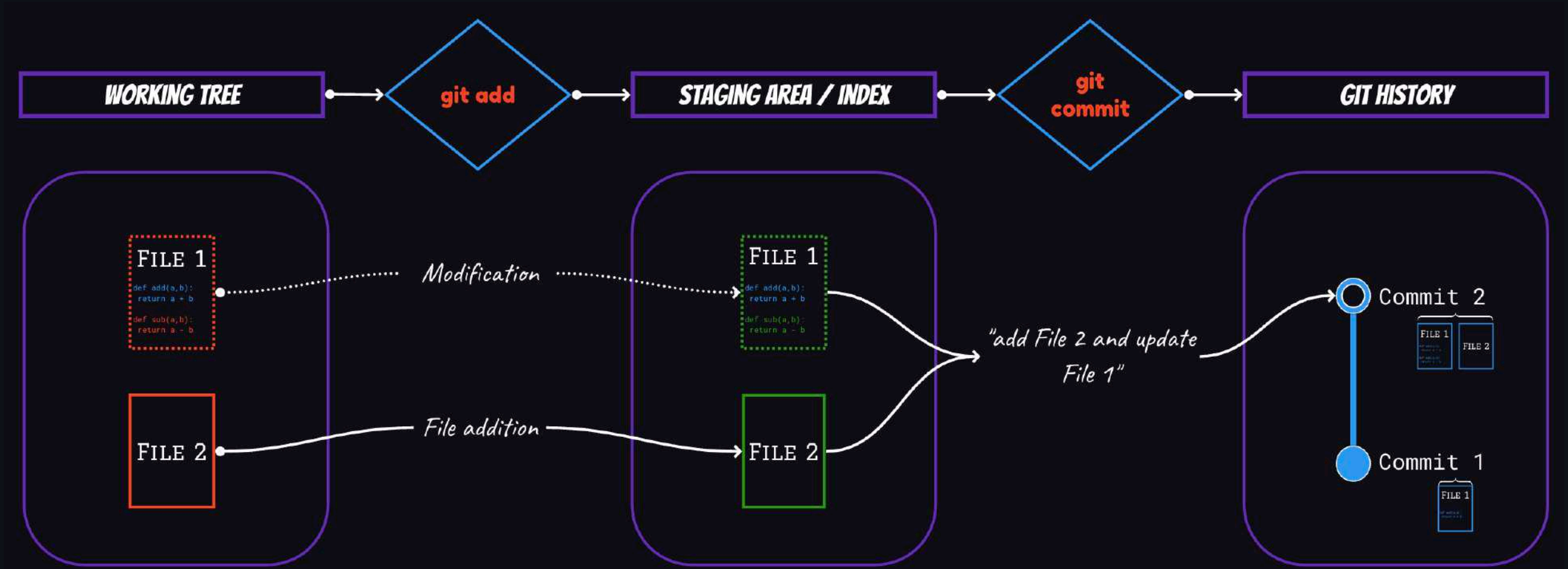
- Versioning is completely dependent on your usage.
- Can not be learnt from a book, must be used on the fly
- But requires some basics to avoid getting lost ...
- Google is your friend .. If you have the problem someone of somewhere had it at some point as well.
- There is never just one solution / option to do something

**The idea here is to give you a general understanding of the Git logic**

# Git journey

## 3 conceptuals ideas of a git repository

- **The Working Tree** : what you see in your file system, when you add, delete or edit files.
- **The Stage Area (or Index)** : allow you to pick the snapshots you want for your next commit. Only those changes will appear in the git history.
- **The Git history** : equivalent to commit graph, stored in hidden directory ***.git***. This holds the metadata, giving it to someone is equivalent to giving your whole project including the history to this someone.


WORKING TREE
git add
STAGING AREA / INDEX
git commit
GIT HISTORY
FILE 1
def add(a,b):
return a + b
def sub(a,b):
return a - b
Modification
FILE 2
File addition
FILE 1
def add(a,b):
return a + b
def sub(a,b):
return a - b
FILE 2
"add File 2 and update File 1"
Commit 2
FILE 1
FILE 2
Commit 1
FILE 1

## What is a repository ?

A collection of files and directories along with metadata stored in a specific directory on your local machine or a remote server.

Creation of first repository :

- ***git init*** : your folder is now a repository too
- ***.git*** directory will be created


WORKING TREE
STAGING AREA / INDEX
GIT HISTORY
FILE 1

# Initializing you Git ID

- Allow identification of name, email and timestamps for a commit.
- Important to know where and when a change has occured

*git config --global user.name "YourName"*
*git config --global user.email "YourEmail"*

*Note: Unique for all user, not easily changeable, stored in the .gitconfig file*

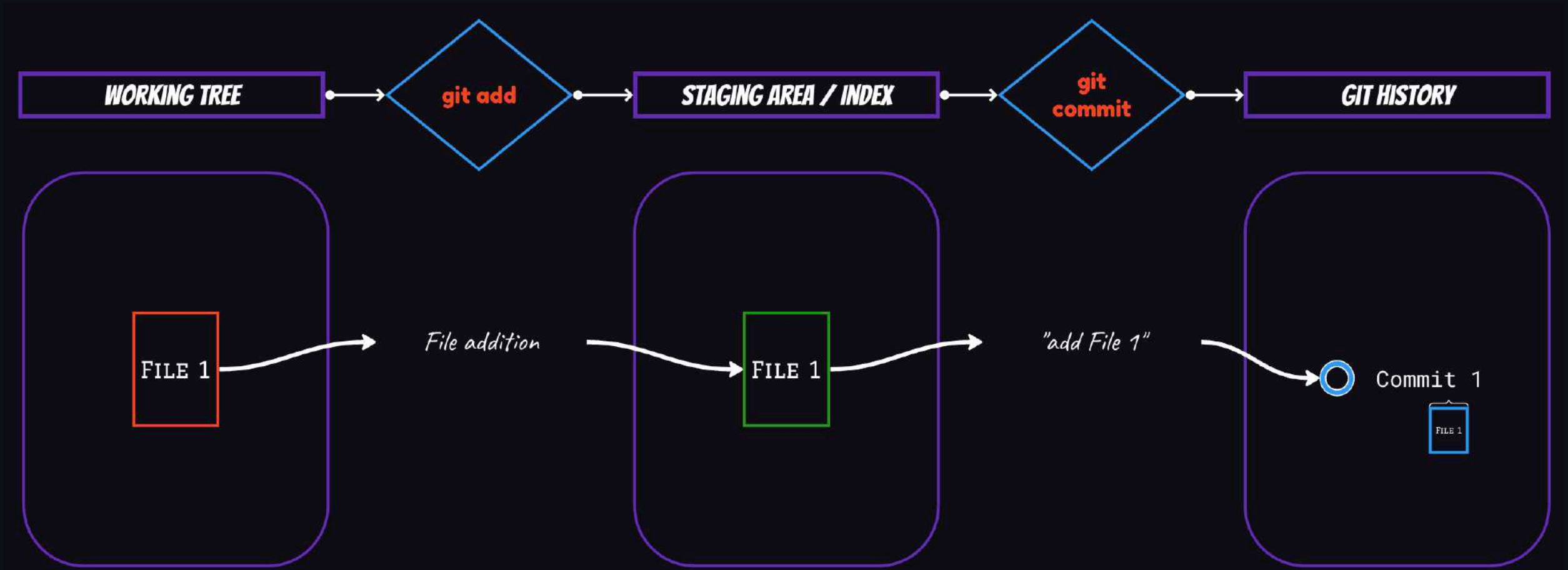
## What is a commit ?

A commit is a snapshot of the project at a specific point in tim

For now, File 1 is ***untracked*** because git isn't doing any tracking on it yet

- Running `git status` : tell us how things stand in the working tree and staging area
- Running `git add` (Working tree -> Staging)
- Running `git commit -m "commit_message"` (Staging -> Git History) allow us to take the snapshot
- Running `git log` info about the commit graph

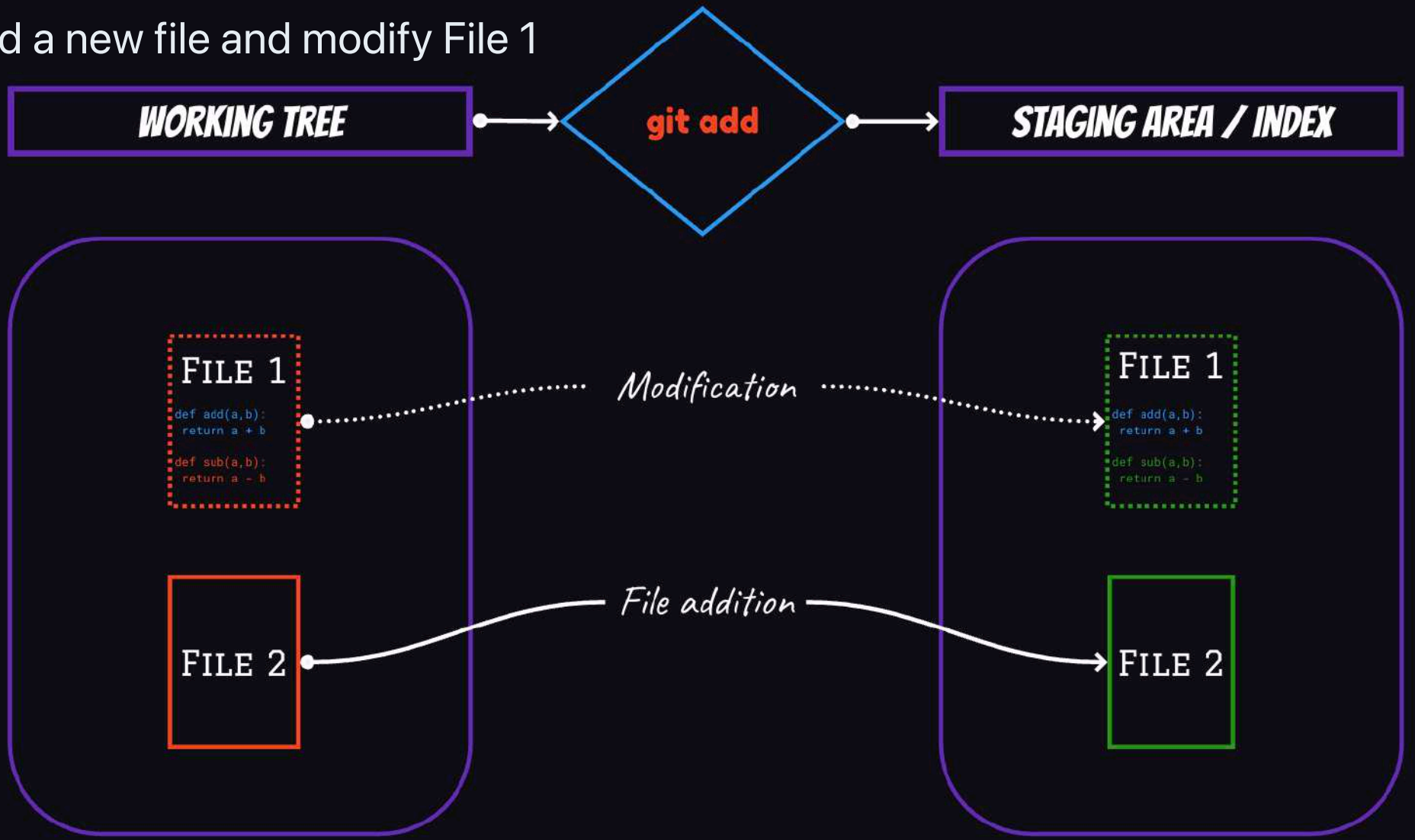

WORKING TREE
STAGING AREA / INDEX
GIT HISTORY
FILE 1


WORKING TREE
git add
STAGING AREA / INDEX
git commit
GIT HISTORY
FILE 1
File addition
FILE 1
"add File 1"
Commit 1
FILE 1

# The commit tree

Let's add a new file and modify File 1

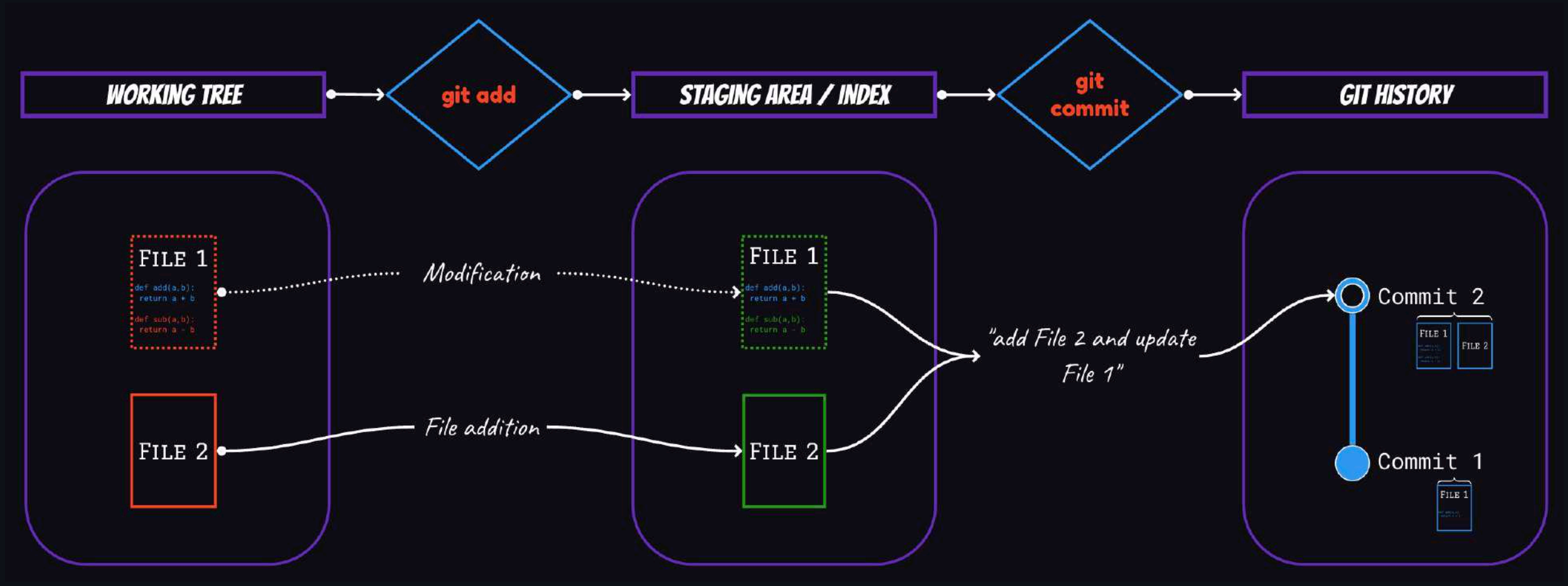
Let's add a new file and modify File 1



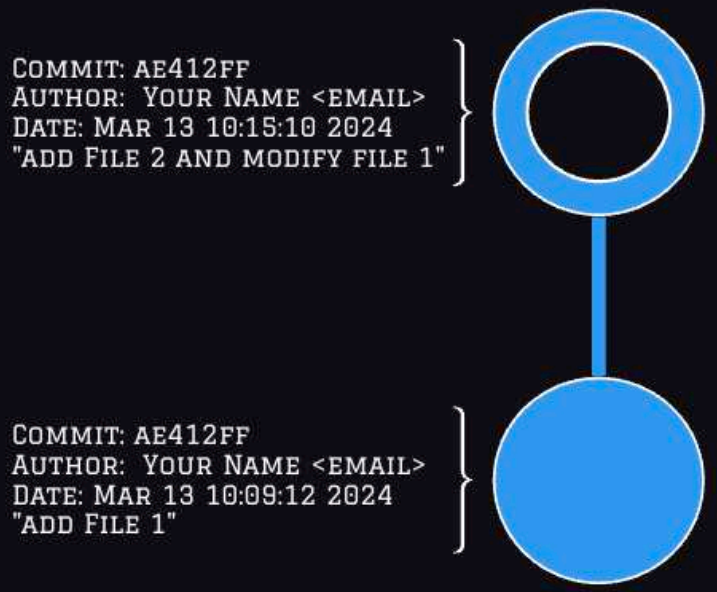
# A few more usefull commands

Running `git diff` will highlight the changes between working tree and Staging area

Running `git diff --staged` will highlight the changes between staging and git history

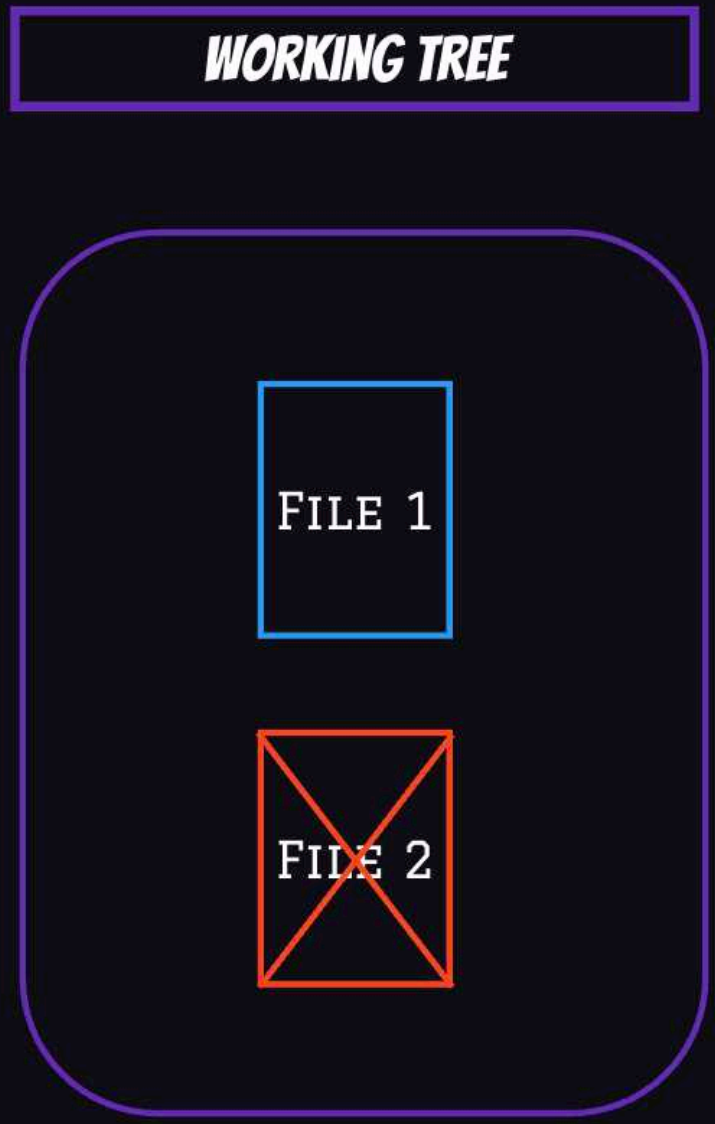

WORKING TREE
git add
STAGING AREA / INDEX
git commit
GIT HISTORY
FILE 1
def add(a,b):
return a + b
def sub(a,b):
return a - b
Modification
FILE 2
File addition
FILE 1
def add(a,b):
return a + b
def sub(a,b):
return a - b
FILE 2
"add File 2 and update File 1"
Commit 2
FILE 1
FILE 2
Commit 1
FILE 1

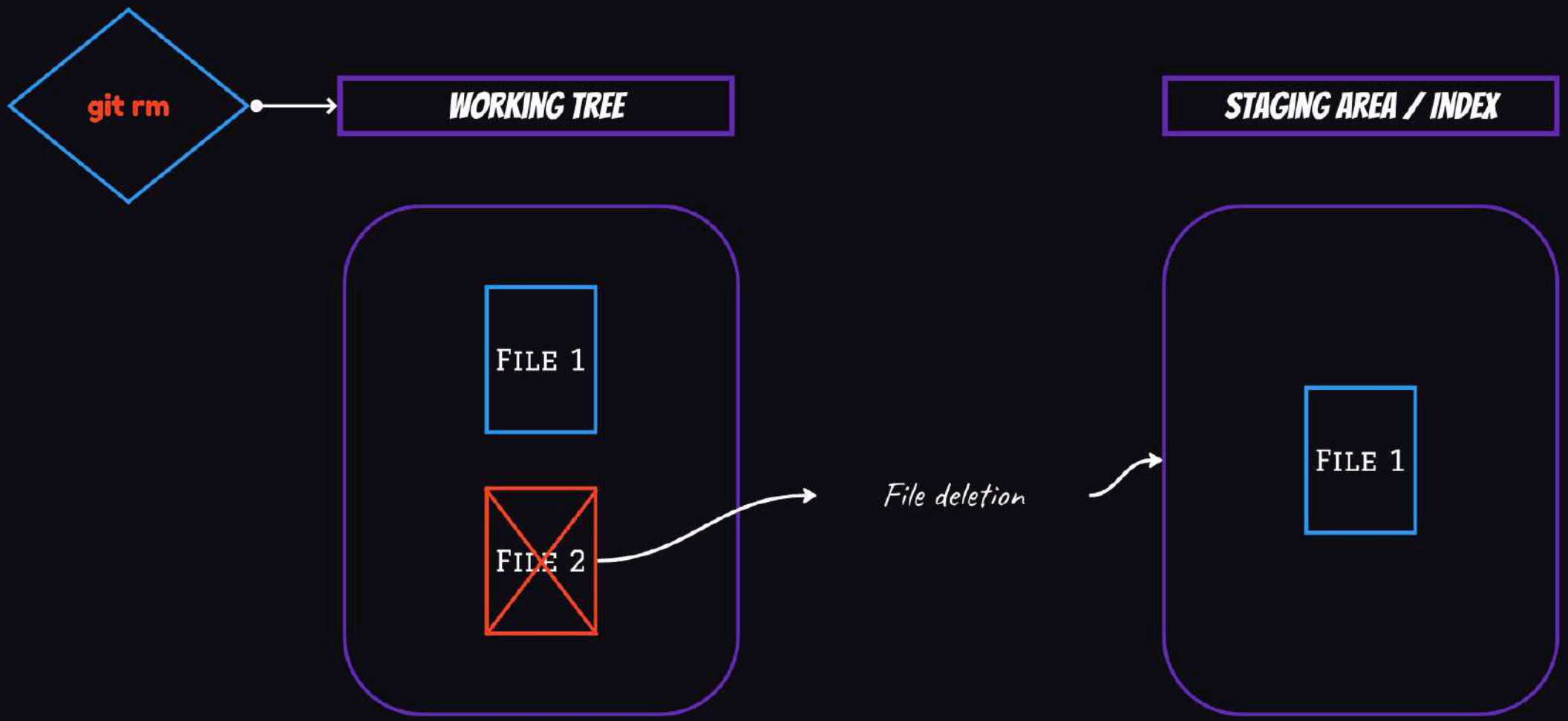
# Updated commit tree

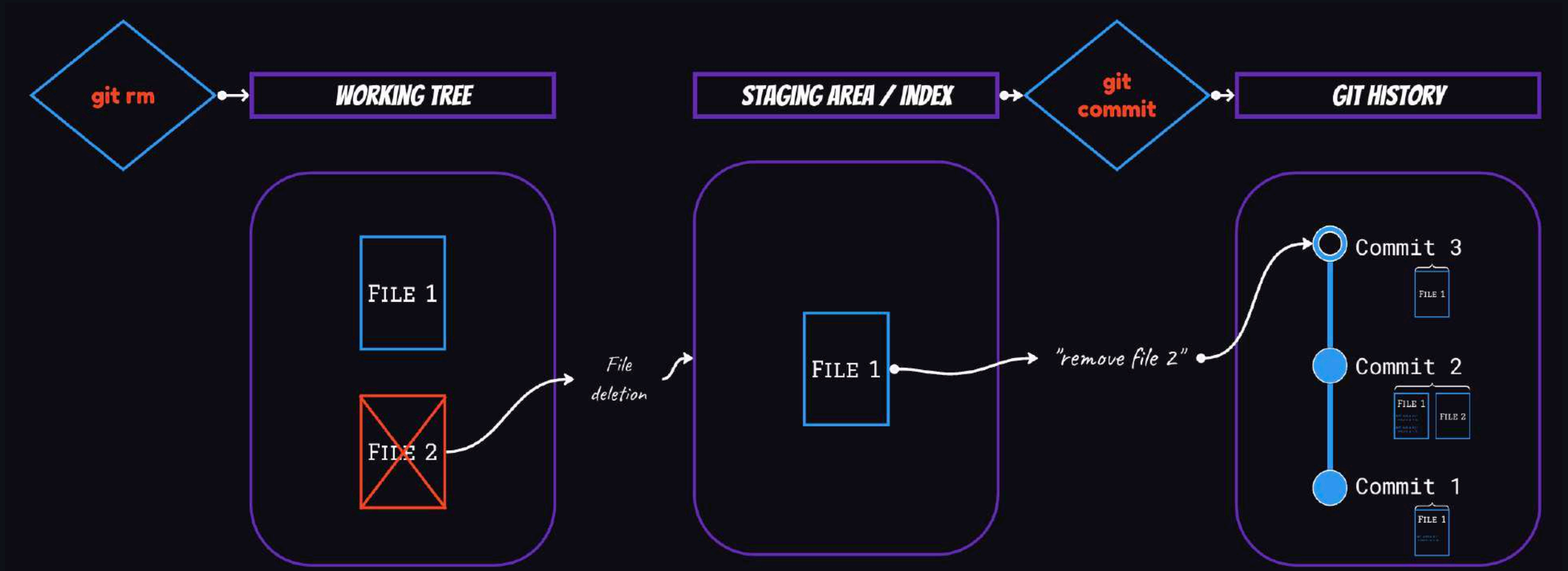


# Remove a file in git

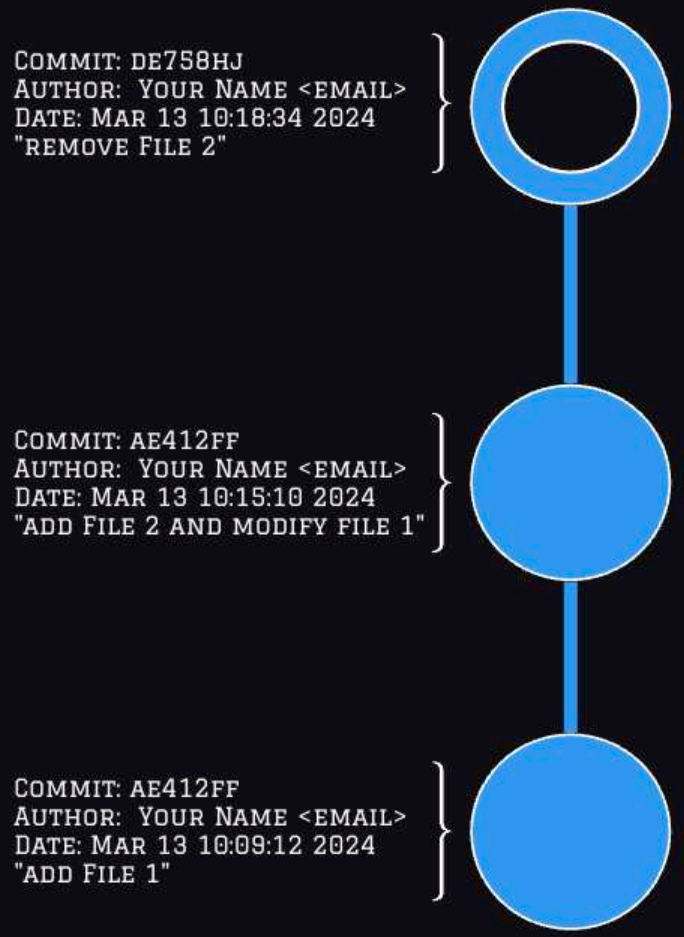
The command `git rm` will remove from working tree and stage the change it


WORKING TREE
FILE 1
FILE 2


git rm
WORKING TREE
STAGING AREA / INDEX
FILE 1
FILE 2
File deletion
FILE 1


git rm
WORKING TREE
STAGING AREA / INDEX
git commit
GIT HISTORY
FILE 1
FILE 2
File deletion
FILE 1
"remove file 2"
Commit 3
FILE 1
Commit 2
FILE 1
FILE 2
Commit 1
FILE 1

# Updated commit tree



# Summary

| Command | Description |
| --- | --- |
| `git init` | Initialize a new Git repository |
| `git add <file>` | Add file(s) to the staging area |
| `git commit -m "commit message"` | Commit staged changes with a message |
| `git status` | Show the status of files in the working directory |
| `git log` | Show commit history |
| `git diff` | Show changes between stage area and working tree |
| `git diff --staged` | Show changes between staged area and git history |
| `git rm <file>` | Remove a file from version control |

# Branching and Merging

## What is a branch ?

- Use of branches to work on different versions of the same file in parallel
- A branch inherits all the history of the branch: it's the child of up to the moment it was created i.e. it's an independant line of development
- Allows you to work on bug fixes or new features for exemples without affecting the ***master*** branch
- Changes can then be reintegrated in the ***master*** branch later

# Implementation of branch

By default: git created the ***master*** branch with `git init` command

# HEAD

- Way for Git to know where we are
- Normaly points to a branch (could also be a commit, a tag)
- Could be called a ***symbolic pointer***
- In git terminology : HEAD pointer tells us where we have ***checked out***

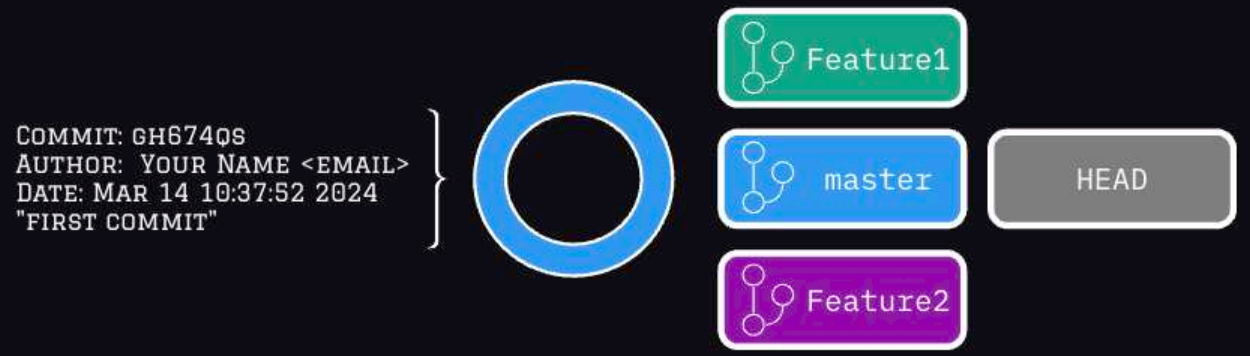
## Creation of 2 branches

- *git branch Feature_1*
- *git branch Feature_2*

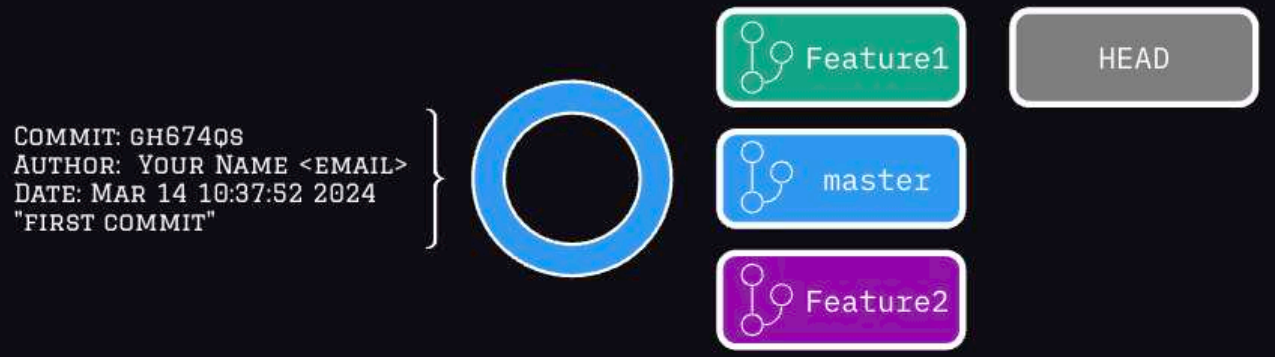
***git checkout <branch_name>*** : move the pointer HEAD to the desired branch

checkout in Feature_1, modify a file
checkout in Feature_2 modify a file

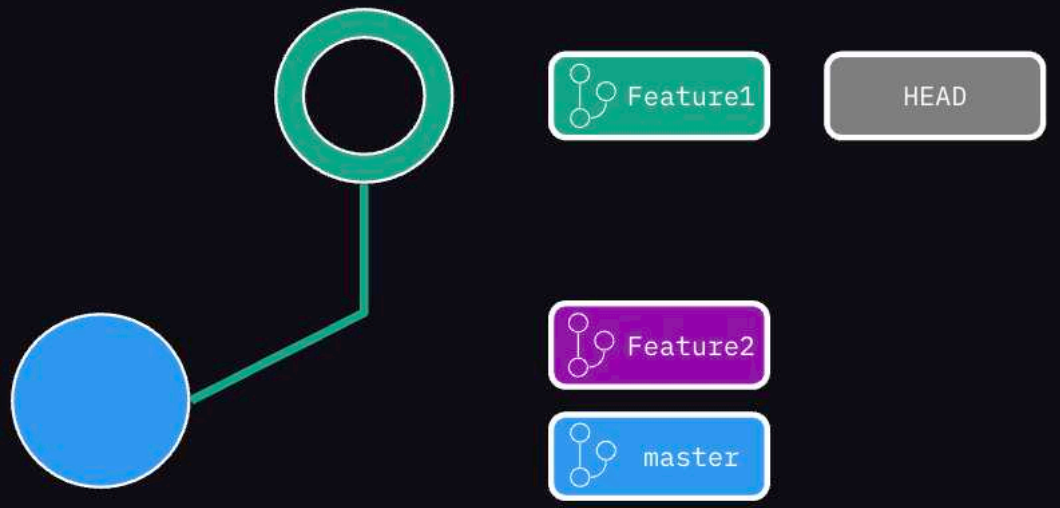
# Creation of Branches



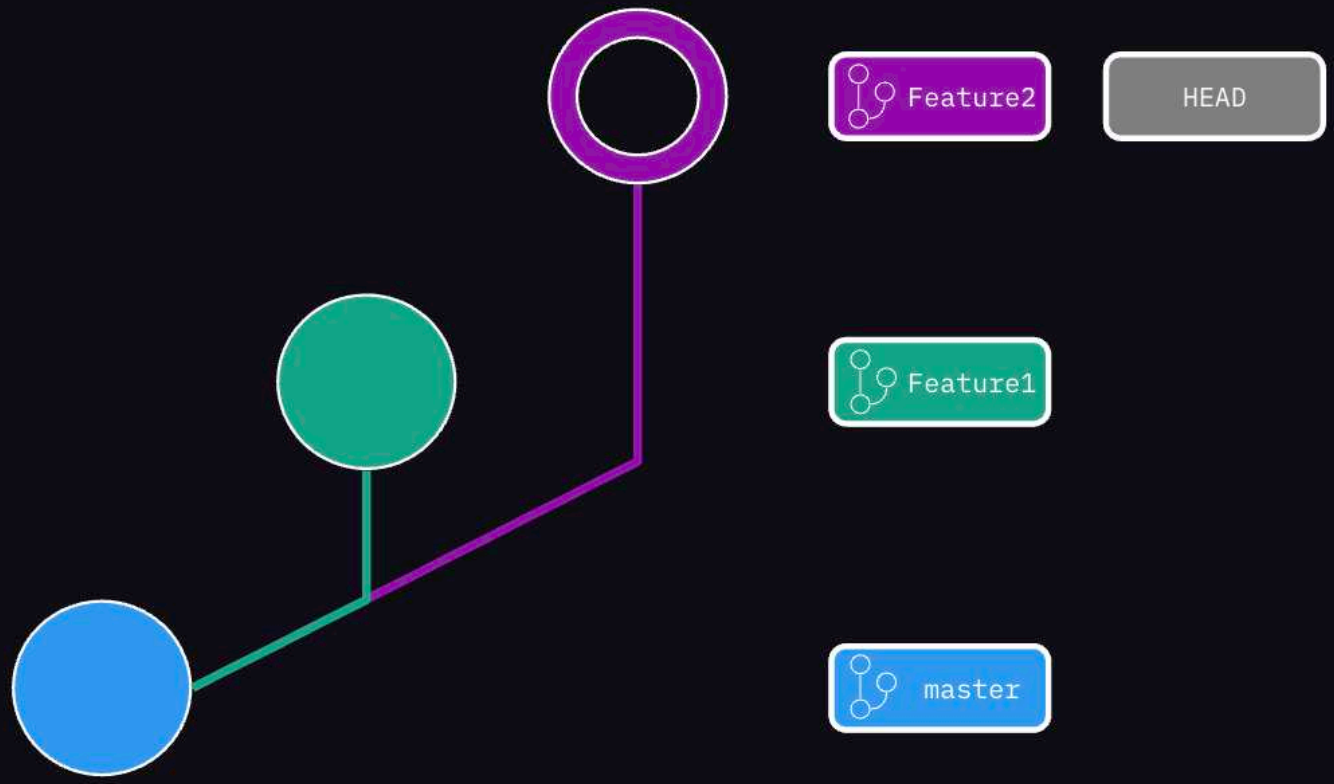
# `git checkout` on Feature_1



# Commit on Feature_1



`git checkout` on Feature_2 and commit

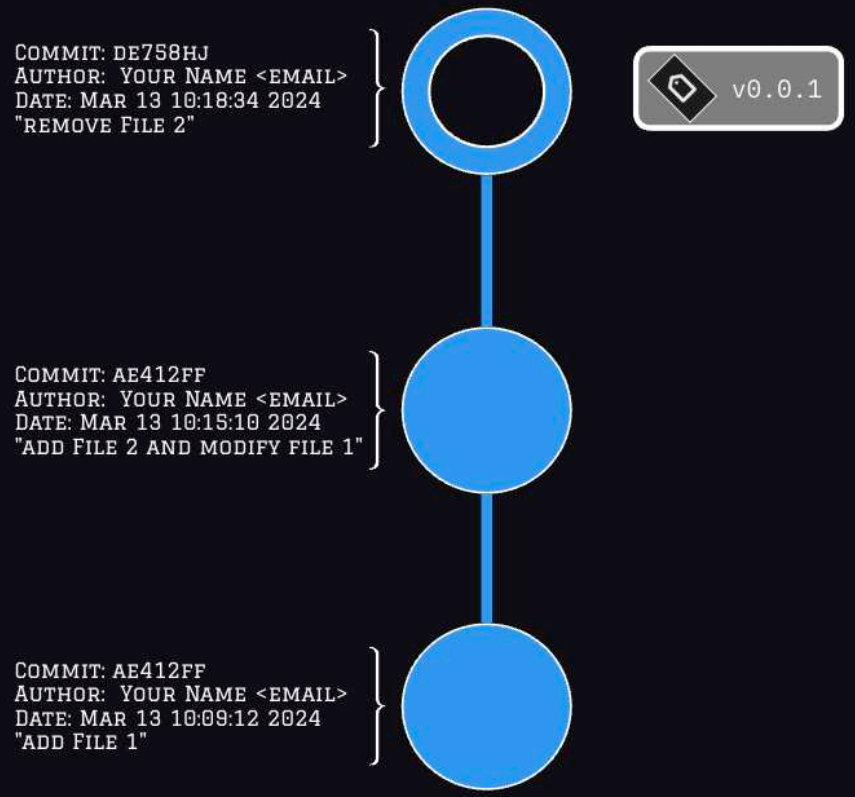


# Tags

## What are tags ? Why do we use them ?

- Label used to mark a specific commit. Denote significant points in a project's history, such as release points (e.g., v1.0, v2.0) or important milestones.
- Tags are immutable, meaning once created, they cannot be changed.
- References to specific commits in the repository, allowing users to easily identify and access important points in the project's timeline.
- A tag is not a branch

# Running `git tag v0.0.0`



# Merging

- Concept : Combining the changes of a branch into another, thus integrating the commits from branch A (the source) to the branch B (the target)
- ***git merge <branch_name>***

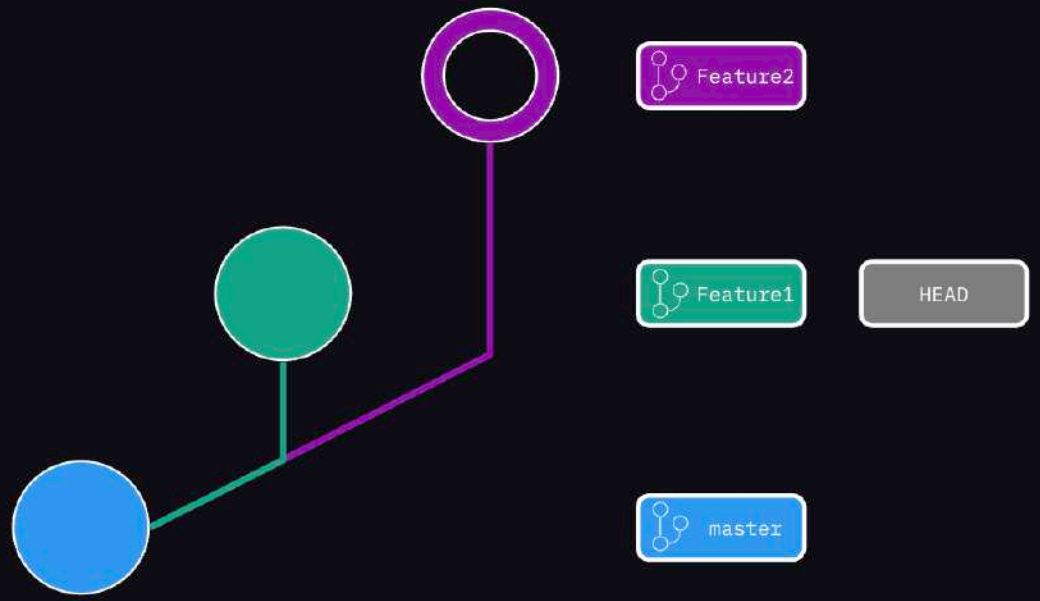
**Fast Forward Merge** :

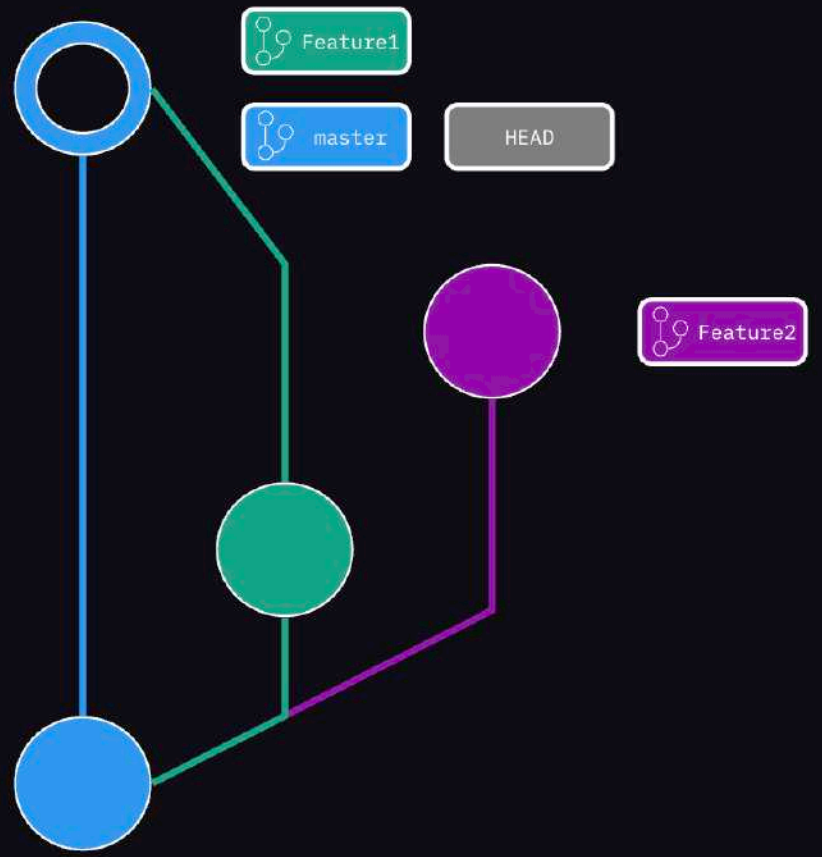
- Happens when there is a direct path between the target and the source.
- Git will move the target branch to the source branch even if there is more than 1 commit

Delete a branch after you're done merging with : ***git branch -d <branch_name>***

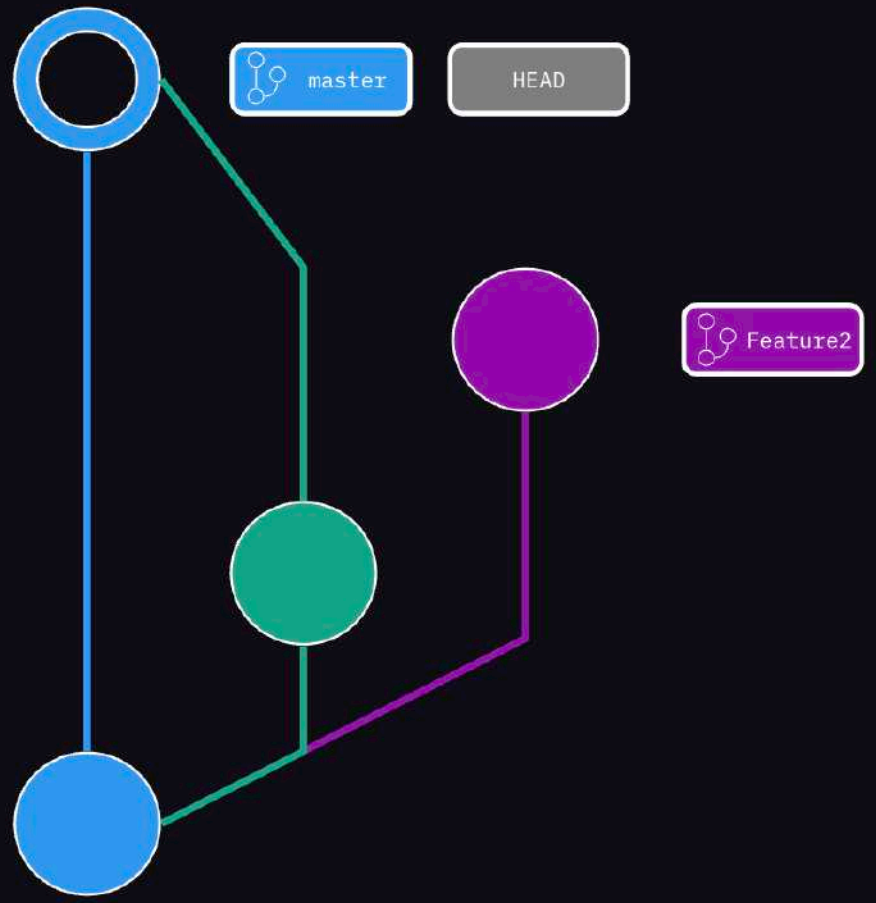
## Fast Forward Merge :

```
git checkout master
git merge Feature1
```



```
git branch -d Feature_1
```



# Merging

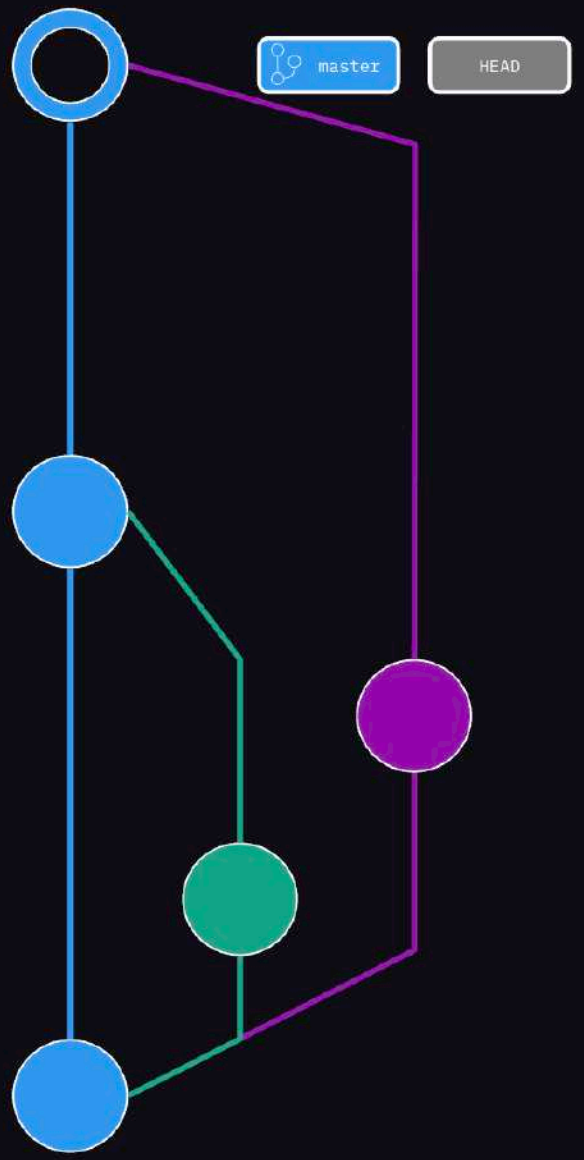
**3-Way merge** :

- No direct path between the source and the target
- Need to do a merge commit

3-Way merge :

```
git checkout master
```

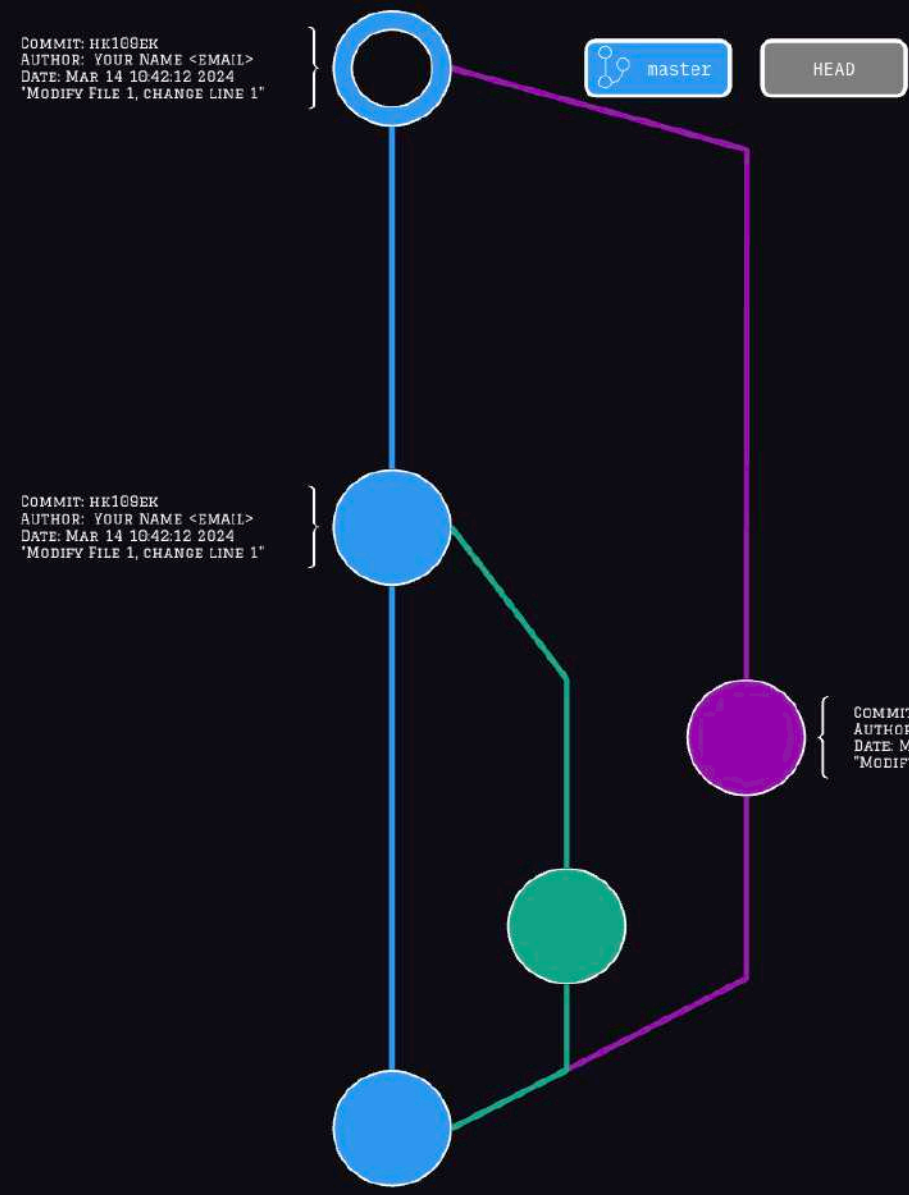
```
git merge Feature2
```



## Merge conflict

- Same file is modified on different branches
- Happens when you try to merge those branches
- You can `vi <file>` to handle the merge conflict or use an IDE like VsCode
- You can then `git add` and `git commit`

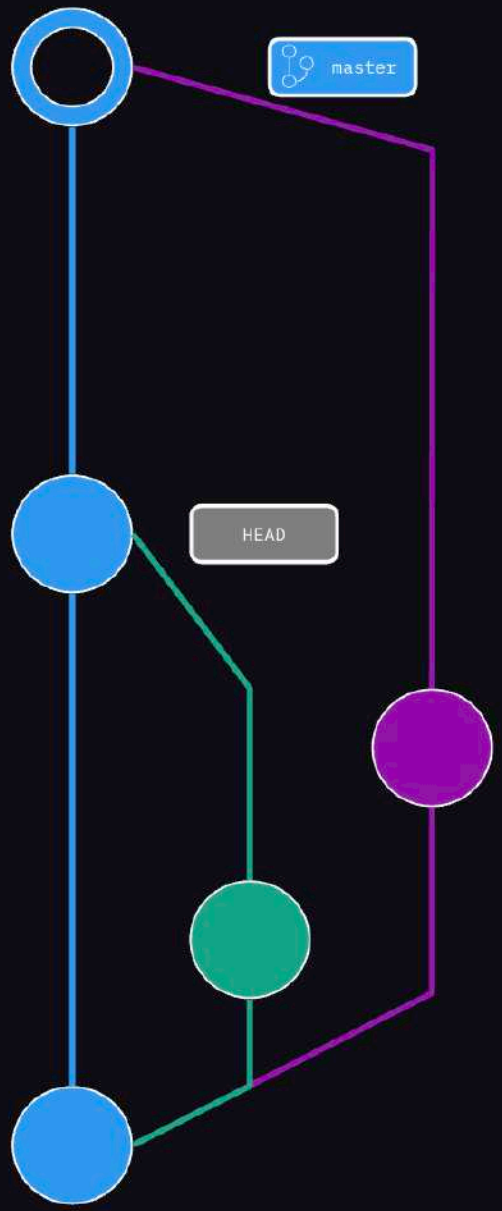
# Merge conflict



## Detached HEAD

- If pointing directly to a commit -> Detached HEAD state
- Any new commit from this state will be on any branch
- Switching to another branch might result in losing track of your work unless you create a new branch or tag to reference it

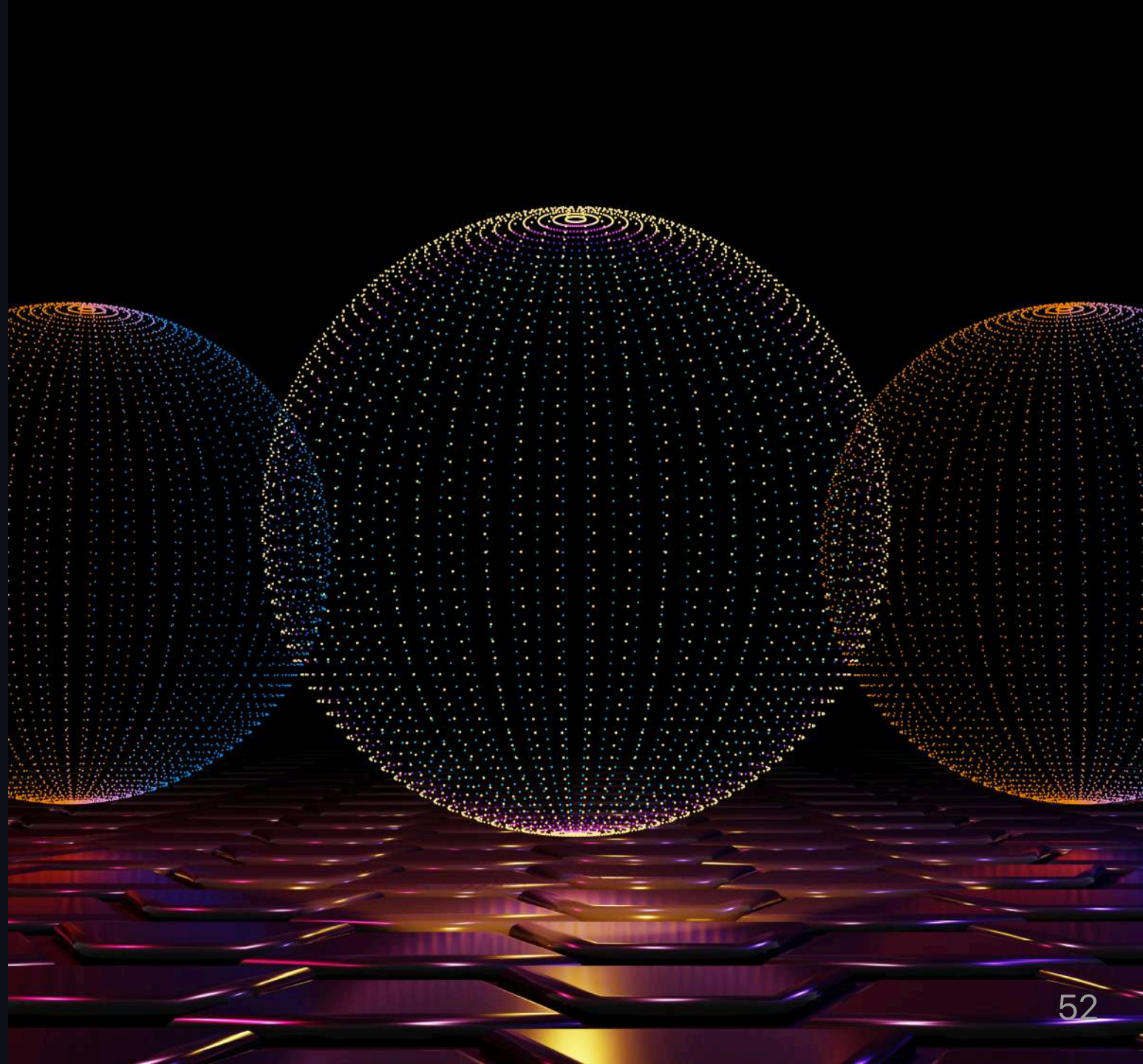
# HEAD detached



# Summary

| Command | Description |
| --- | --- |
| `git branch` | List branches (current branch marked with *) |
| `git branch <branch-name>` | Create a new branch |
| `git branch -d <branch-name>` | Delete the branch |
| `git checkout <branch-name>` | Switch to a different branch |
| `git checkout -b <branch-name>` | Switch to a new branch |
| `git merge <branch>` | Merge specified branch into current branch |
| `git tag <tag-name>` | Create a tag for the current commit |

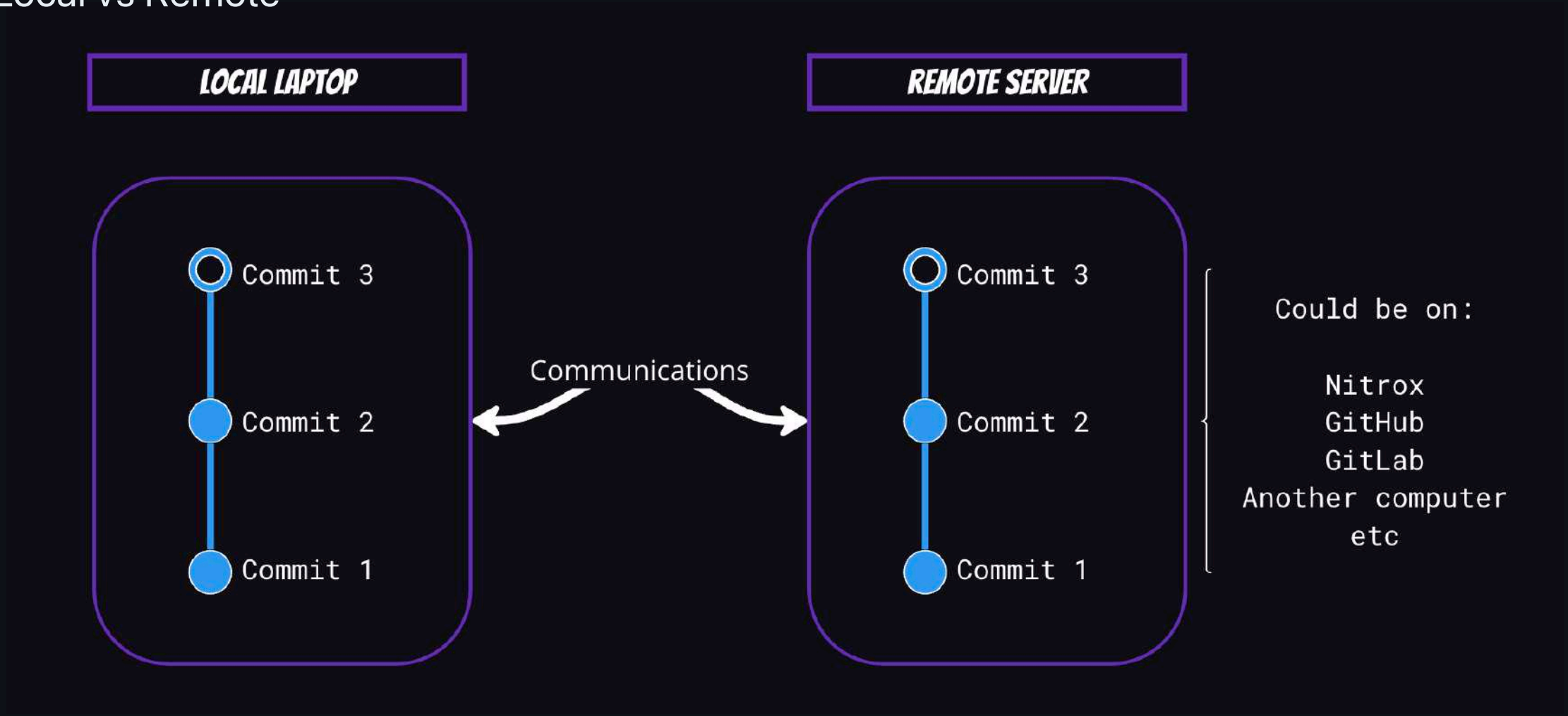
# Remotes



## What is a remote repository ?

- A repository in another location (could be on GitHub)
- You can download changes that are made to the remote repository
- You can upload changes you make in your local repository
- Can be connected to multiple remotes, could also be a coworker's laptop

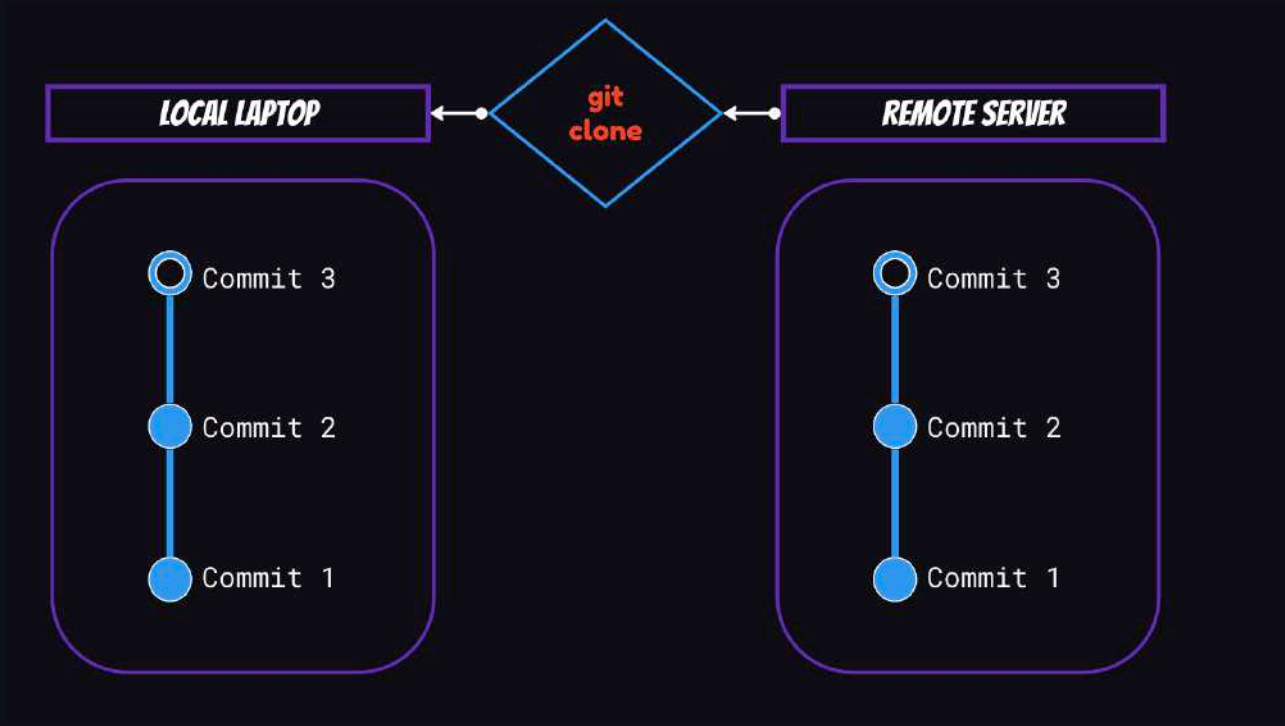
# Local vs Remote



# Getting a repository from a remote

`git clone <location-of-repository>`

Will clone the repository on your local machine



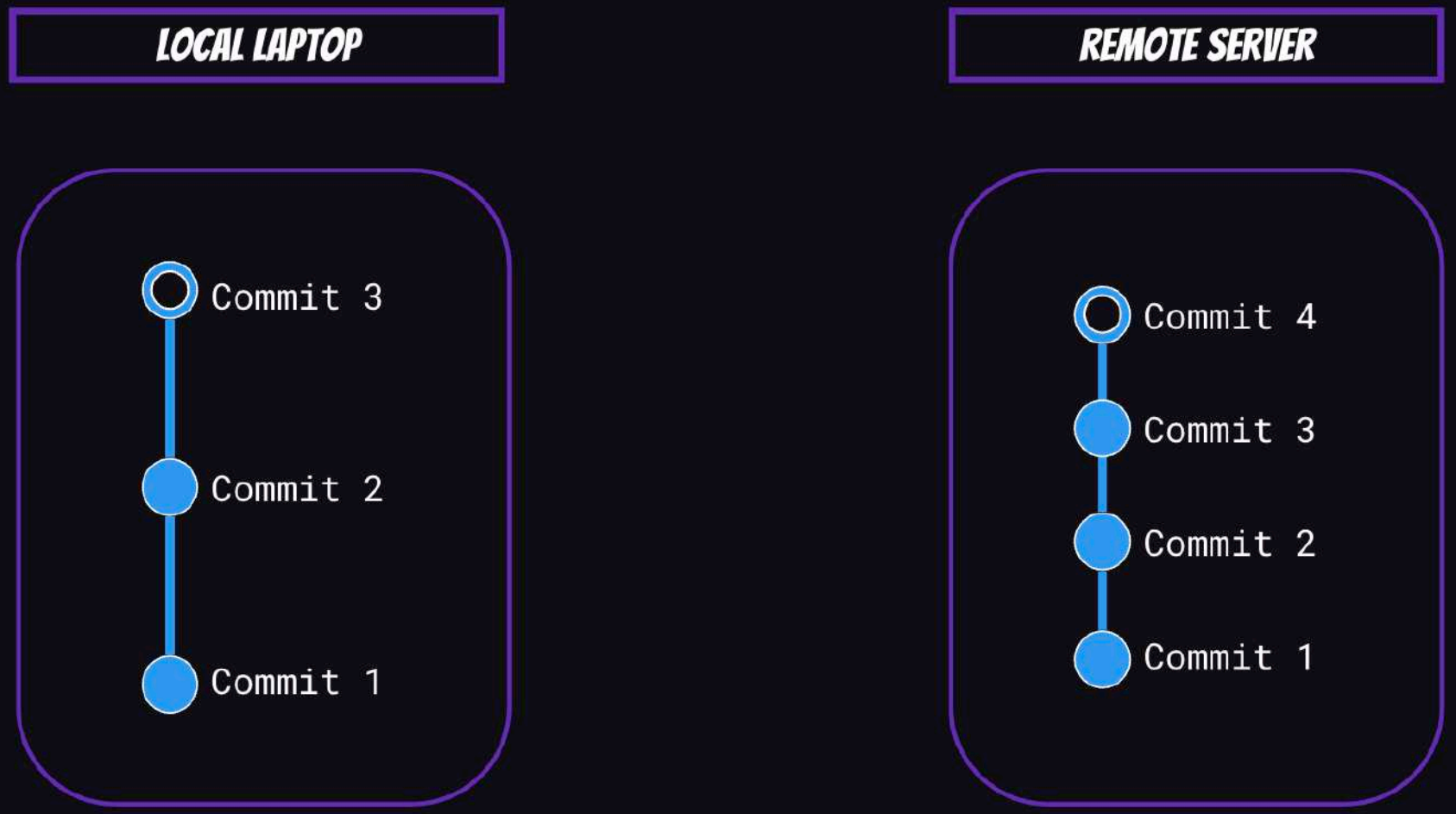
## How to get the new change ?

Two different ways to retrieve what has been done remotely

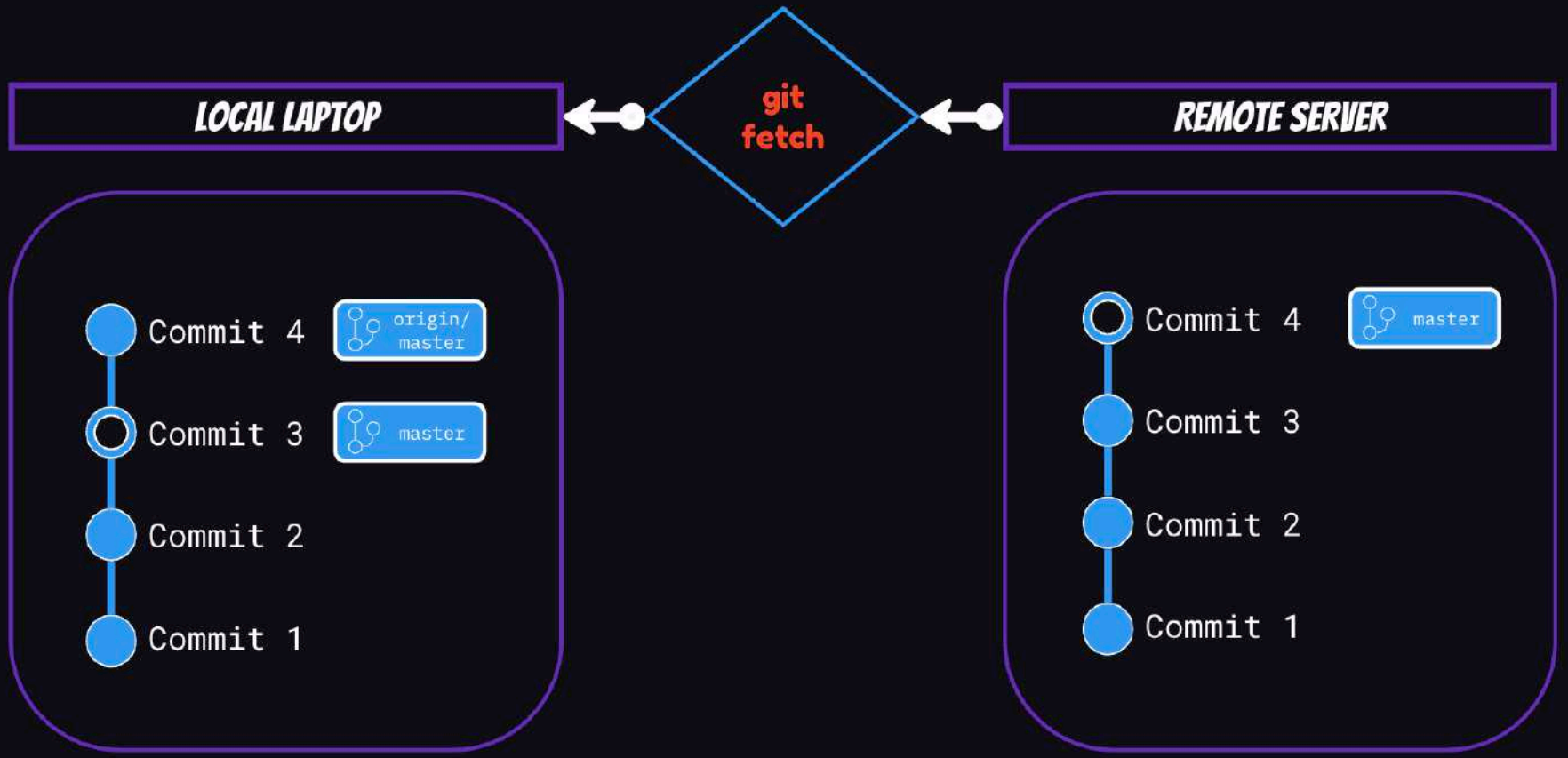
- `git fetch` will retrieve the changes without merging them to the current branch. You then have to `git merge` with your current working directory to absorbs those changes.
- `git pull` will retrieve the changes and automatically merges in the current branch, conflicts will happen if your working directory isn't clean

***Note: Run `git status` to know if you can safely `git pull`***

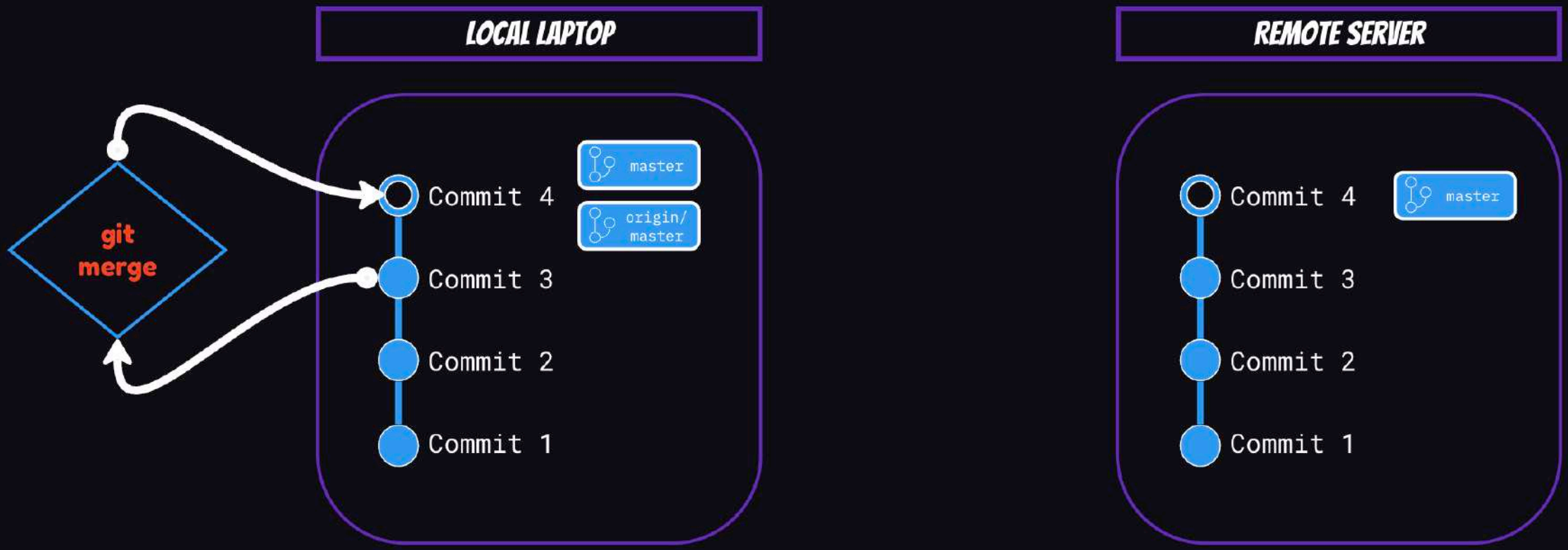
# Change in Remote



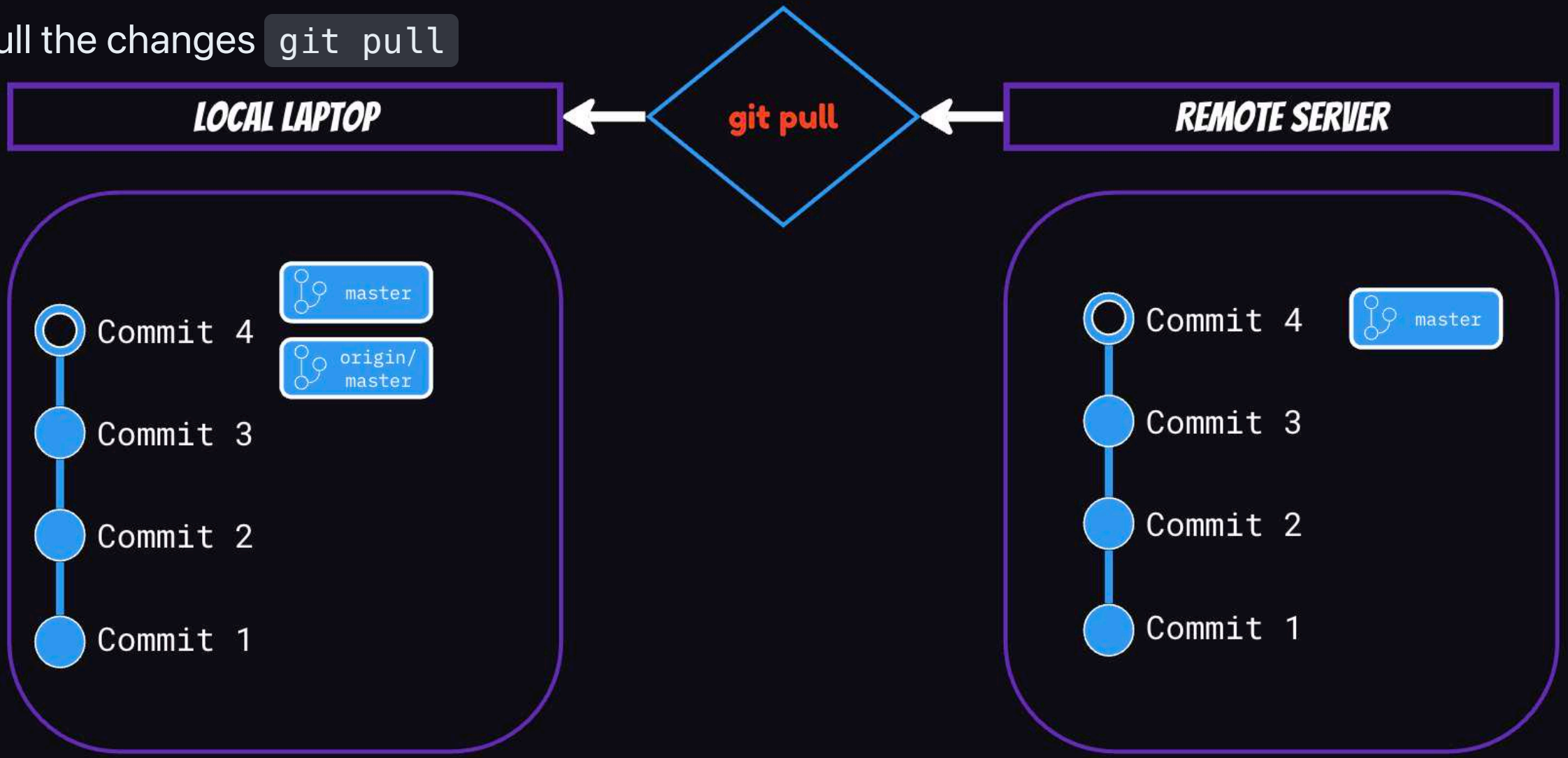
Fetch changes from remote : `git fetch`



## Merge fetched changes : `git merge`



Pull the changes `git pull`

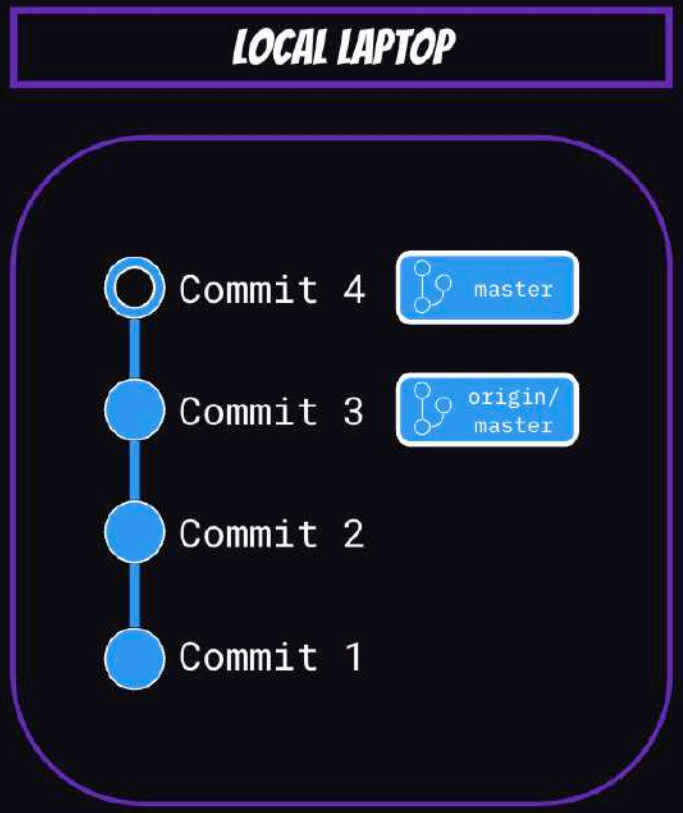


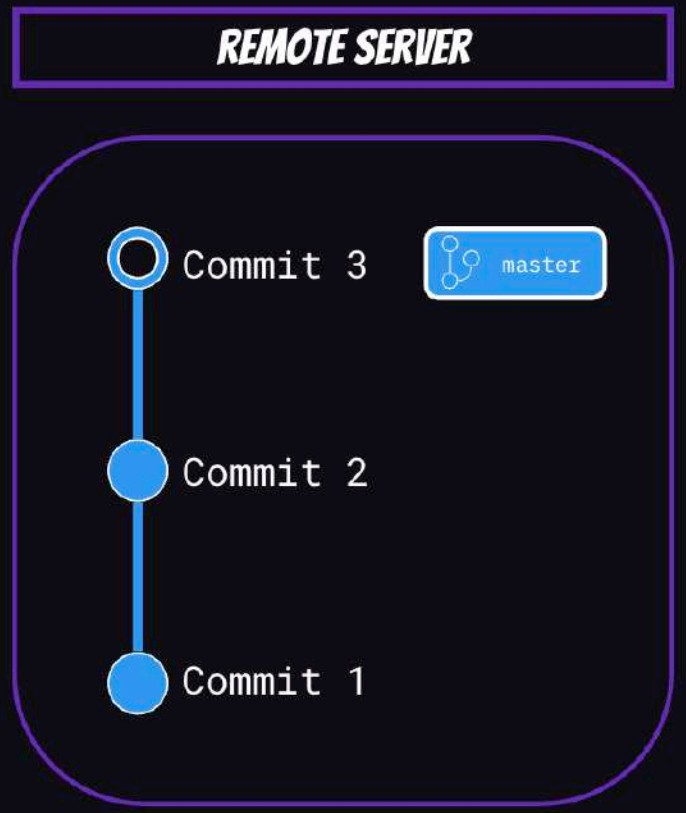
## How to send our news changes ?

Now we made some changes in our local repository :

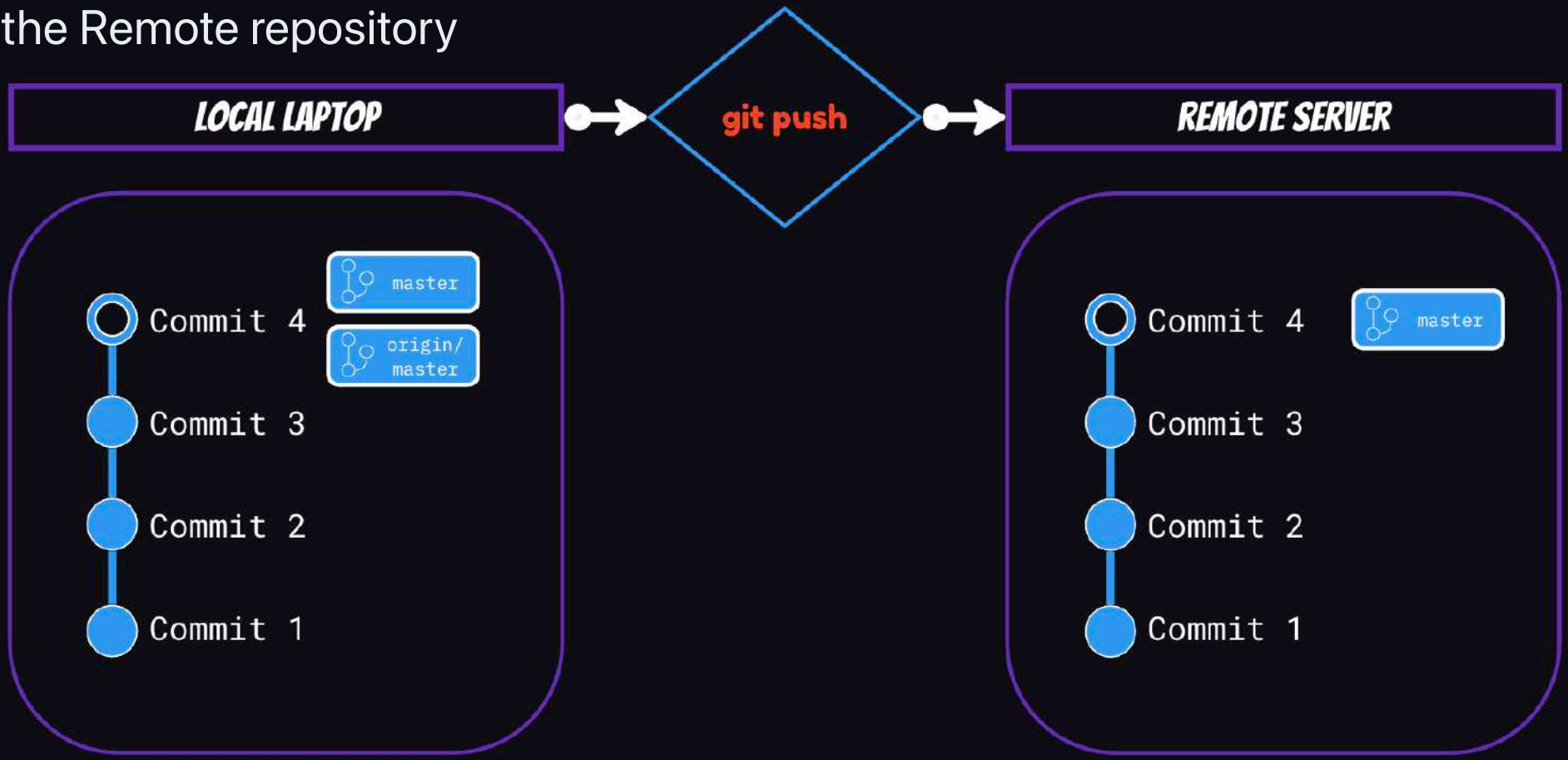
- `git add .` then `git commit -m "New changes"`
- `git push` will upload the changes to the remote repository

# Changes in local repository



# Update the Remote repository

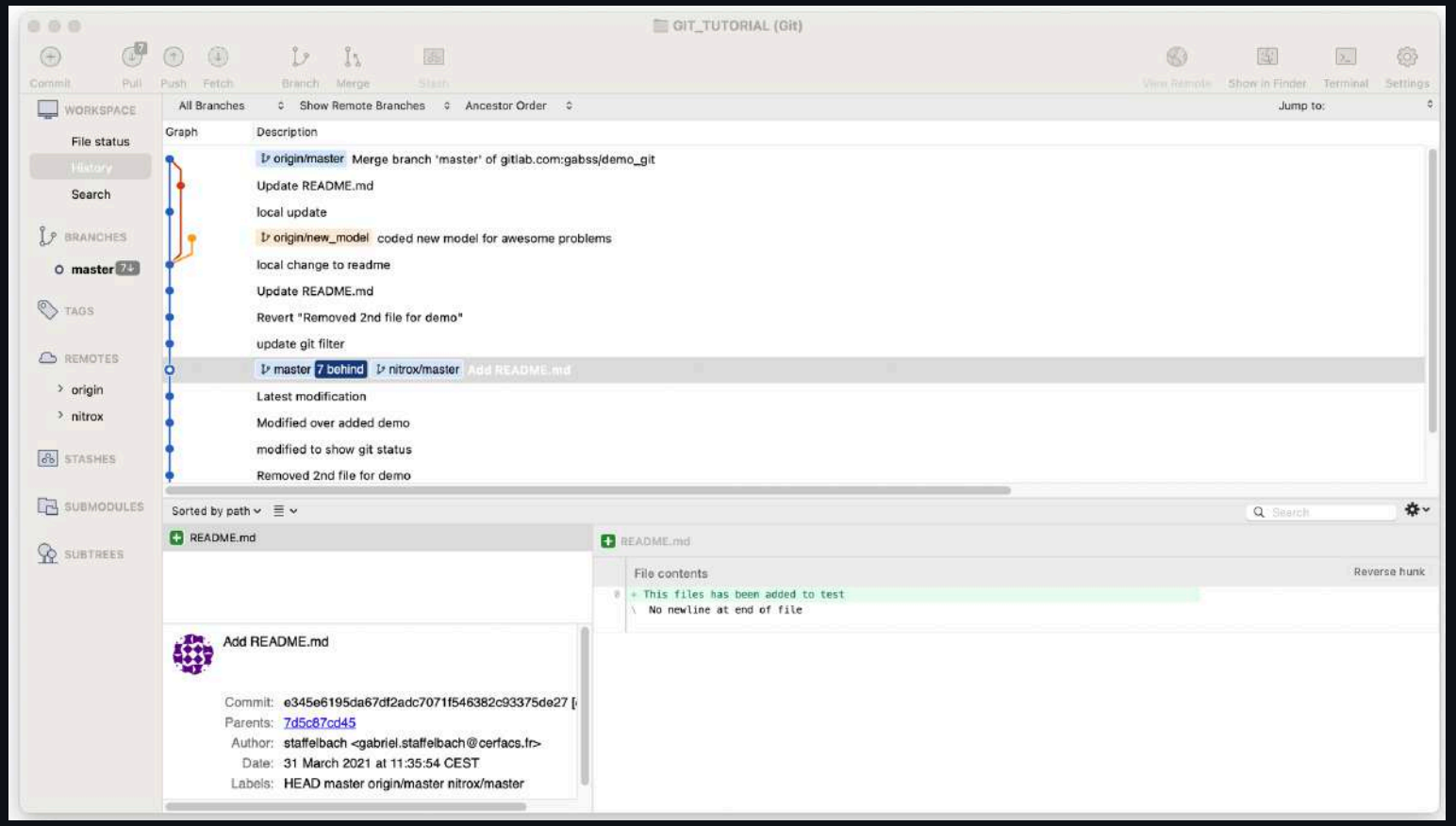


# Summary

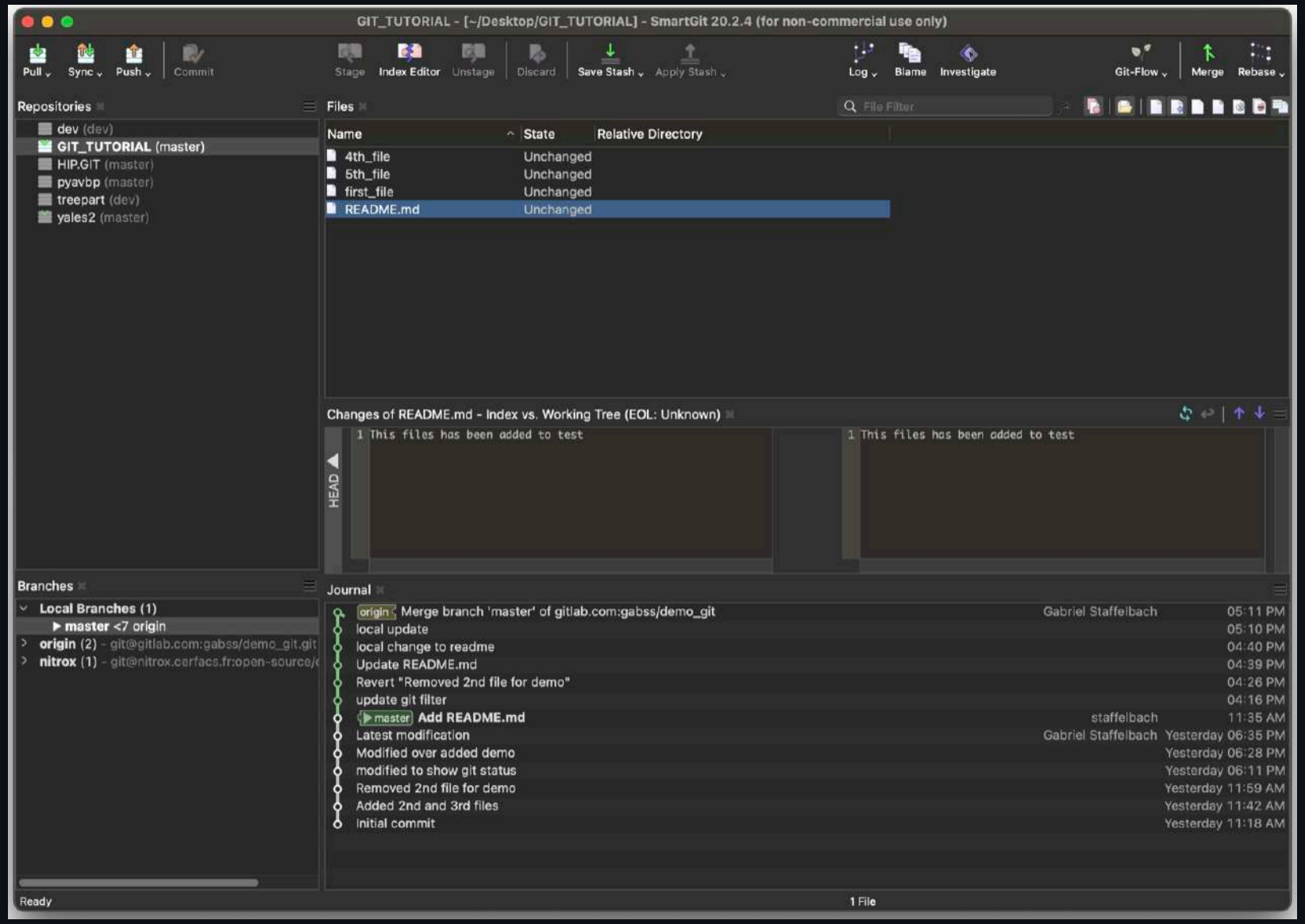
| Command | Description |
| --- | --- |
| `git clone <repository-url>` | Clone an existing repository |
| `git pull` | Fetch from and integrate with another repository |
| `git push` | Update remote repository with local changes |
| `git remote add <remote-name> <url>` | Add a new remote repository |
| `git remote -v` | List remote repositories and their URLs |
| `git fetch <remote-name>` | Fetch changes from a remote repository |
| `git push --tags` | Push tags to the remote repository |

# External tools

## Source tree



# Smartgit


GIT_TUTORIAL - [~/Desktop/GIT_TUTORIAL] - SmartGit 20.2.4 (for non-commercial use only)
Pull
Sync
Push
Commit
Stage
Index Editor
Unstage
Discard
Save Stash
Apply Stash
Log
Blame
Investigate
Git-Flow
Merge
Rebase
Repositories
dev (dev)
GIT_TUTORIAL (master)
HIP.GIT (master)
pyavbp (master)
treepart (dev)
yales2 (master)
Files
File Filter
Name
State
Relative Directory
4th_file Unchanged
5th_file Unchanged
first_file Unchanged
README.md Unchanged
Changes of README.md - Index vs. Working Tree (EOL: Unknown)
HEAD
1 This files has been added to test
1 This files has been added to test
Branches
Local Branches (1)
master <7 origin
origin (2) - git@gitlab.com:gabss/demo_git.git
nitrox (1) - git@nitrox.cerfacs.fr:open-source/
Journal
origin Merge branch 'master' of gitlab.com:gabss/demo_git Gabriel Staffelbach 05:11 PM
local update 05:10 PM
local change to readme 04:40 PM
Update README.md 04:39 PM
Revert "Removed 2nd file for demo" 04:26 PM
update git filter 04:16 PM
master Add README.md staffelbach 11:35 AM
Latest modification Gabriel Staffelbach Yesterday 06:35 PM
Modified over added demo Yesterday 06:28 PM
modified to show git status Yesterday 06:11 PM
Removed 2nd file for demo Yesterday 11:59 AM
Added 2nd and 3rd files Yesterday 11:42 AM
initial commit Yesterday 11:18 AM
Ready
1 File

## Best practices

- Create local branch
- Do your work and commit locally
- Push local branch to remote
- Merge master into current branch

## SOURCES

- https://git-scm.com
- https://www.youtube.com/@DavidMahler
- https://marklodato.github.io/visual-git-guide/index-en.html