VCS-R

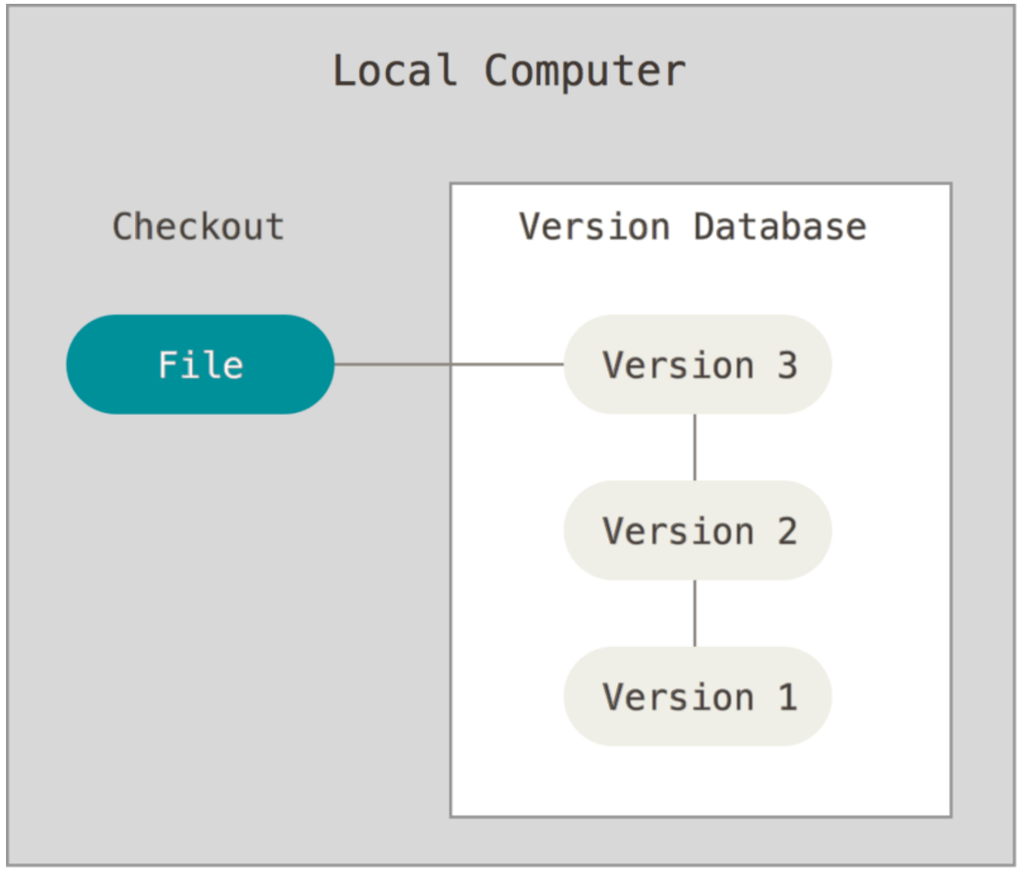
Version Control System in Rust

Lawson Oliveira Lima
Lucas Vitoriano De Queiroz Lira

CONTENT

WHAT IS ?

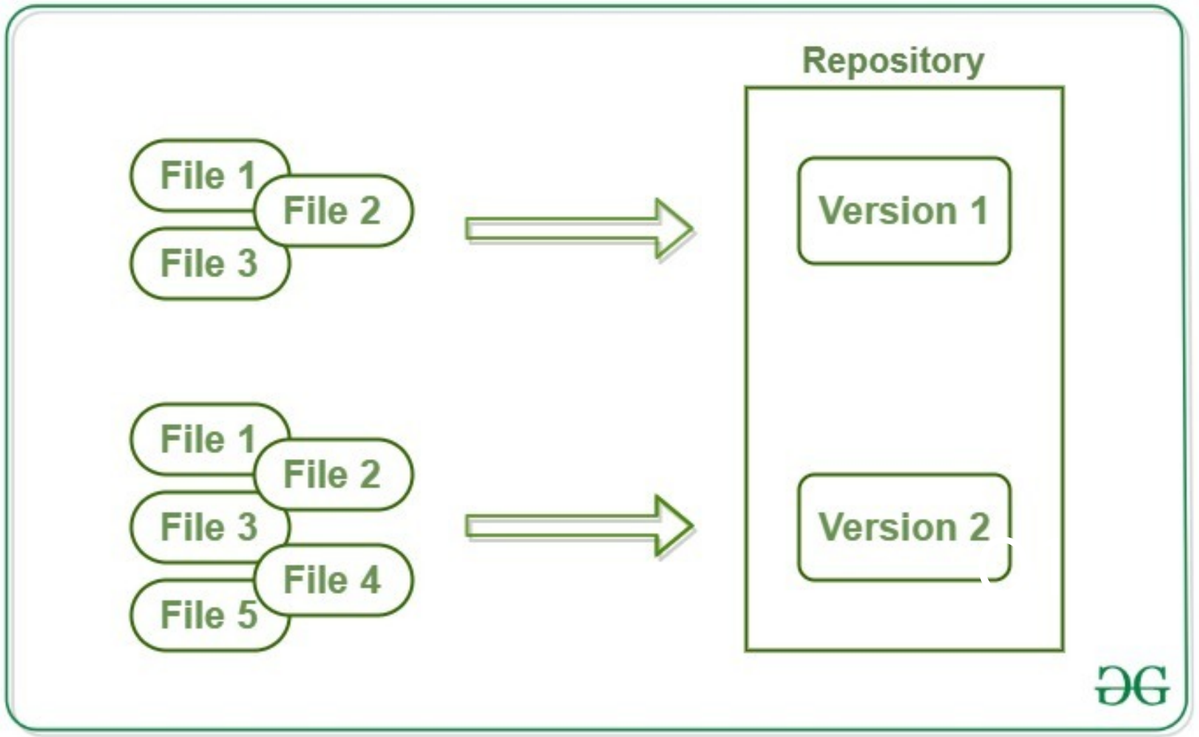

git


Local Computer
Checkout
Version Database
File
Version 3
Version 2
Version 1

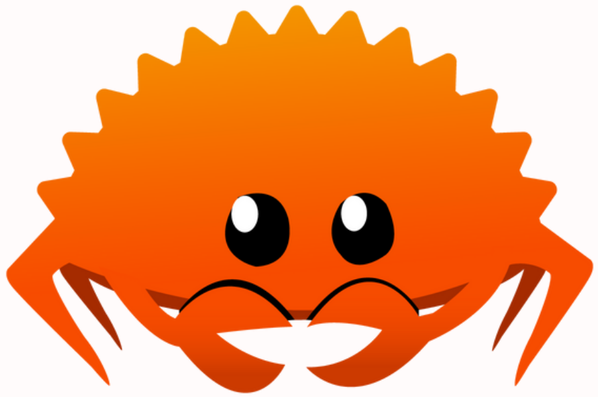
OBJECTIVE

SYSTEM CAPABLE OF GERENCIATE A REPOSITORY

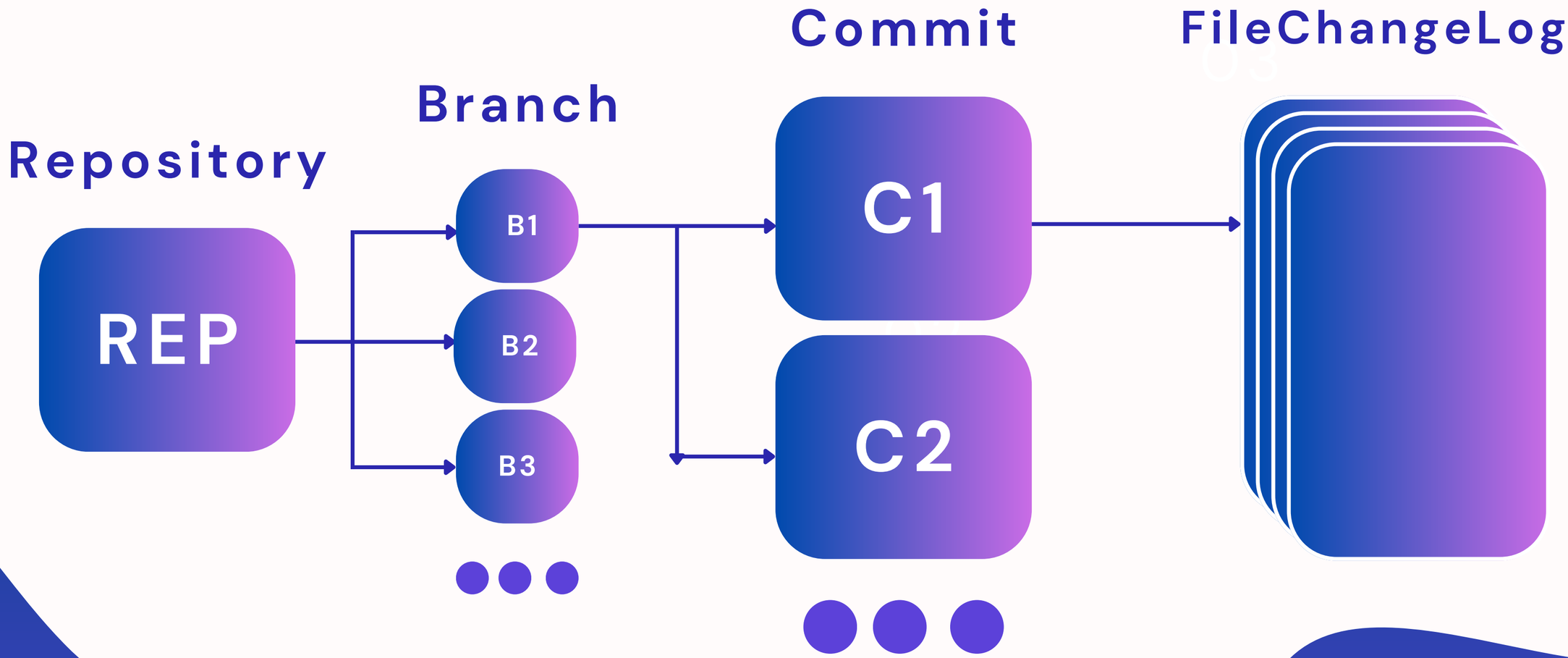
Create a repository
Save a version of the file (commit)
Delete commit
Change the actual Version
Manage Branches



WHY RUST ?



STRUCT



STRUCTS

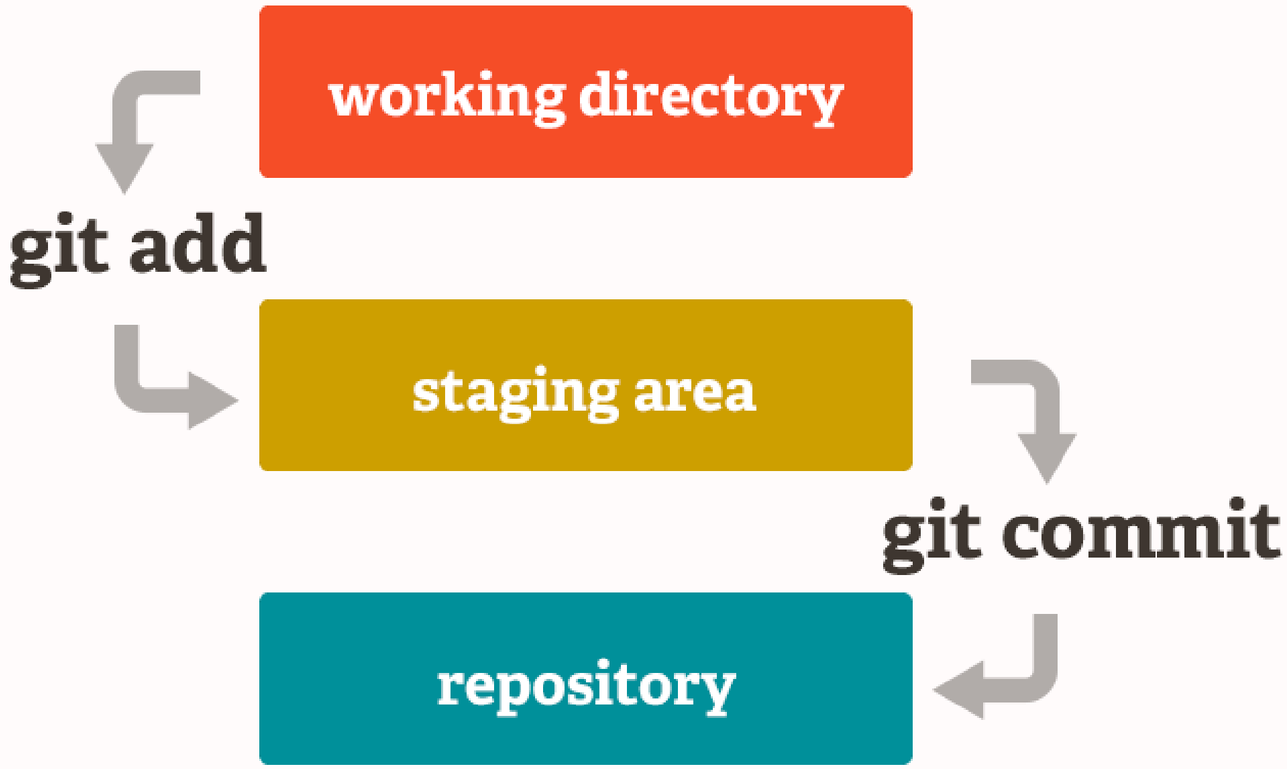
```rust
#[derive(Debug, Serialize, Deserialize)]
3 implementations
pub struct Branch {
    pub branch_name: String,
    pub head_commit_hash: String,
    pub commits: Vec<Commit>,
}

#[derive(Serialize, Deserialize)]
2 implementations
pub struct Repository {
    pub current_branch: String,
    pub branches: Vec<Branch>,
}
```

```rust
#[derive(Debug, Clone, Serialize, Deserialize)]
4 implementations
pub struct FileChangeLog {
    pub original_file_path: String,
    pub original_file: String,
    pub last_file: String,
    pub last_file_path: String,
    pub hash_changelog: String,
    pub hash_files_path: String,
    pub version: u32,
    pub parent_version: u32,
}

#[derive(Debug, Clone, Serialize, Deserialize)]
4 implementations
pub struct Commit {
    pub files_changelogs: Vec<FileChangeLog>,
    pub commit_hash: String,
    pub parent_commits: Vec<String>,
}
```

ADD / REMOVE


working directory
git add
staging area
git commit
repository

HOW IT WORKS ?

Add

Write the file name and file hash in staging_area.yml

↓

Copy the file to add_contents

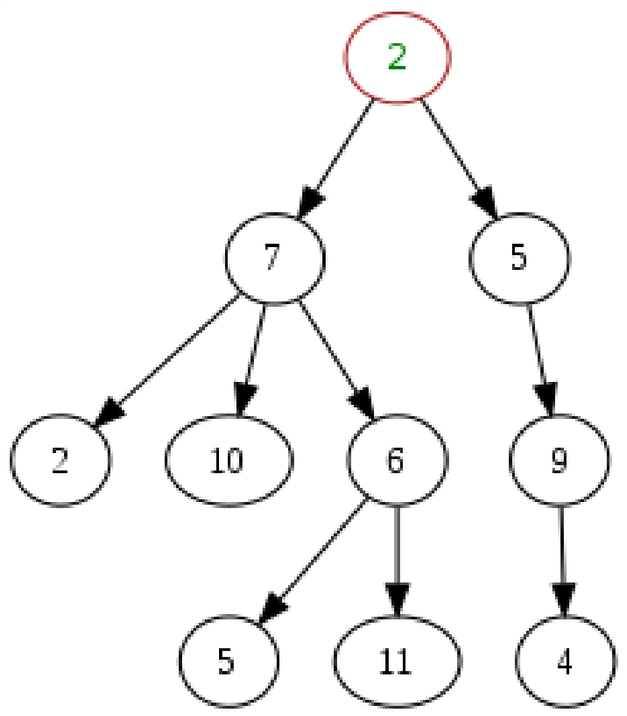
Remove

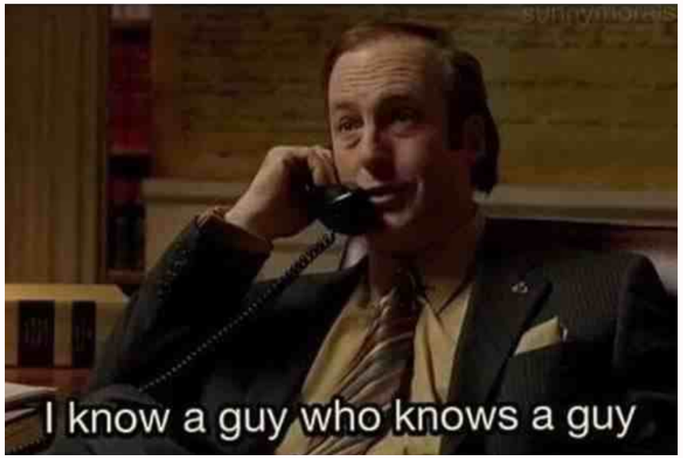
Erase the file name and file hash in staging_area.yml

↓

Delete the file from add_contents

COMMIT





HOW IT WORKS ?

Commit

Hash of commit header

↓

Build commit tree

↓

Build FileChangeLog tree for each file in the staging area

Build a version tree for each file in the staging area

↓

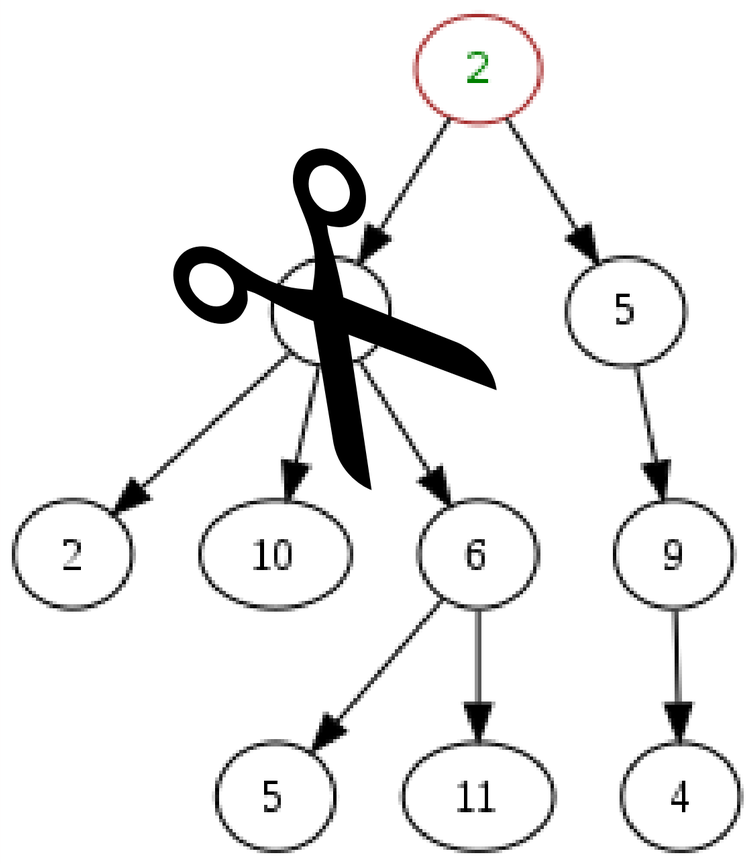
Apply each patch and finally create the new diff file

↓

Save the diff in .saves using the hash file as name

Remove files from staging area

DELETE COMMIT



HOW IT WORKS ?

Delete commit

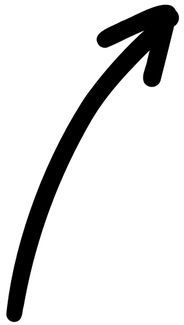
Hash of the commit to delete

↓

Build commit tree

↓

Delete the patchs that are after the desired commit

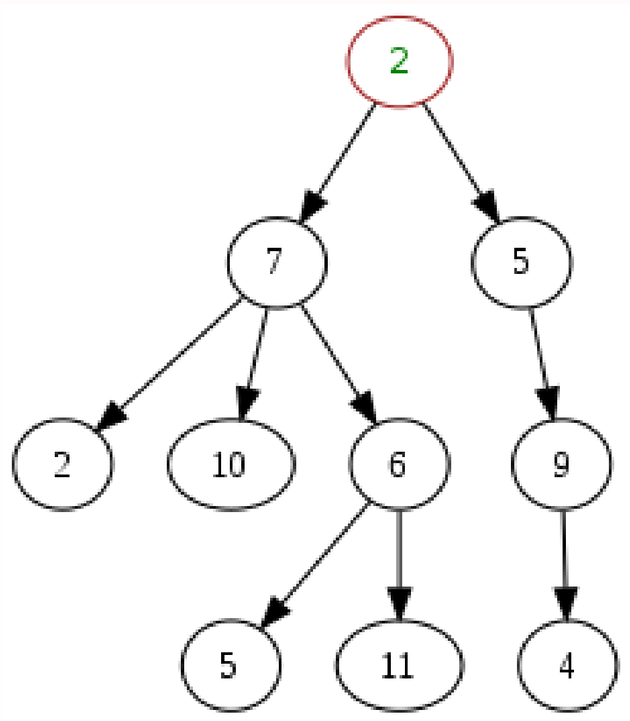


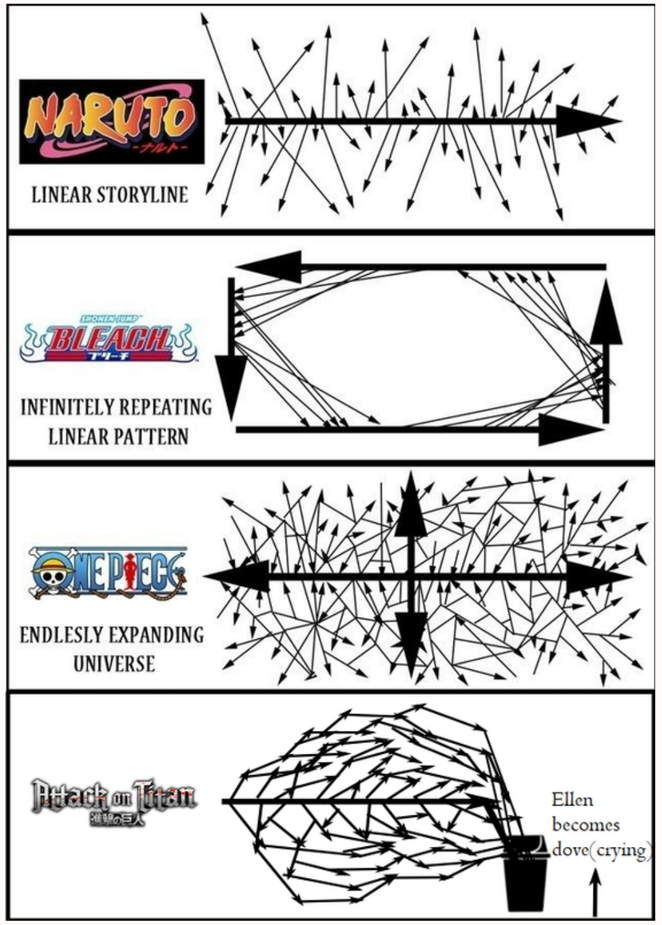
Replace the current commit tree

↓

Update the struct file

CHANGE VERSION





CHANGE VERSION

Commit hash

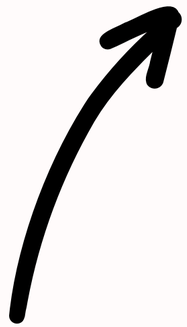
Desired version

↓

Build commit tree until the root version

↓

Build FileChangeLog tree for each file in commit tree



Build a version tree for each file in the commit tree

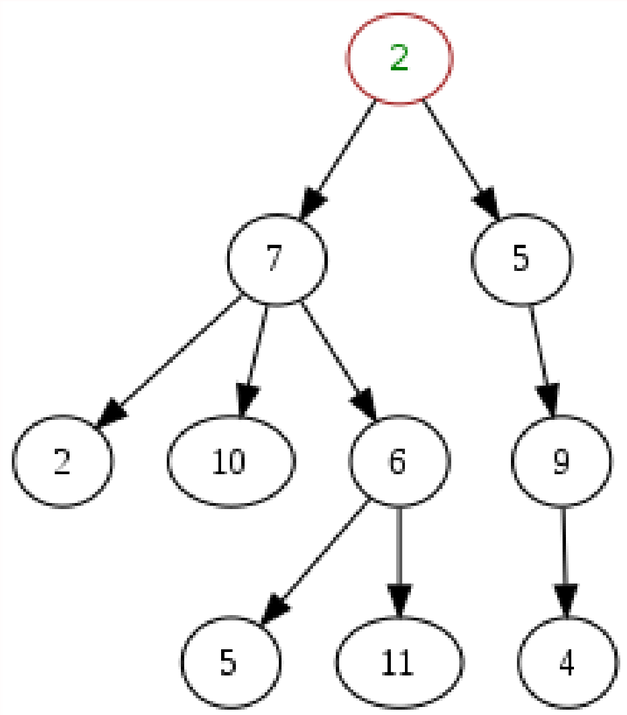
↓

Apply each patch to the correspondent file

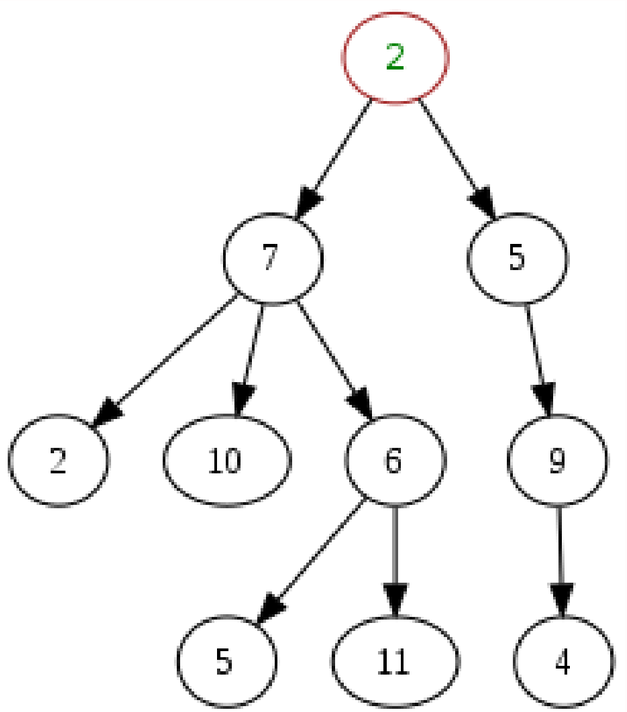
↓

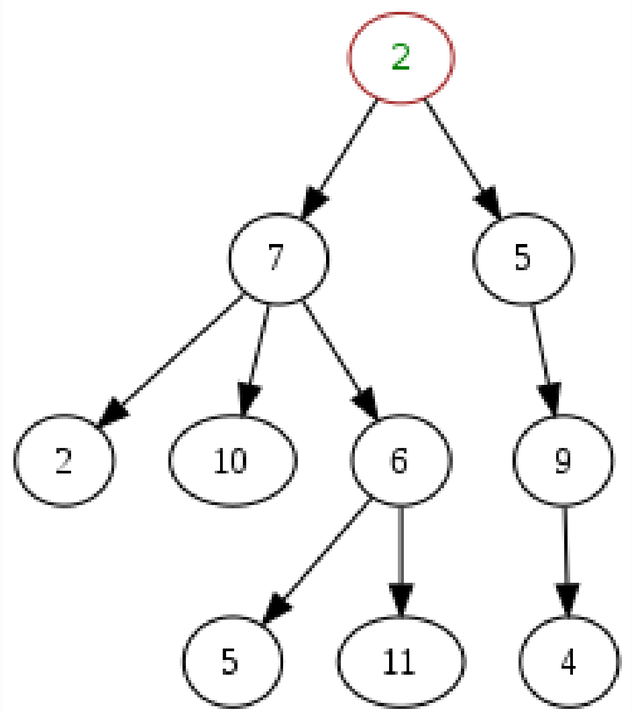
Apply the result for each file

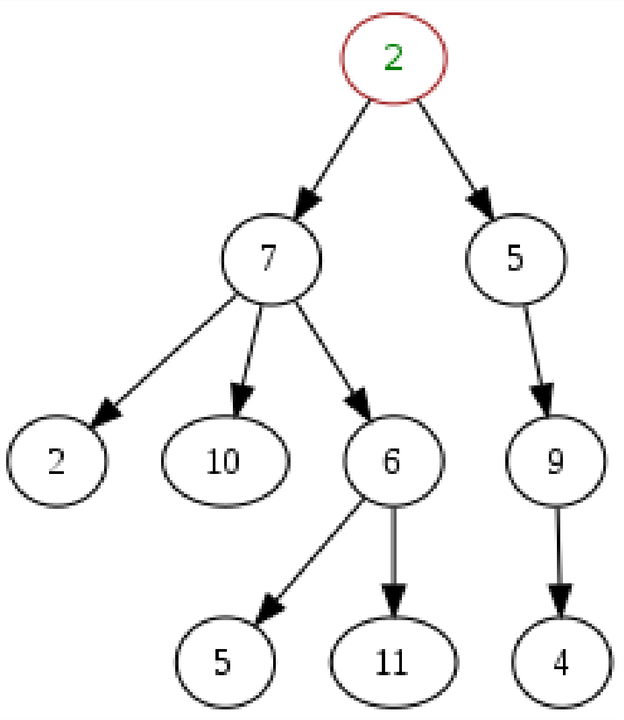
CHANGE VERSION



BRANCH







CREATE BRANCH

```rust
#[derive(Debug, Serialize, Deserialize)]
3 implementations
pub struct Branch {
    pub branch_name: String,
    pub head_commit_hash: String,  // Added: Hash
    pub commits: Vec<Commit>,  // Added: List of
}
```

```rust
#[derive(Serialize, Deserialize)]
2 implementations
pub struct Repository {
    pub current_branch: String,
    pub branches: Vec<Branch>,
}
```

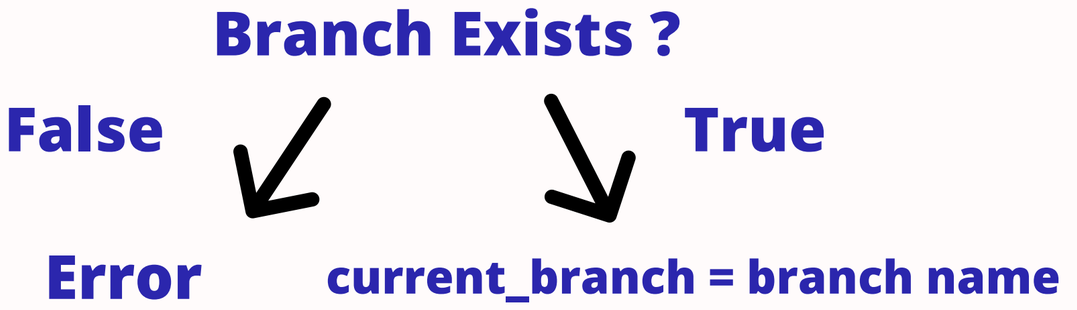
Duplicate the current branch

↓

push in the branches vector

CHANGE BRANCH

```rust
#[derive(Serialize, Deserialize)]
2 implementations
pub struct Repository {
    pub current_branch: String,
    pub branches: Vec<Branch>,
}
```



DELETE BRANCH

Runs over the branch to delete

↓

for each file hash :

files_hashs.push(file_hash)

↓

for file_hash in files_hash:

for each other branch:

if another branch contains file hash

False ↙ ↘ True

Delete file — nothing

↓

delete branch struct from repository

```rust
#[derive(Serialize, Deserialize)]
2 implementations
pub struct Repository {
    pub current_branch: String,
    pub branches: Vec<Branch>,
}
```

Demonstration