GIT Talk

From zero to ~~hero~~ crab

Oriol Agost Batalla - Ferran Aran Domingo

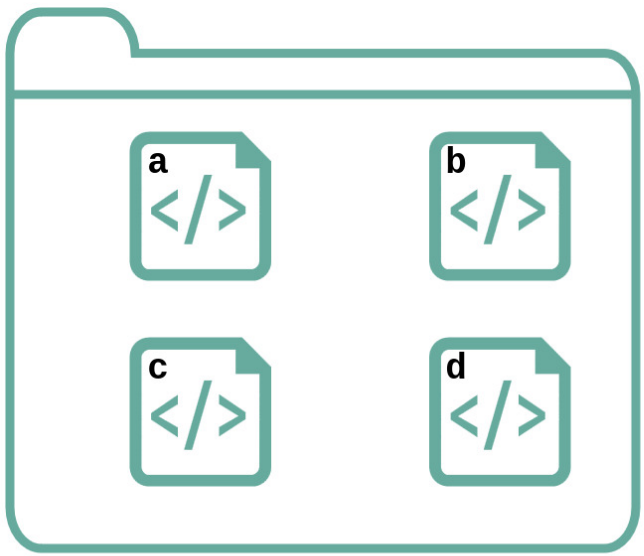
Lleidahack

Friday 8th March, 2024

Index

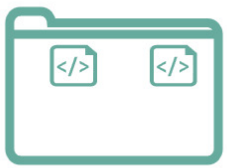
Why Git?

project/

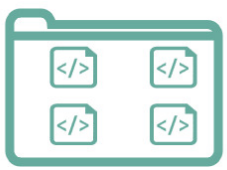


Historial

Canvis del dilluns
[Apartat 1]



Canvis del dimarts
[Apartat 2]



Canvis del dimecres
[Apartat 3]

Why Git?

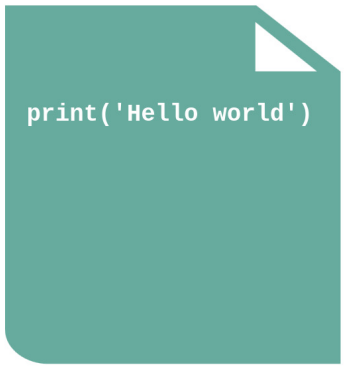
- `git init`: Create a new repository in the current directory
- `git add <file>`: Add a file to the staging area (`.` adds the whole directory)
- `git commit -m "message"`: Commit the changes in the staging area

Why Git?

```
[ferran@arch] ➜ project git init
Initialized empty Git repository in /home/ferran/project/.git/
[ferran@arch] ➜ project ( main) nvim a
[ferran@arch] ➜ project (! main) cat a
print('Hello world')
[ferran@arch] ➜ project (! main) git add a
[ferran@arch] ➜ project (S main) git commit -m "Add file a"
[main (root-commit) b4fb96f] Add file a
 1 file changed, 1 insertion(+)
 create mode 100644 a
[ferran@arch] ➜ project ( main)
```

Basics

project/a



Historial

Canvis del dilluns
[Apartat 1]

C1

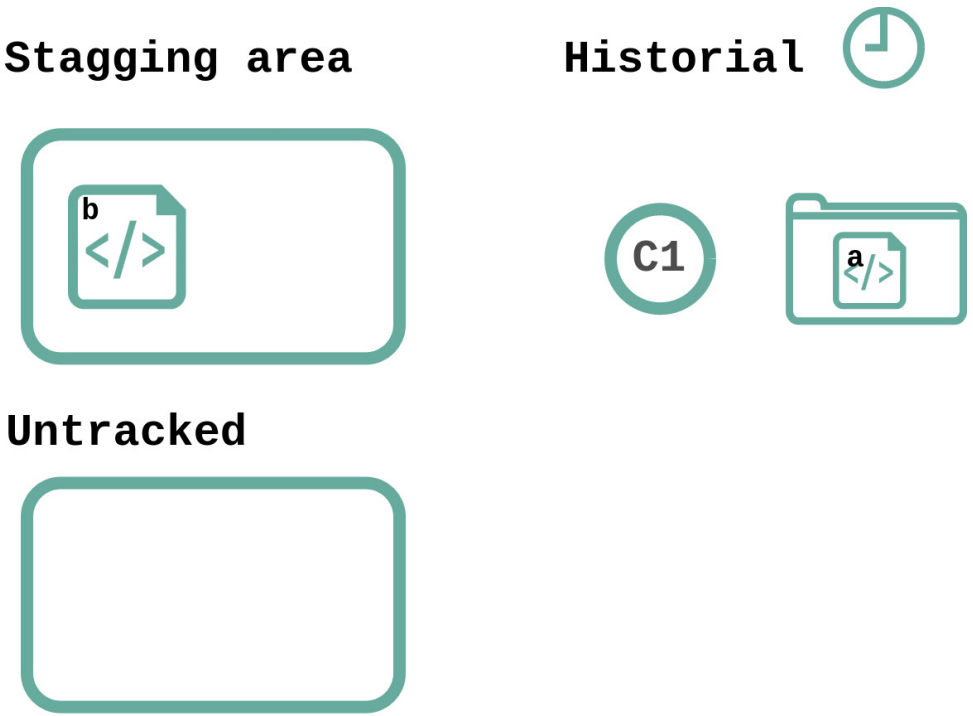
- `git status`: Check the status of the repository
- `git diff`: Show the changes between the working directory and the staging area

Staging area, git diff and status

Staging area, git diff and status

```
[ferran@arch] →project (main) nvim b
[ferran@arch] →project (!main) cat b
print('Bye bye')
[ferran@arch] →project (!main) git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        b

nothing added to commit but untracked files present (use "git add" to track)
[ferran@arch] →project (!main)
```

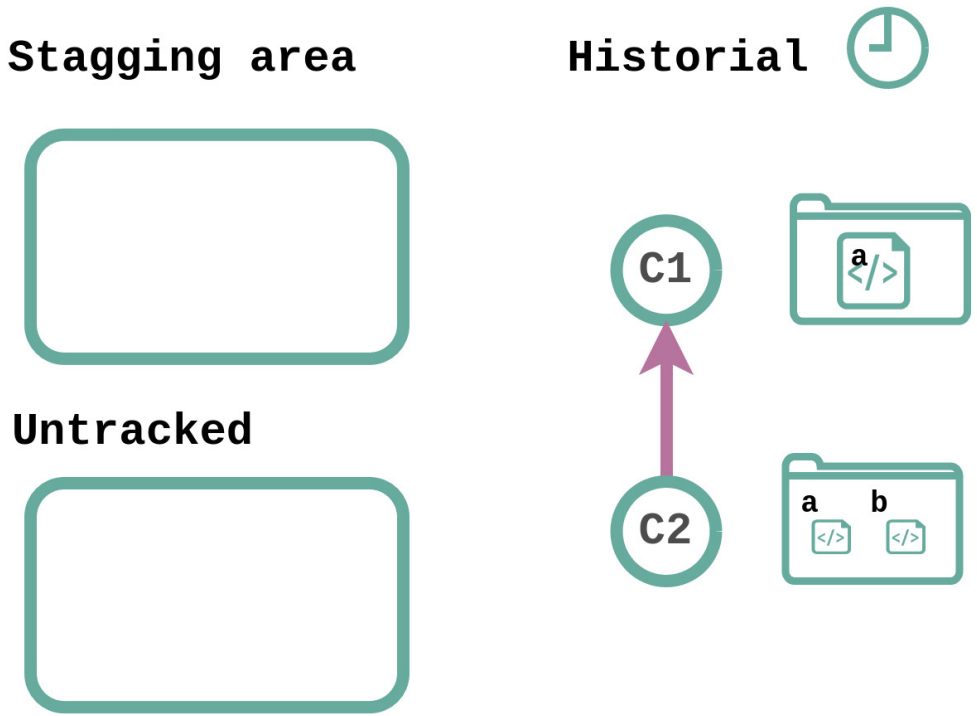
Staging area, git diff and status



Staging area, git diff and status

```
[ferran@arch] ➜ project (!ᚠ main) git add b
[ferran@arch] ➜ project (Sᚠ main) git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   b

[ferran@arch] ➜ project (Sᚠ main)
```

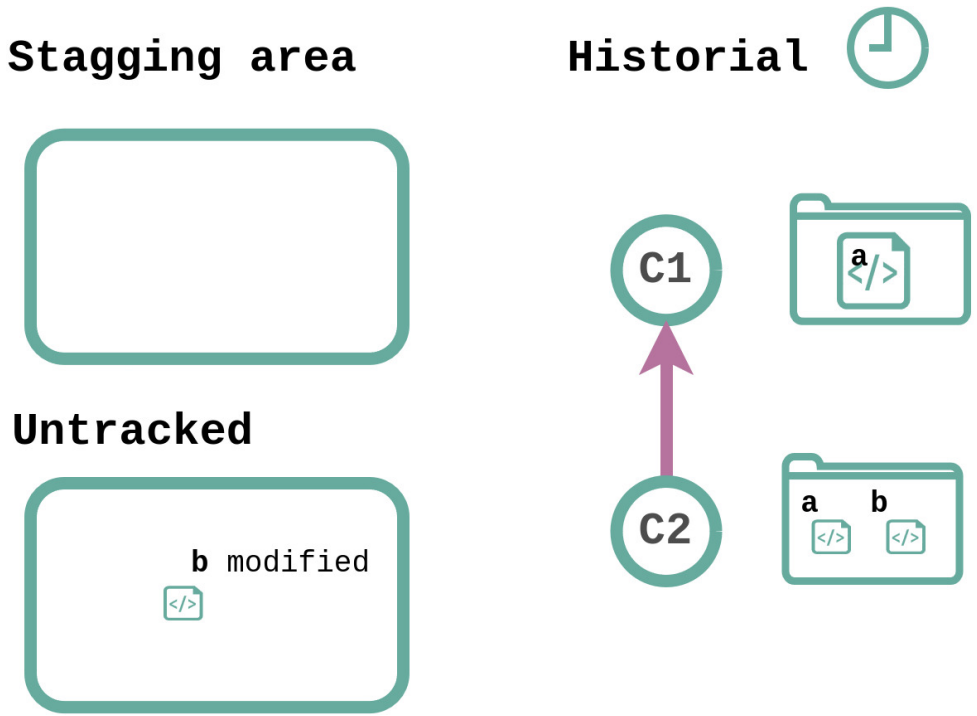
Staging area, git diff and status



Staging area, git diff and status

```
[ferran@arch] → project (S main) git commit -m "Add file b"
[main 0dc0a22] Add file b
 1 file changed, 1 insertion(+)
 create mode 100644 b
[ferran@arch] → project ( main) git status
On branch main
nothing to commit, working tree clean
[ferran@arch] → project ( main)
```

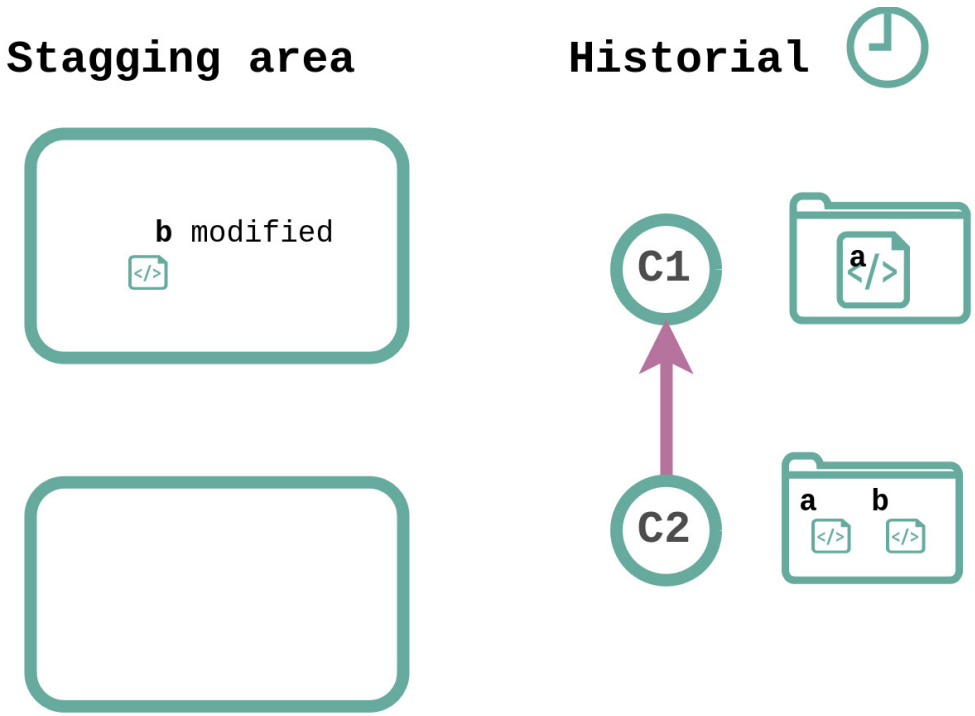
Staging area, git diff and status



Staging area, git diff and status

```
[ferran@arch] → project ( main) ls
 a  b
[ferran@arch] → project ( main) echo "test" > a
[ferran@arch] → project ( main) git status
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   a

no changes added to commit (use "git add" and/or "git commit -a")
[ferran@arch] → project ( main) git diff
diff --git a/a b/a
index f7d1785..9daeafb 100644
--- a/a
+++ b/a
@@ -1 +1 @@
-print('Hello world')
+test
```

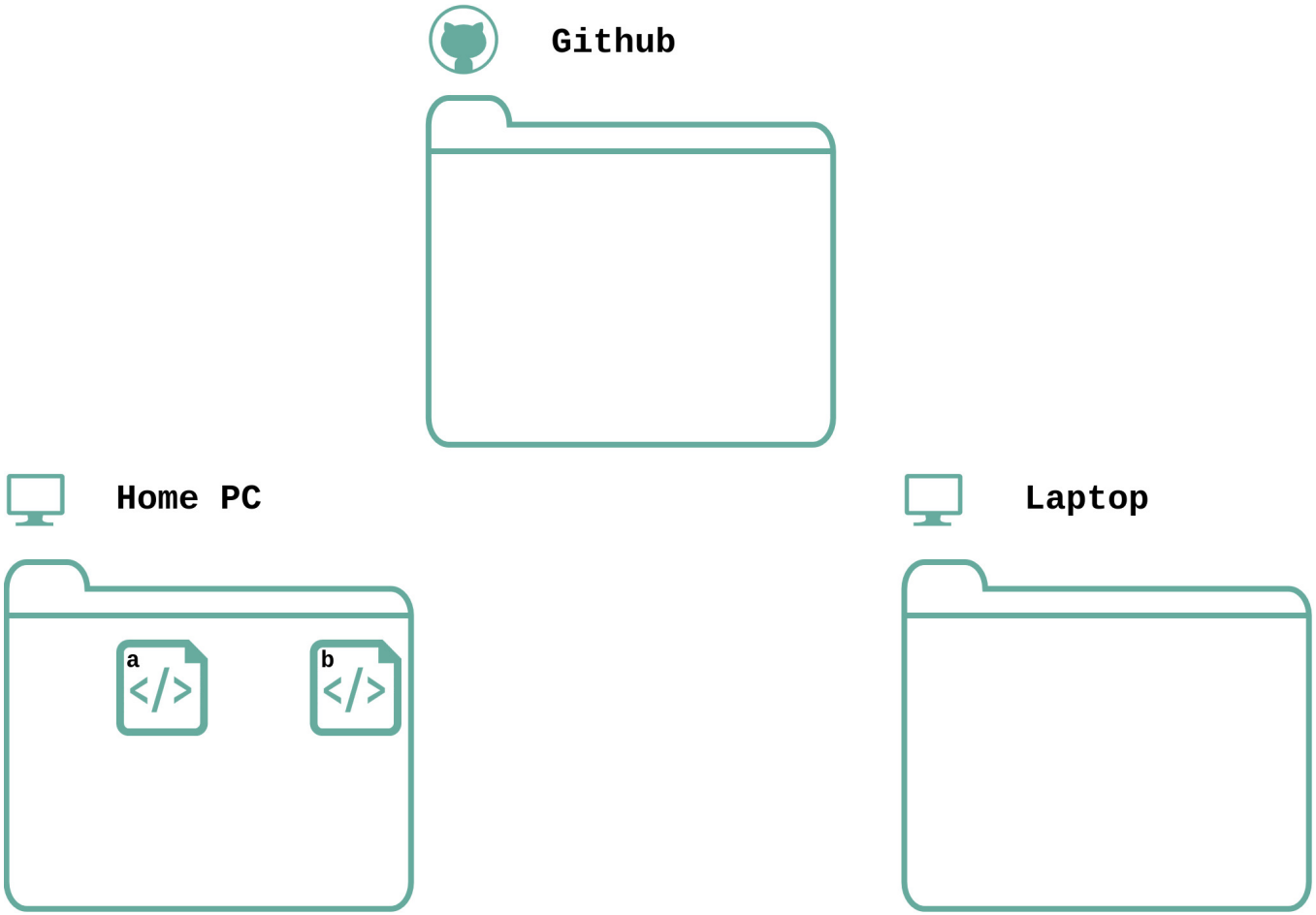
Staging area, git diff and status



Staging area, git diff and status

```
[ferran@arch] → project (U main) git add a
[ferran@arch] → project (S main) git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   a

[ferran@arch] → project (S main) git diff
[ferran@arch] → project (S main)
```

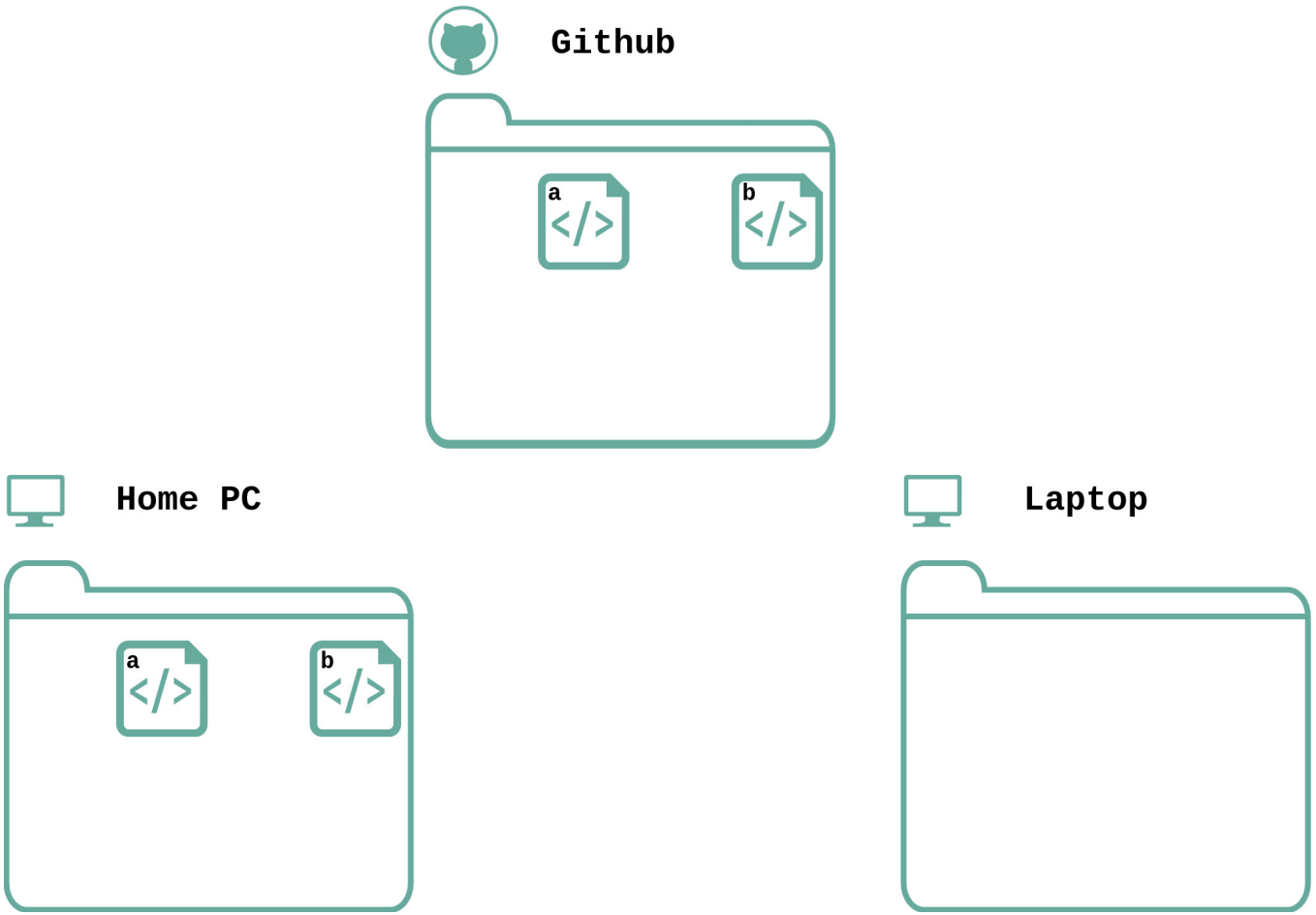
Working with remotes



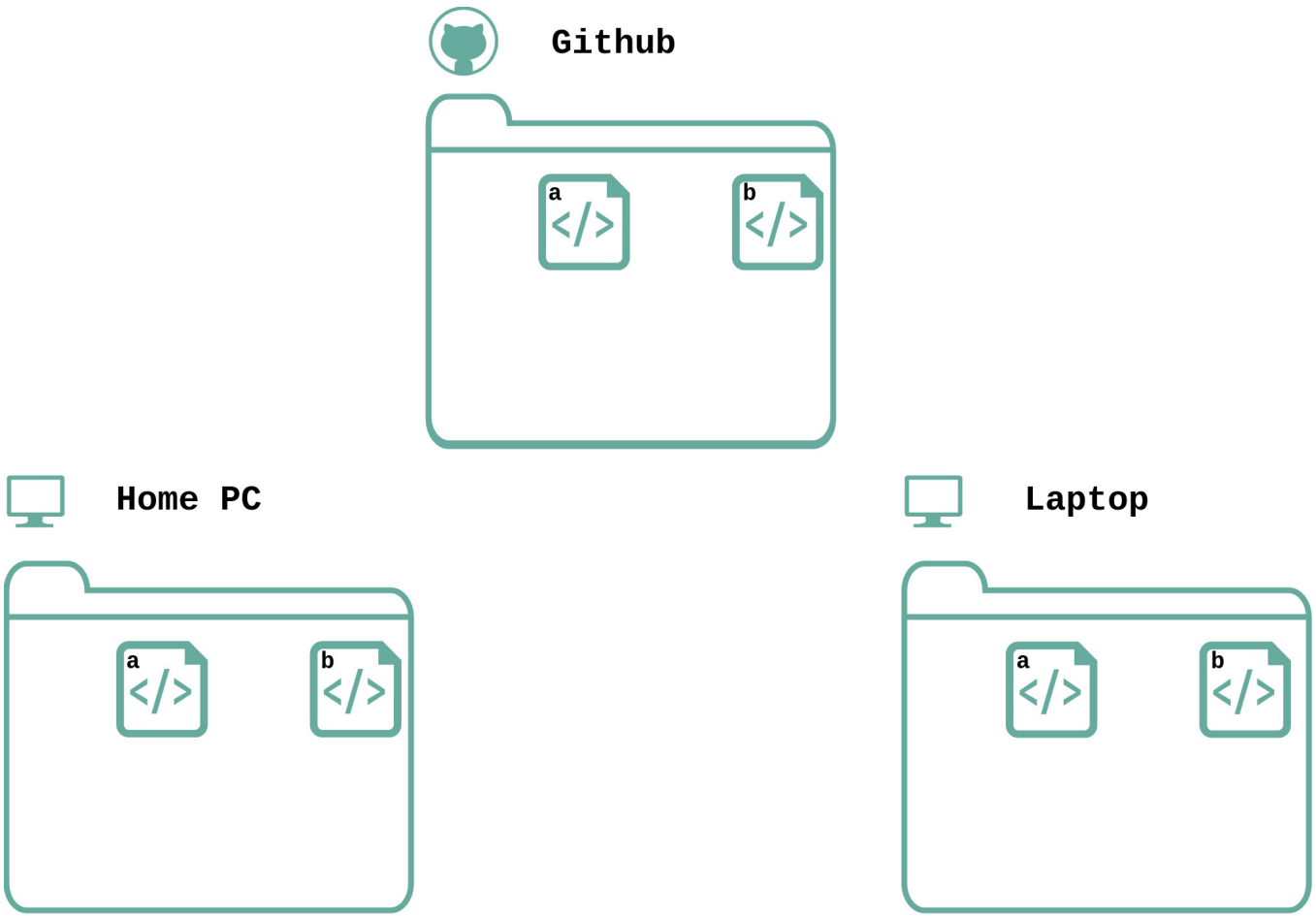
Working with remotes

- `git remote add <name> <url>`: Add a remote repository
- `git branch -M main`: Rename the current branch to main
- `git push -u <remote> <branch>`: Push the current branch to the remote repository
- `git pull <remote> <branch>`: Pull the changes from the remote repository

Working with remotes

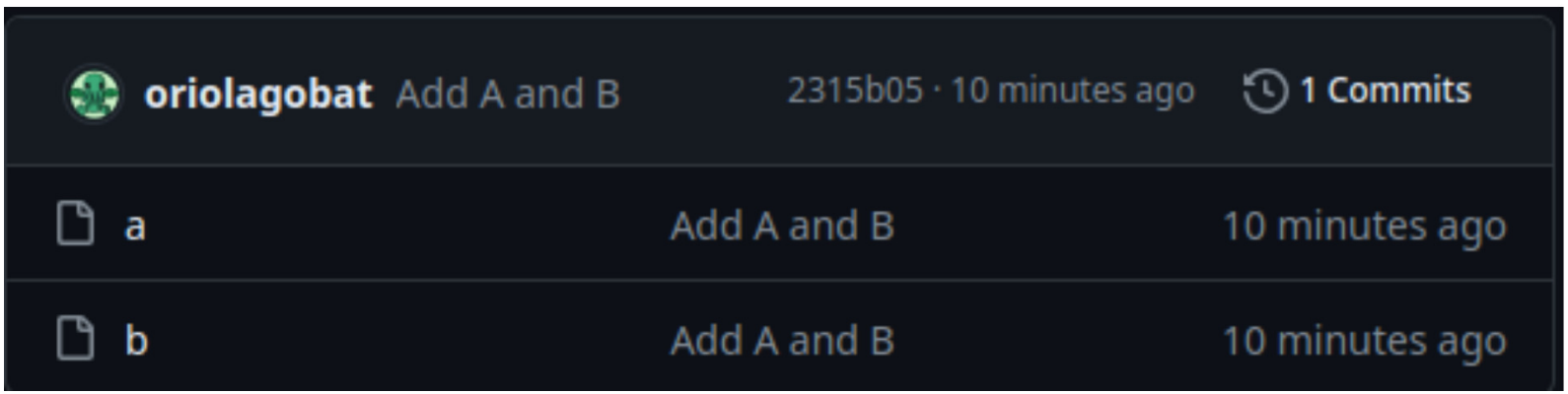


Working with remotes

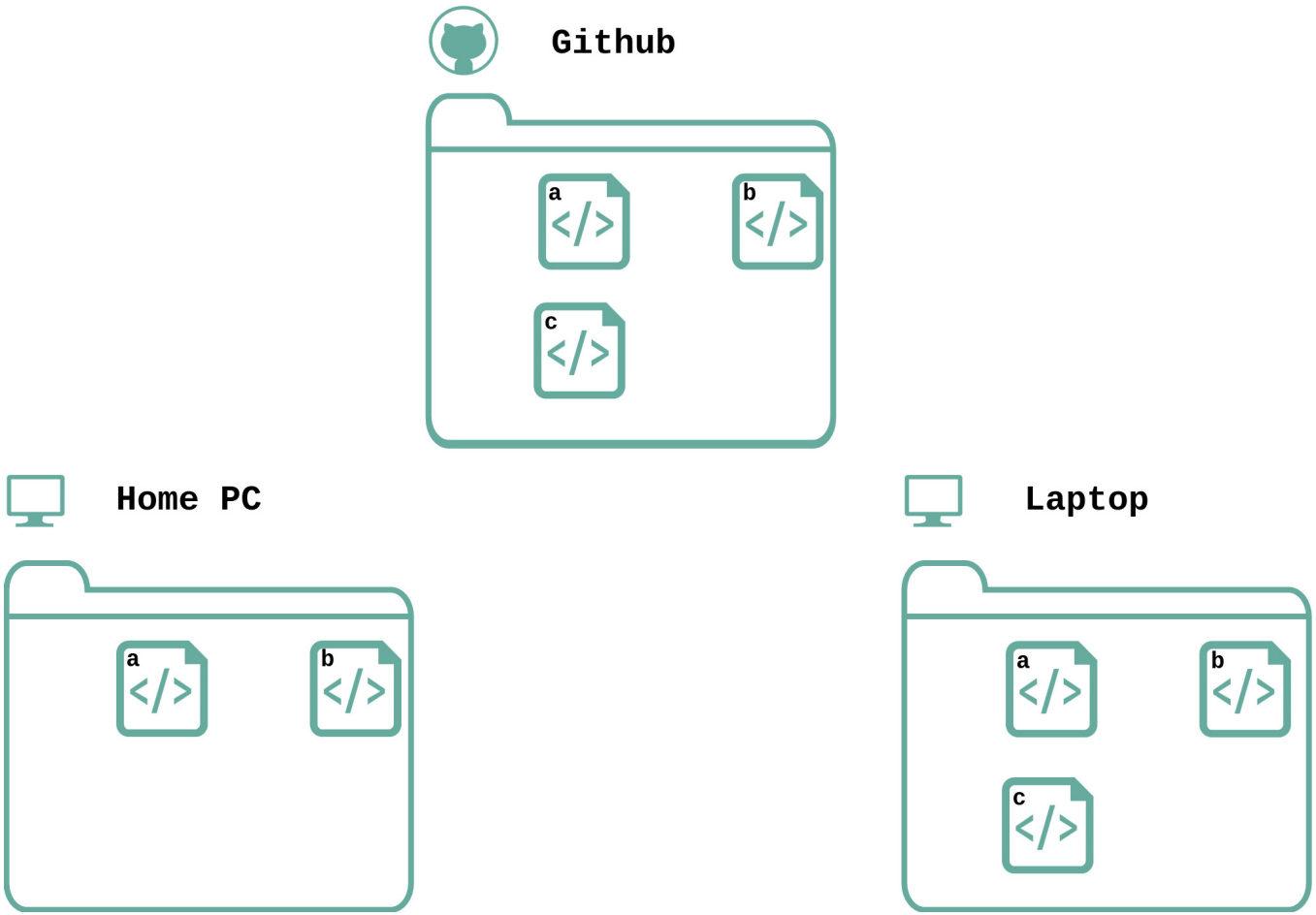


Working with remotes

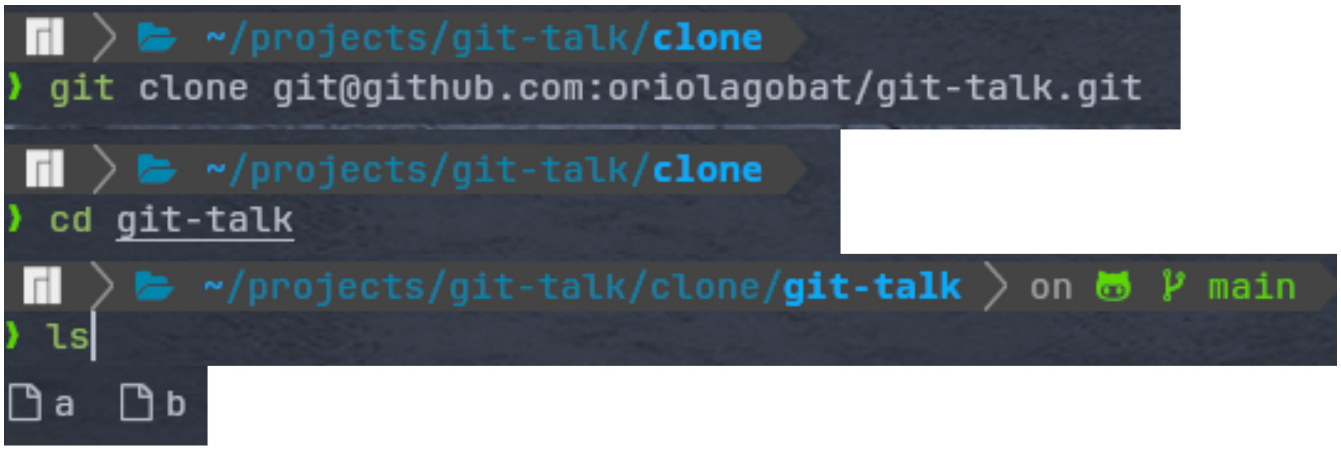
```
git remote add origin git@github.com:oriolagobat/gft-talk.git
git branch -M main
git push -u origin main
```



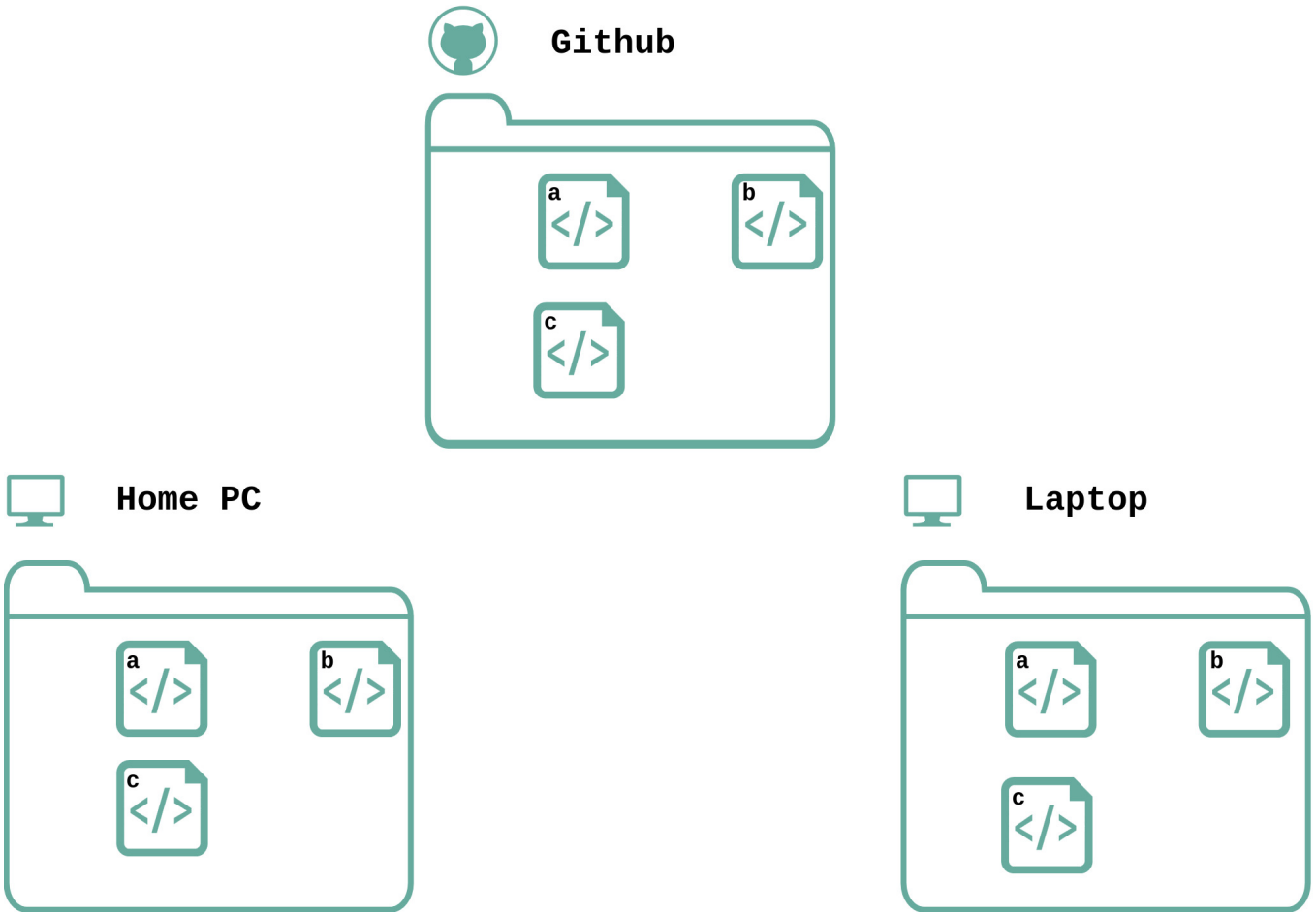
Working with remotes



Working with remotes



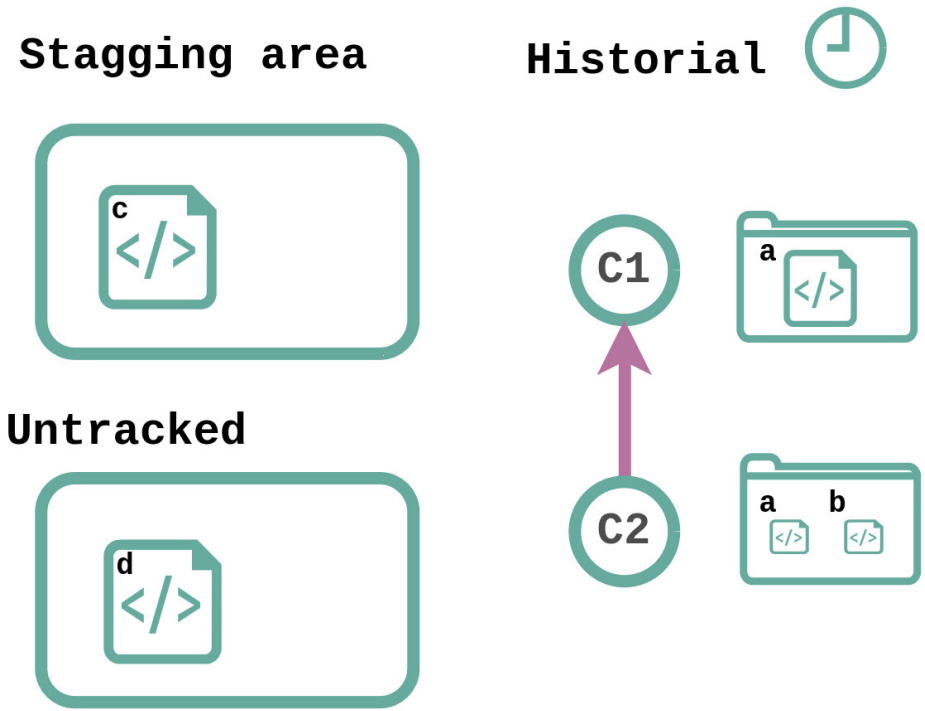
Working with remotes



Working with remotes

```
❯ nvim c
❯ git add c
❯ git commit -m "Add c"
[main 6e42d64] Add c
 1 file changed, 1 insertion(+)
 create mode 100644 c
❯ git push
Confirm user presence for key ED25519-SK SHA256:EYuERof4a3T1WjClYxrE9WdWCG66gO6B4nh6fImO+58
User presence confirmed
Enumerando objetos: 4, listo.
Contando objetos: 100% (4/4), listo.
Compresión delta usando hasta 12 hilos
Comprimiendo objetos: 100% (2/2), listo.
Escribiendo objetos: 100% (3/3), 928 bytes | 928.00 KiB/s, listo.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To github.com:oriolagobat/git-talk.git
   2315b05..6e42d64  main → main
```

Log and resets



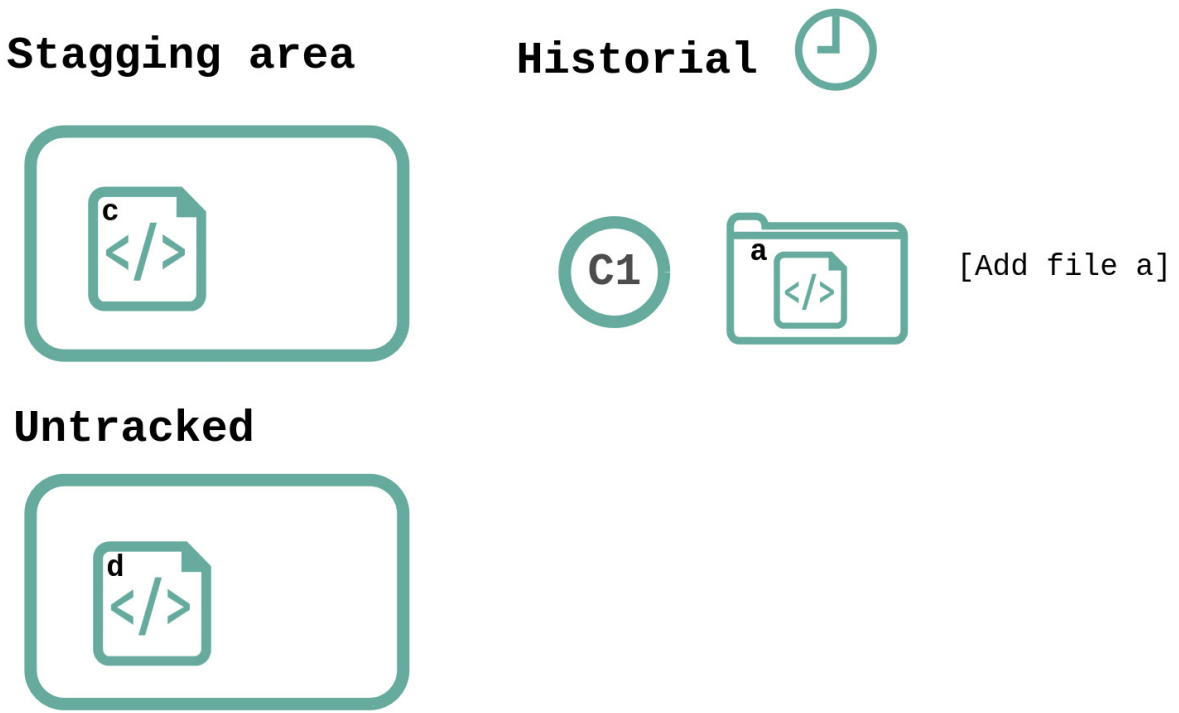
Log and resets

- `git log`: Show the commit history
- `git checkout <commit>`: Move the HEAD to the specified commit
- `git reset --hard <commit>`: Move the HEAD to the specified commit and reset the working directory
- `git reset --soft <commit>`: Move the HEAD to the specified commit and keep the working directory

Log and resets

```
[ferran@arch] ➜ project ( main) touch c
[ferran@arch] ➜ project (! main) touch d
[ferran@arch] ➜ project (! main) git add c
[ferran@arch] ➜ project (S! main) git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   c

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        d

[ferran@arch] ➜ project (S! main)
```

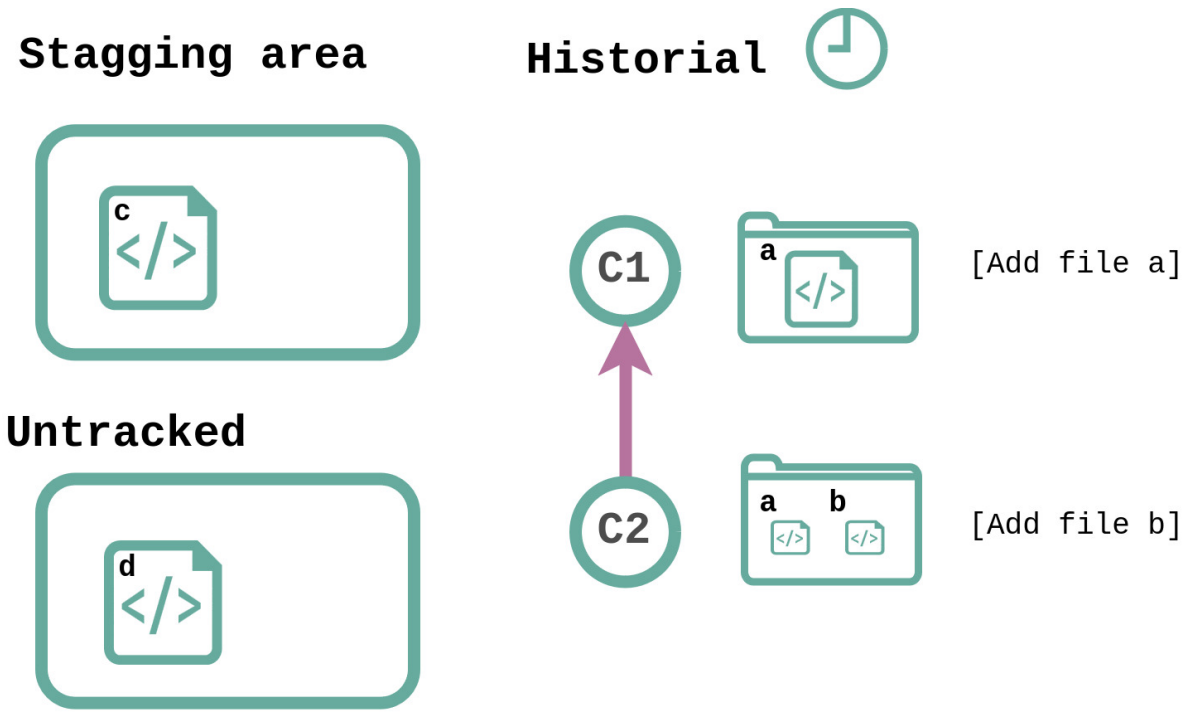
Log and resets



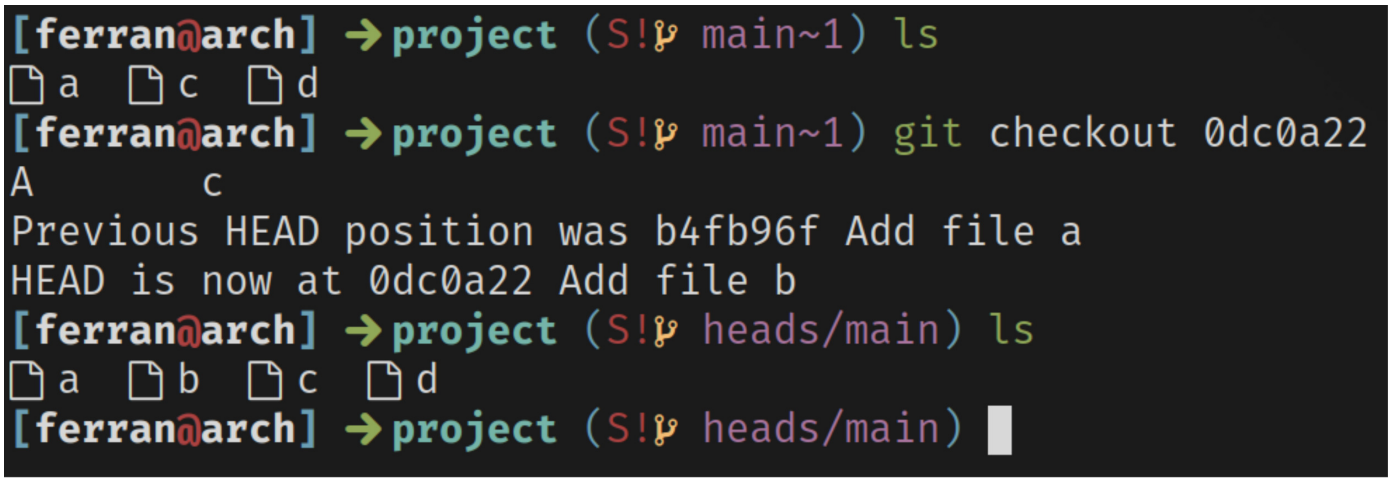
Log and resets

```
[ferran@arch] →project (S!ᚠ heads/main) ls
a  b  c  d
[ferran@arch] →project (S!ᚠ heads/main) git log --oneline
0dc0a22 (HEAD, main) Add file b
b4fb96f Add file a
[ferran@arch] →project (S!ᚠ heads/main) git checkout b4fb96f
A       c
Previous HEAD position was 0dc0a22 Add file b
HEAD is now at b4fb96f Add file a
[ferran@arch] →project (S!ᚠ main~1) git status
HEAD detached at b4fb96f
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   c

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        d

[ferran@arch] →project (S!ᚠ main~1) ls
a  c  d
[ferran@arch] →project (S!ᚠ main~1)
```

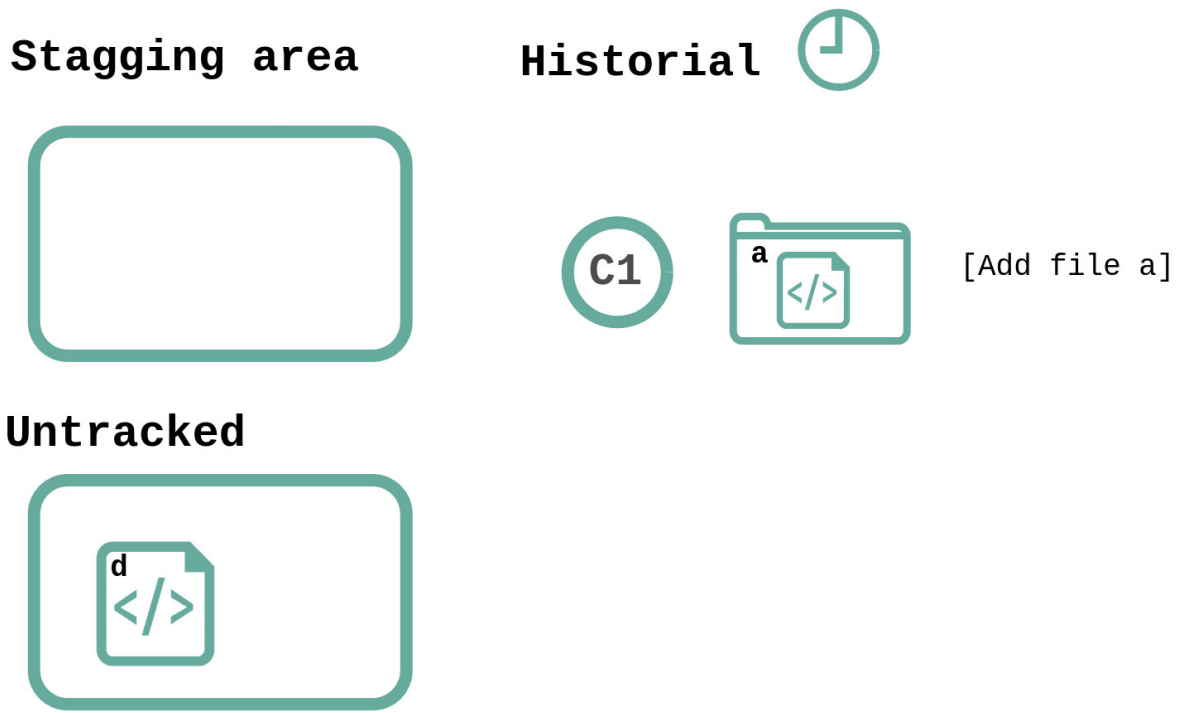
Log and resets



Log and resets



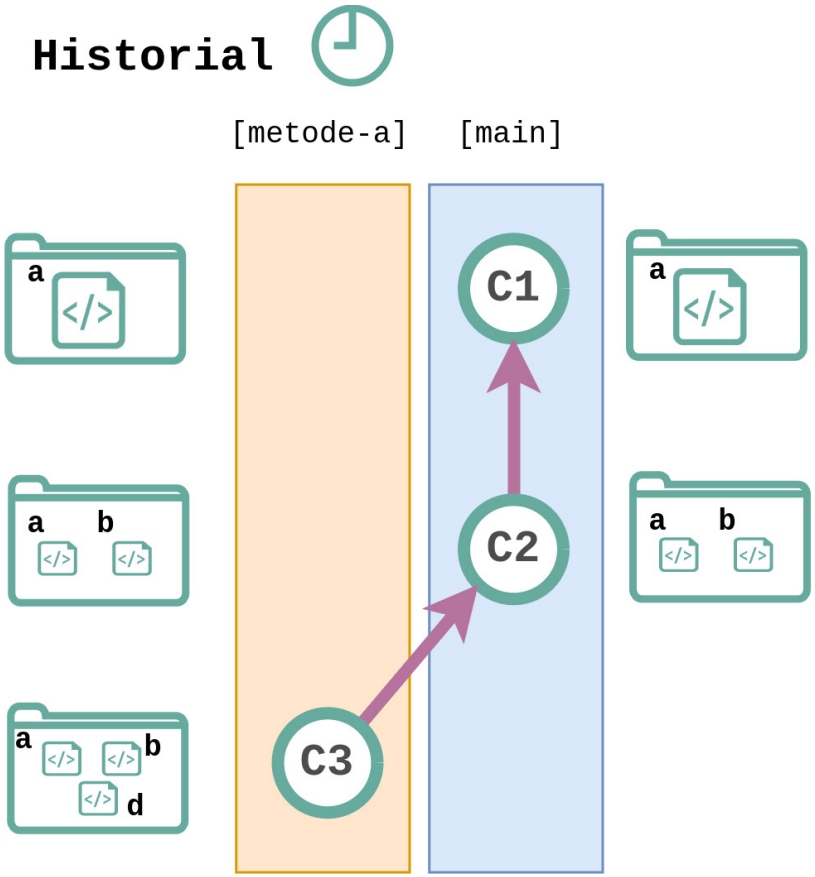
Log and resets



Log and resets

```
[ferran@arch] →project (S!ᚠ main~1) ls
a  c  d
[ferran@arch] →project (S!ᚠ main~1) git checkout 0dc0a22
A       c
Previous HEAD position was b4fb96f Add file a
HEAD is now at 0dc0a22 Add file b
[ferran@arch] →project (S!ᚠ heads/main) ls
a  b  c  d
[ferran@arch] →project (S!ᚠ heads/main) git status
HEAD detached at 0dc0a22
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   c

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        d

[ferran@arch] →project (S!ᚠ heads/main) git reset --hard b4fb96f
HEAD is now at b4fb96f Add file a
[ferran@arch] →project (!ᚠ main~1) ls
a  d
```

Branches



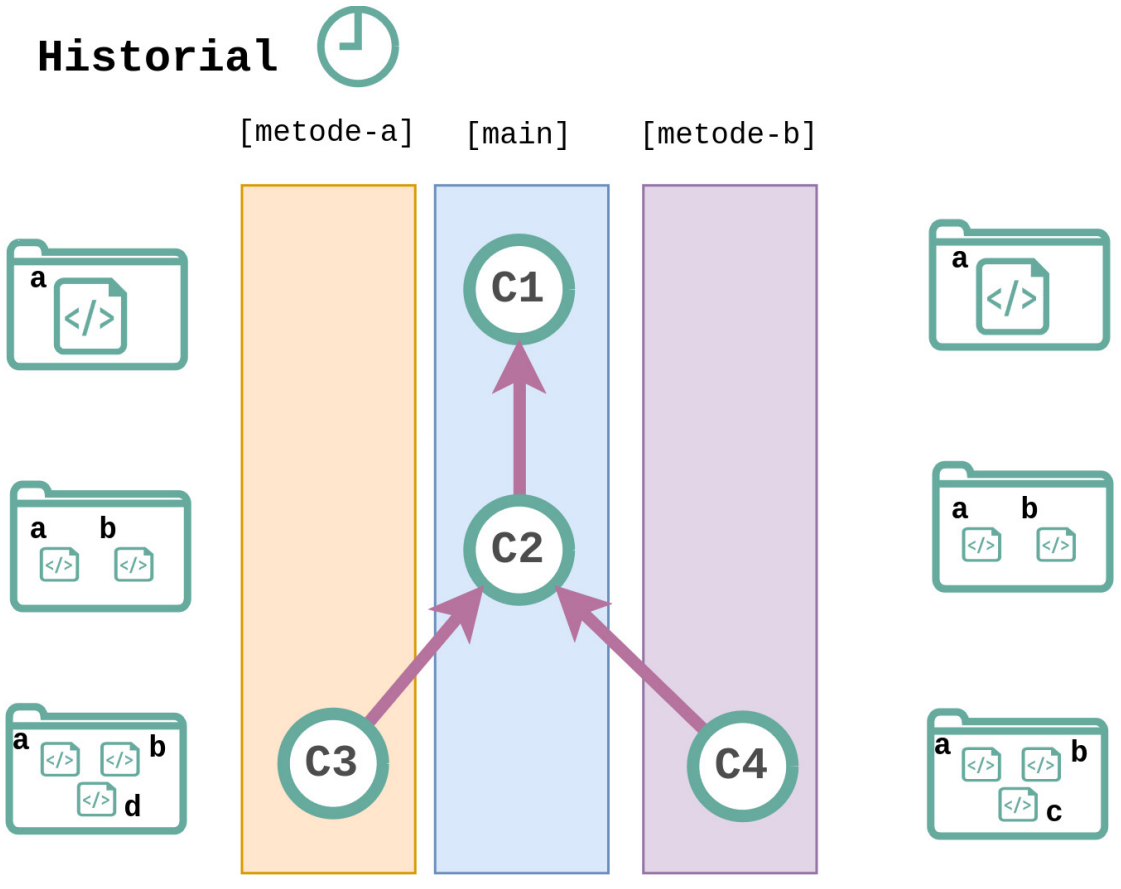
Branches

- `git branch`: List the branches
- `git checkout -b <branch>`: Create a new branch and switch to it
- `git checkout`: Switch to another branch

Branches

```
[ferran@arch] → project ( main) ls
 a  b
[ferran@arch] → project ( main) git checkout -b metode-a
Switched to a new branch 'metode-a'
[ferran@arch] → project ( metode-a) touch d
[ferran@arch] → project (! metode-a) git add d
[ferran@arch] → project (S metode-a) git commit -m "add file d"
[metode-a 017b78f] add file d
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 d
[ferran@arch] → project ( metode-a) git branch
  main
* metode-a
[ferran@arch] → project ( metode-a)
```

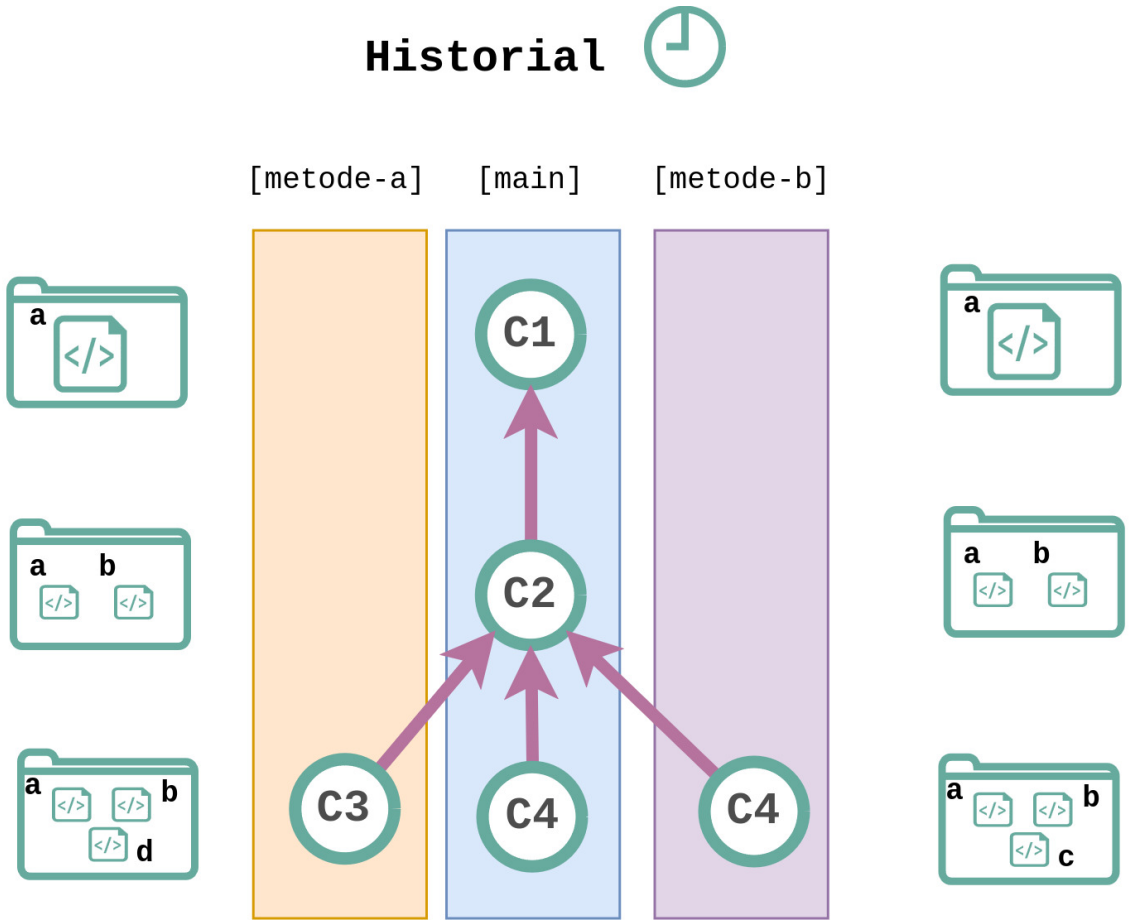
Branches



Branches

```
[ferran@arch] → project (ᚠ metode-b) ls
a  b
[ferran@arch] → project (ᚠ metode-b) touch c
[ferran@arch] → project (!ᚠ metode-b) git add c
[ferran@arch] → project (Sᚠ metode-b) git commit -m "add file c"
[metode-b ea8922f] add file c
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 c
[ferran@arch] → project (ᚠ metode-b) git branch
  main
  metode-a
* metode-b
[ferran@arch] → project (ᚠ metode-b)
```

Merge and rebase



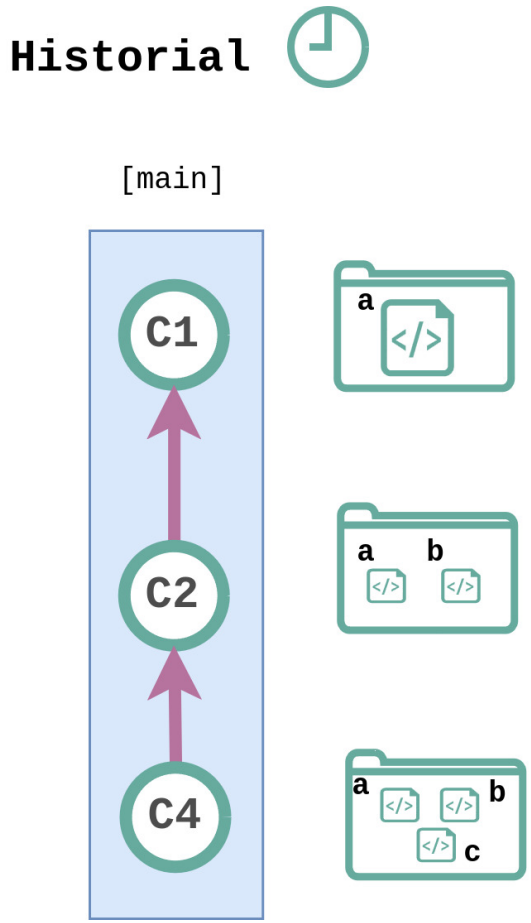
Merge and rebase

- `git merge <branch>`: Merge the specified branch into the current branch
- `git rebase <branch>`: Rebase the current branch onto the specified branch

Merge and rebase

```
[ferran@arch] →project (ᚠ metode-b) git checkout main
Switched to branch 'main'
[ferran@arch] →project (ᚠ main) git merge metode-b
Updating 0dc0a22..ea8922f
Fast-forward
 c | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 c
[ferran@arch] →project (ᚠ main) git log --oneline
ea8922f (HEAD -> main, metode-b) add file c
0dc0a22 Add file b
b4fb96f Add file a
[ferran@arch] →project (ᚠ main) git branch
* main
  metode-a
  metode-b
[ferran@arch] →project (ᚠ main) ls
a  b  c
[ferran@arch] →project (ᚠ main)
```

Merge and rebase



Merge and rebase

```
[ferran@arch] → project (ᚠ main) git branch -D metode-a
Deleted branch metode-a (was 017b78f).
[ferran@arch] → project (ᚠ main) git branch -D metode-b
Deleted branch metode-b (was ea8922f).
[ferran@arch] → project (ᚠ main) git branch
* main
[ferran@arch] → project (ᚠ main)
```

Standard commits

Standard commits are a set of rules to write commit messages that makes them more readable and easier to understand. They are based on the following structure:

- `<type>(<scope>): <subject>`
- `<body>`
- `<footer>`

The different types are:

- `feat`: A new feature
- `fix`: A bug fix
- `docs`: Documentation changes
- `refactor`: A code change that neither fixes a bug nor adds a feature
- `ci`: Changes to the CI configuration files and scripts
- `test`: Adding missing tests

Standard commits

```
.
├── main.py
├── test.py              $ git commit -m "test(about-page): add test for about page"
├── requirements.txt
├── README.md
└── website
    ├── about.md
    └── index.md
```

Standard commits

```
.
├── main.py
├── test.py
├── requirements.txt
├── README.md
└── website
    ├── about.md                    $ git commit -m "feat(about-page): add skeleton"
    └── index.md
```

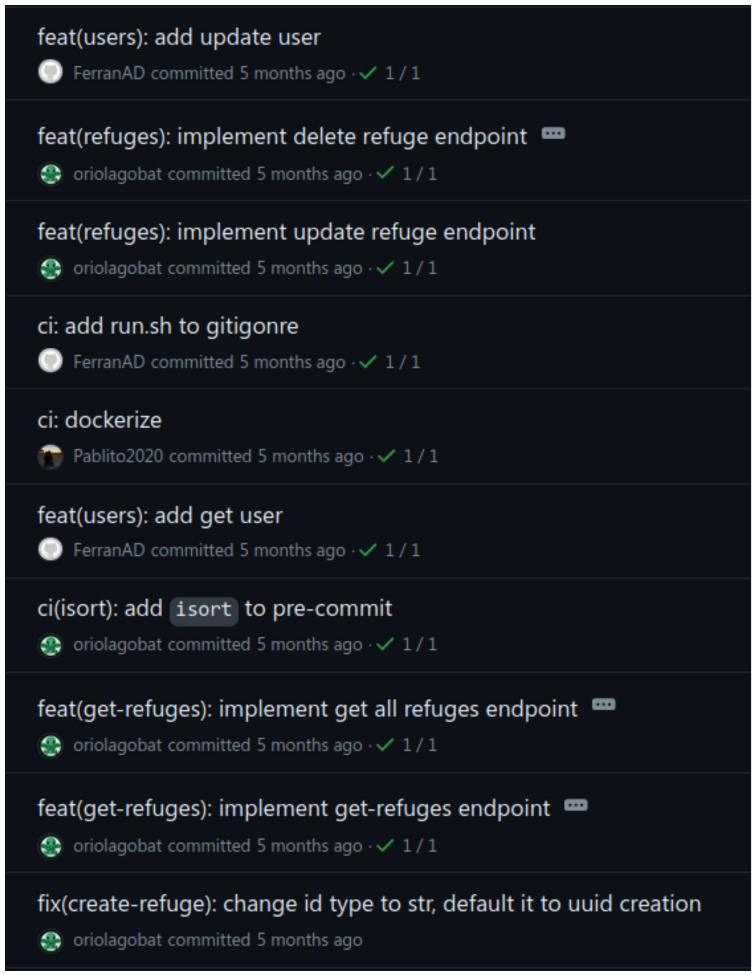
Standard commits

```
.
├── main.py
├── test.py
├── requirements.txt
├── README.md                                $ git commit -m "docs(readme): add header"
└── website
    ├── about.md
    └── index.md
```

Standard commits

```
.
├── main.py
├── test.py
├── requirements.txt      $ git commit -m "build(requirements): add `mkdocs` package"
├── README.md
└── website
    ├── about.md
    └── index.md
```

Standard commits



Standard commits

- https://github.com/RefuAPP/refuapp
- https://github.com/Spam-Number-Filter/Spam-Filter-WebPage
- https://github.com/SNMP-Python/snmp-data-analyzer
- https://github.com/ToDoTurtle/ToDoTurtle-Android
- https://github.com/hackudc-unplug/backend

Merge vs Rebase

- **Merge:**
  - Integrates changes from another branch into the current branch.
  - Creates a new commit with a merge message.
  - Maintains a linear history with clear separation of work.
- **Rebase:**
  - Replays your commits on top of another branch.
  - Rewrites history, making the current branch appear as if the commits were originally made on the other branch.
  - Creates a cleaner linear history but can be risky if the branch has already been shared.

Merge vs Rebase

Use Merge:

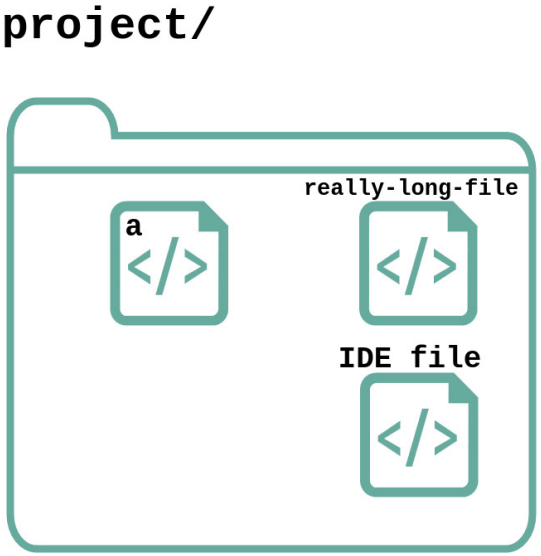
- When collaborating with others and want a clear history of who made what changes.
- When the branch being merged into has already been shared with others.

Use Rebase:

- When working on a private branch and want a clean, linear history.
- Before pushing your branch to a shared repository to avoid merge commits.

- `git merge <branch-name>`: Merges the specified branch into the current branch.
- `git rebase <branch-name>`: Rebases the current branch onto the specified branch.
- `git merge --abort`: Aborts a merge that is in progress.
- `git rebase --abort`: Aborts a rebase that is in progress.

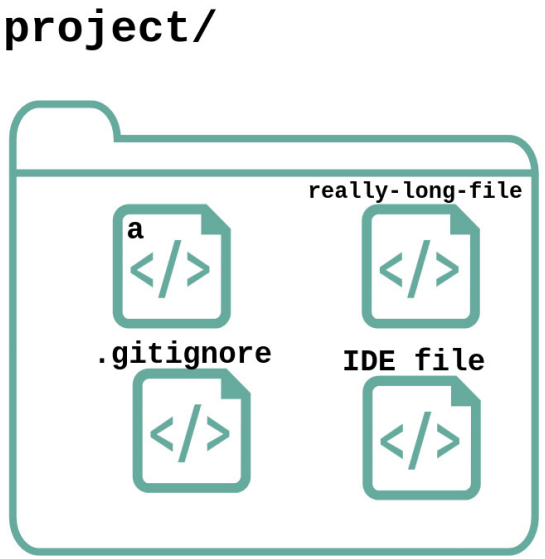
Gitignore



Gitignore

```
) git status
En la rama master

No hay commits todavía

Archivos sin seguimiento:
  (usa "git add <archivo>..." para incluirlo a lo que será confirmado)
        a
        ide-file
        really-long-file

no hay nada agregado al commit pero hay archivos sin seguimiento presentes (usa "git add" para hacerles seguimiento)
```

Gitignore



Gitignore

```
❯ cat .gitignore
       File: .gitignore
   1   really-long-file
   2   ide-file
❯ git status
En la rama master

No hay commits todavía

Archivos sin seguimiento:
  (usa "git add <archivo>..." para incluirlo a lo que será confirmado)
        .gitignore
        a

no hay nada agregado al commit pero hay archivos sin seguimiento presentes (usa "git add" para hacerles seguimiento)
```

Gitignore

Stash

Git stash allows you to temporarily save your uncommitted changes without losing them. This is useful when you need to switch tasks or want to keep your working directory clean.

- `git stash`: Creates a stash entry, saving your uncommitted changes.
- `git stash list`: Shows a list of all your stashed changes.
- `git stash apply <stash@n>`: Applies the most recent stash (or the nth stash) on top of your current work.
- `git stash pop <stash@n>`: Applies the most recent stash (or the nth stash) and removes it from the stash list.
- `git stash drop <stash@n>`: Removes the specified stash entry from the list.

Coauthor

In Git, commits are attributed to the user who committed them by default. However, you can add co-authors to a past commit if someone else contributed to the changes.
Important note: Modifying past commits directly is generally discouraged as it rewrites history and can cause issues in shared repositories. Use this feature with caution.

There are two main approaches to adding a co-author:

1. **Interactive rebase:**
   - Use `git rebase -i <commit-hash>` to initiate interactive rebase.
   - Edit the commit message and add the co-author using the format: "Co-authored-by: Name <email>".
   - Save and continue the rebase to apply the changes.
2. **Commit amend:**
   - Use `git commit --amend` to open the commit message editor for the latest commit.
   - Add the co-author line in the format: "Co-authored-by: Name <email>".
   - Stage any additional changes and save the commit message.

Rebase Interactive

Git rebase interactive allows you to rewrite commit history by modifying or discarding commits. This can be helpful for cleaning up your branch history, squashing multiple commits into one, or fixing mistakes.

Use with caution: Rebasing rewrites history, so ensure the branch hasn't been shared with others before using this feature.

Steps to perform an interactive rebase:
1. Run `git rebase -i <starting-commit-hash>`. This opens an editor showing your commit history. 2. Edit the instruction next to each commit:

- pick: Keep the commit unchanged.
- edit: Modify the commit message and content.
- squash: Combine the current commit with the previous one.
- drop: Discard the commit.

3. Save the changes in the editor. 4. Git will replay the commits based on your instructions, rewriting the branch history.