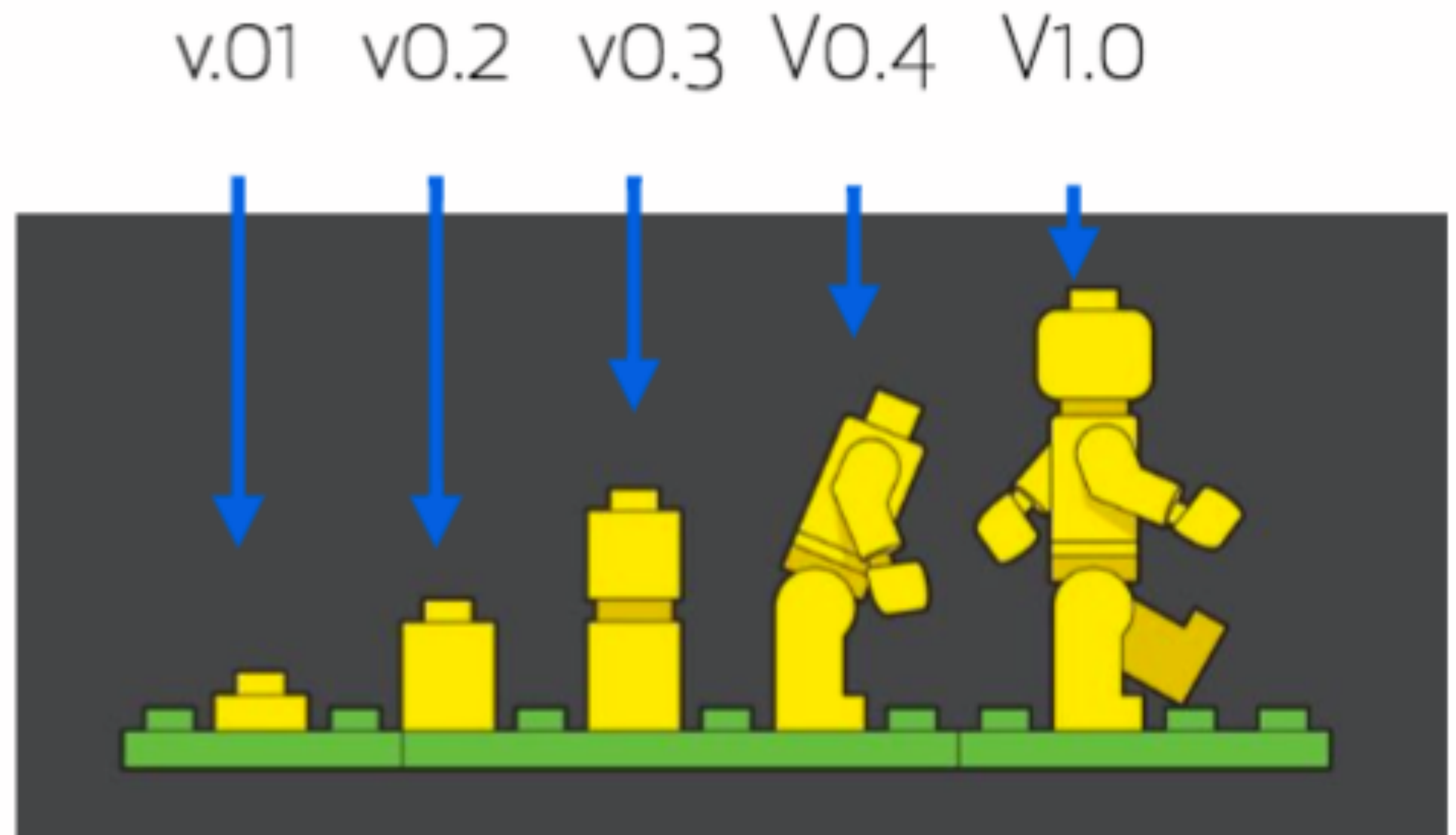


Git is a *version control system*

A tool that lets you track your progress over time.



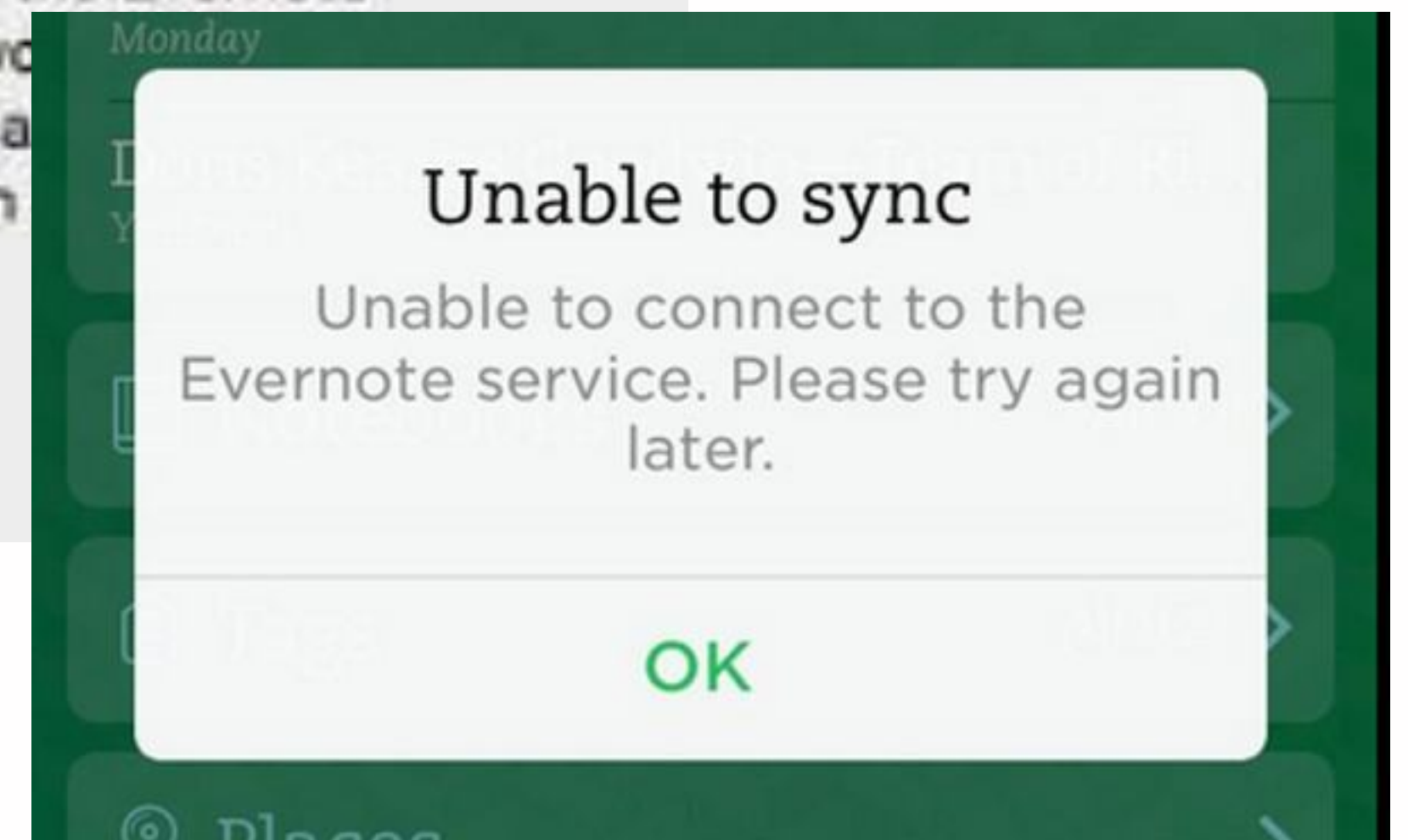
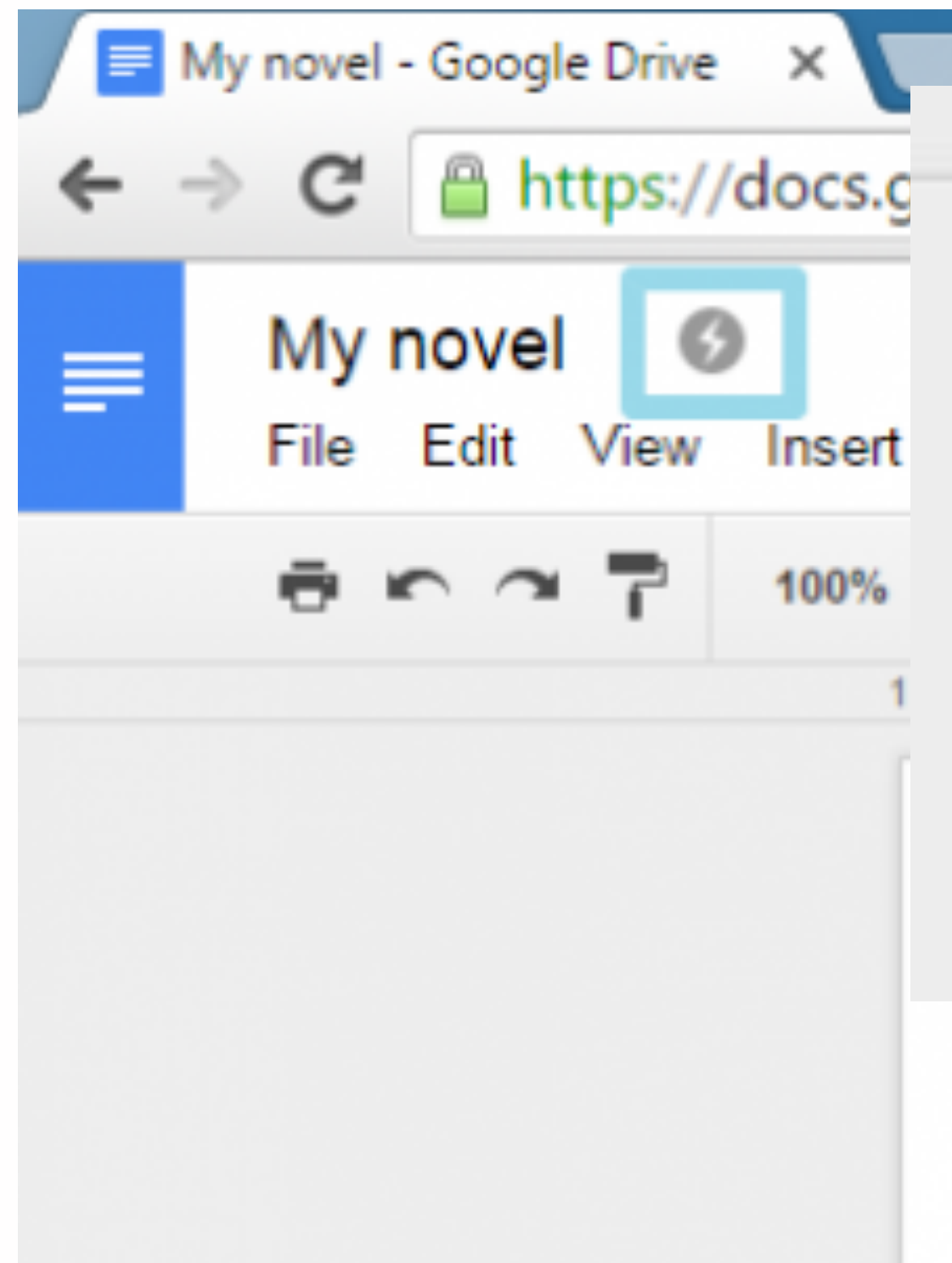
Git takes snapshots

Save snapshots to your history to retrace your steps.

Also keeps others up-to-date with your latest work.



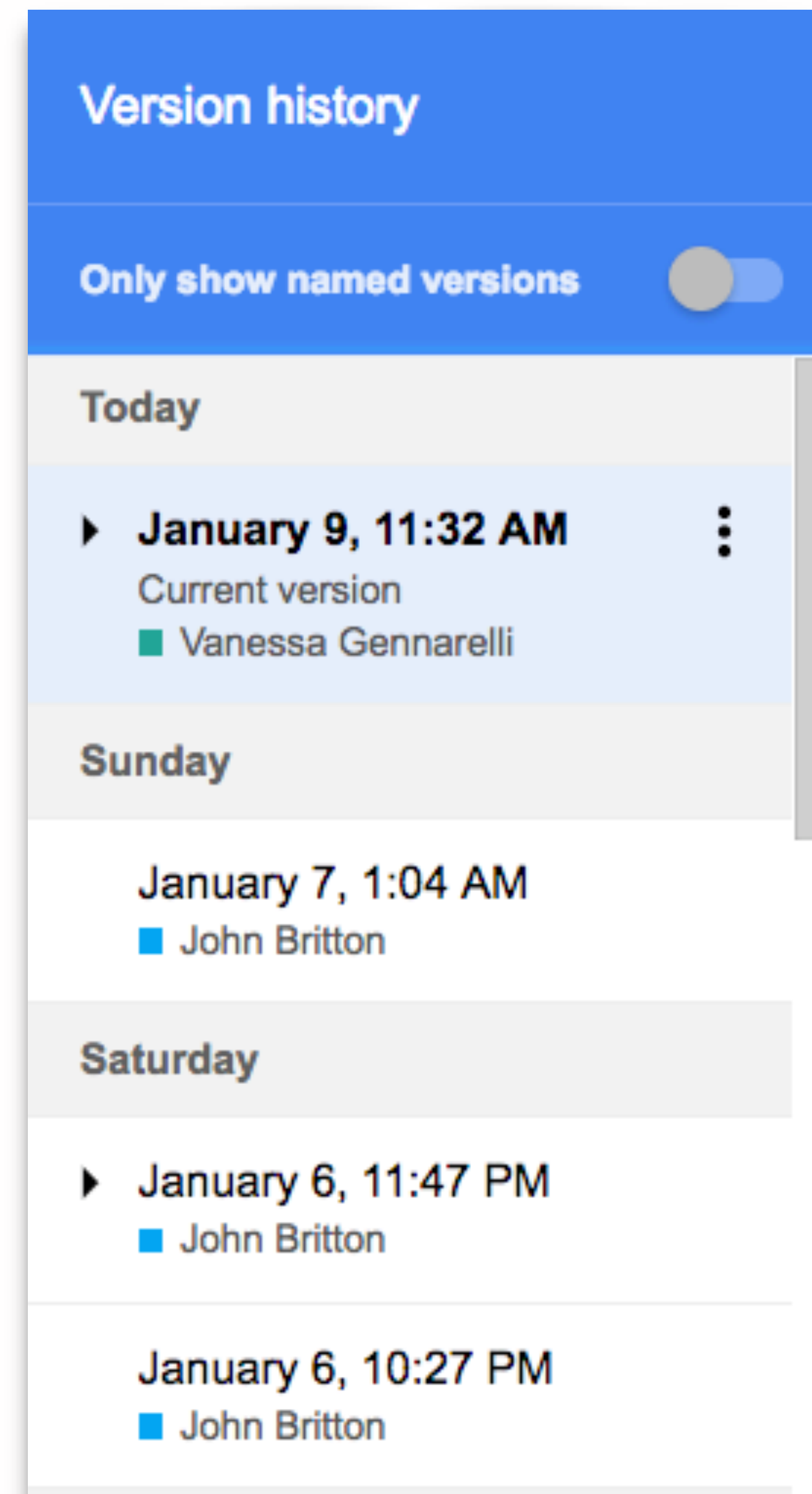
Centralized systems require coordination...



Order with coordination:

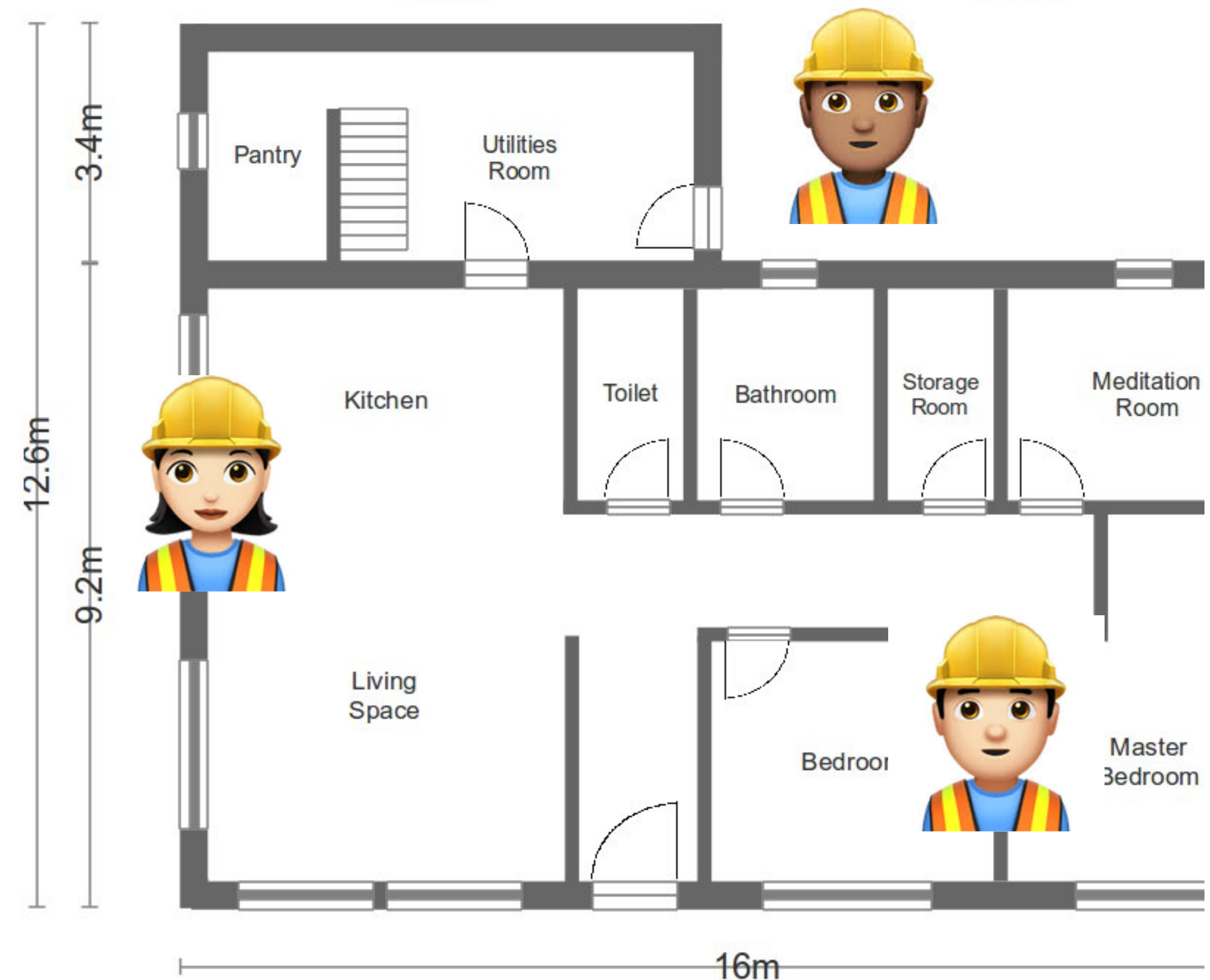
In a centralized system, you can objectively call versions a numerical progression: version 1, version 2, version 3...

Since John made a new version before Vanessa, his is $n+1$, and Vanessa is $n+2$.



Working in parallel: order without coordination

Git goes after this idea of distributed version control, so you can keep track of your versions without coordination.



In your terminal, check to see if you have
Git installed.

```
git --version
```



If it's not installed, configure Git to recognize you:

```
git config user.name "Mona Lisa"
```

```
git config --global user.email  
"email@example.com"
```



A repository holds the entire history of your project

A repository is the unit of separation between projects in Git.

Each project, library or discrete piece of software should have its own repository.



Create a repository

```
cd desktop  
git init exercise-1  
cd exercise-1  
ls -al
```



Git is like a desk

Working directory
where you write

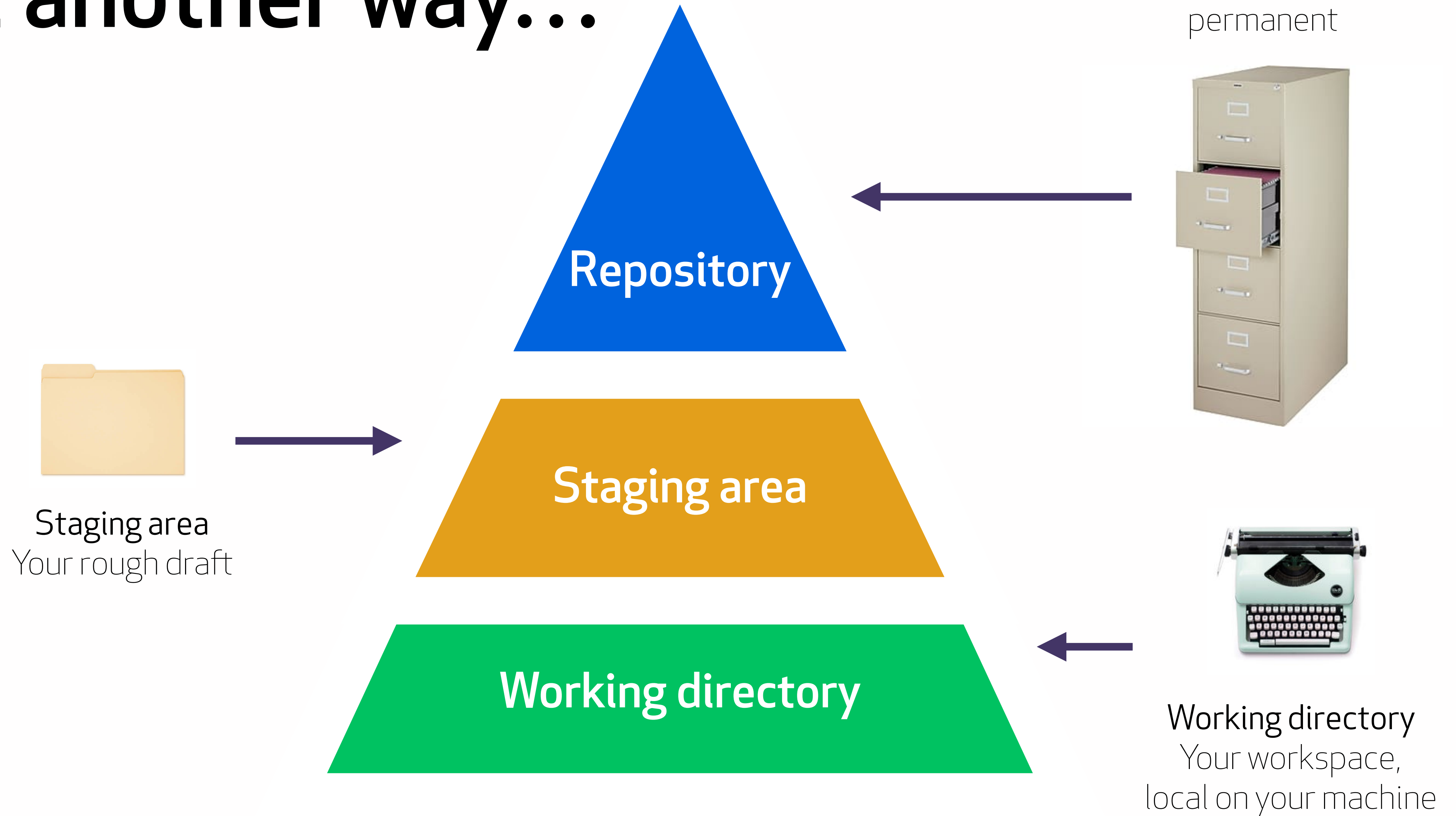


Staging area
rough draft, in a
manila folder

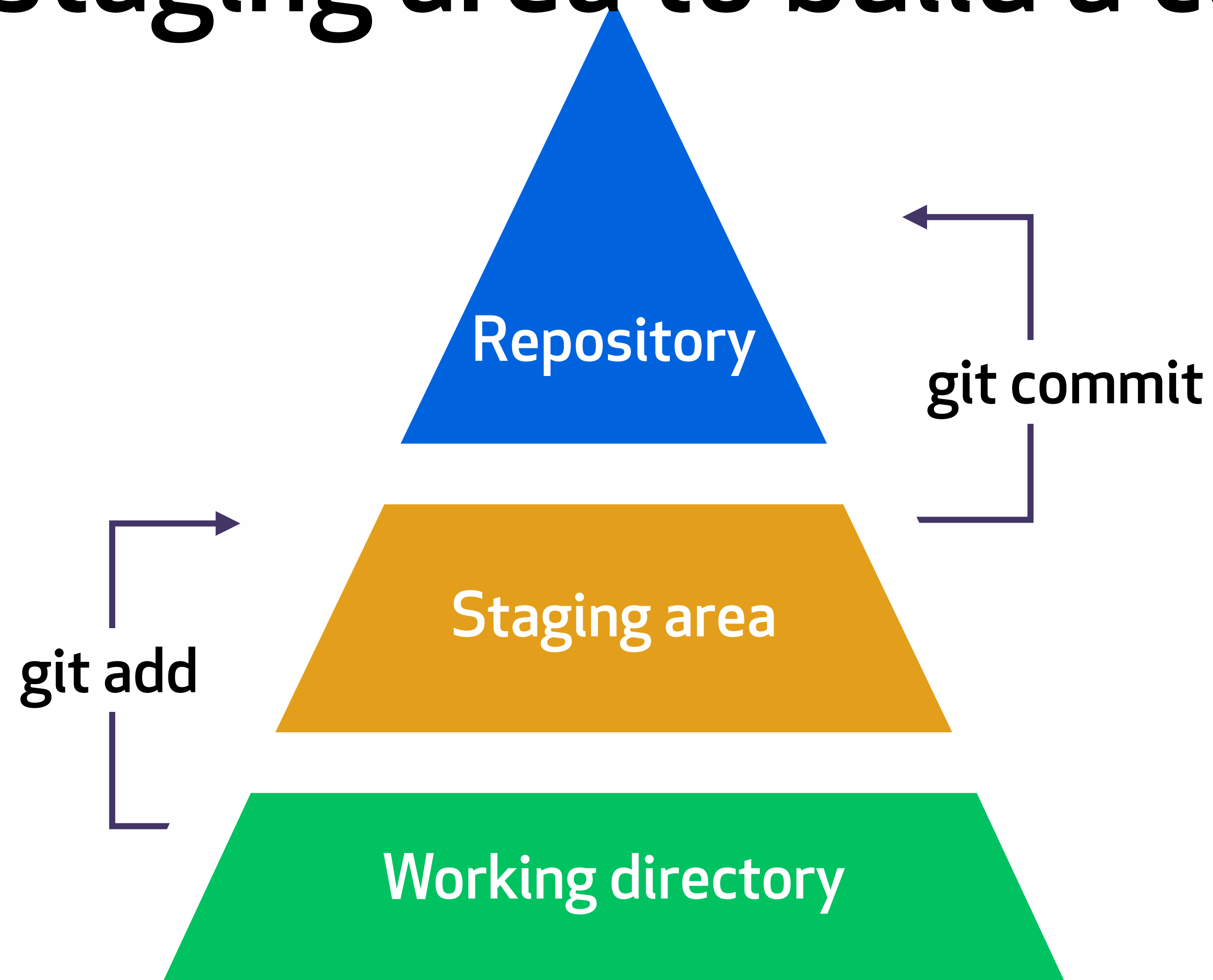
Repository
final draft
in the filing cabinet



Put another way...



Use the staging area to build a commit



Create a file in your Git repository + add it to staging.

```
touch readme.md  
git status  
git add readme.md  
git status
```



Making commits

`'git commit'`

tells Git to save that portion of the project from the staging area into the repository history.



**Understanding the state
of your repository**

Let's put together an exercise-1 for your students

1. Edit the readme with directions for exercise-1.
2. We're going to add the changes to the staging area.
3. Commit those changes.
4. 🎉



Understanding the state of your repository

```
git status
```

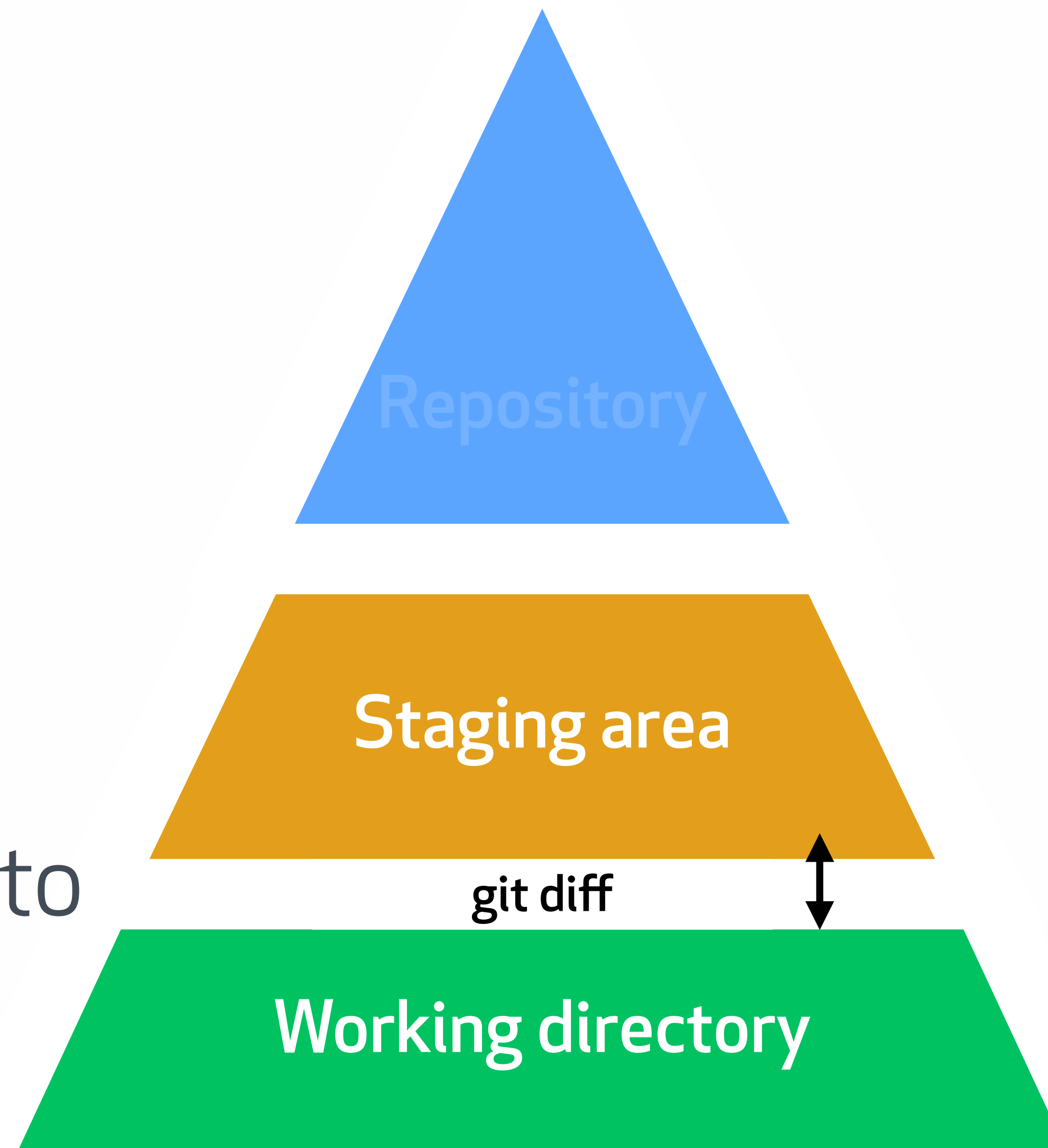
```
git diff
```

```
git diff --staged
```



**When we run git diff
what two things are we comparing?**

git diff



Compares staging to
working directory.

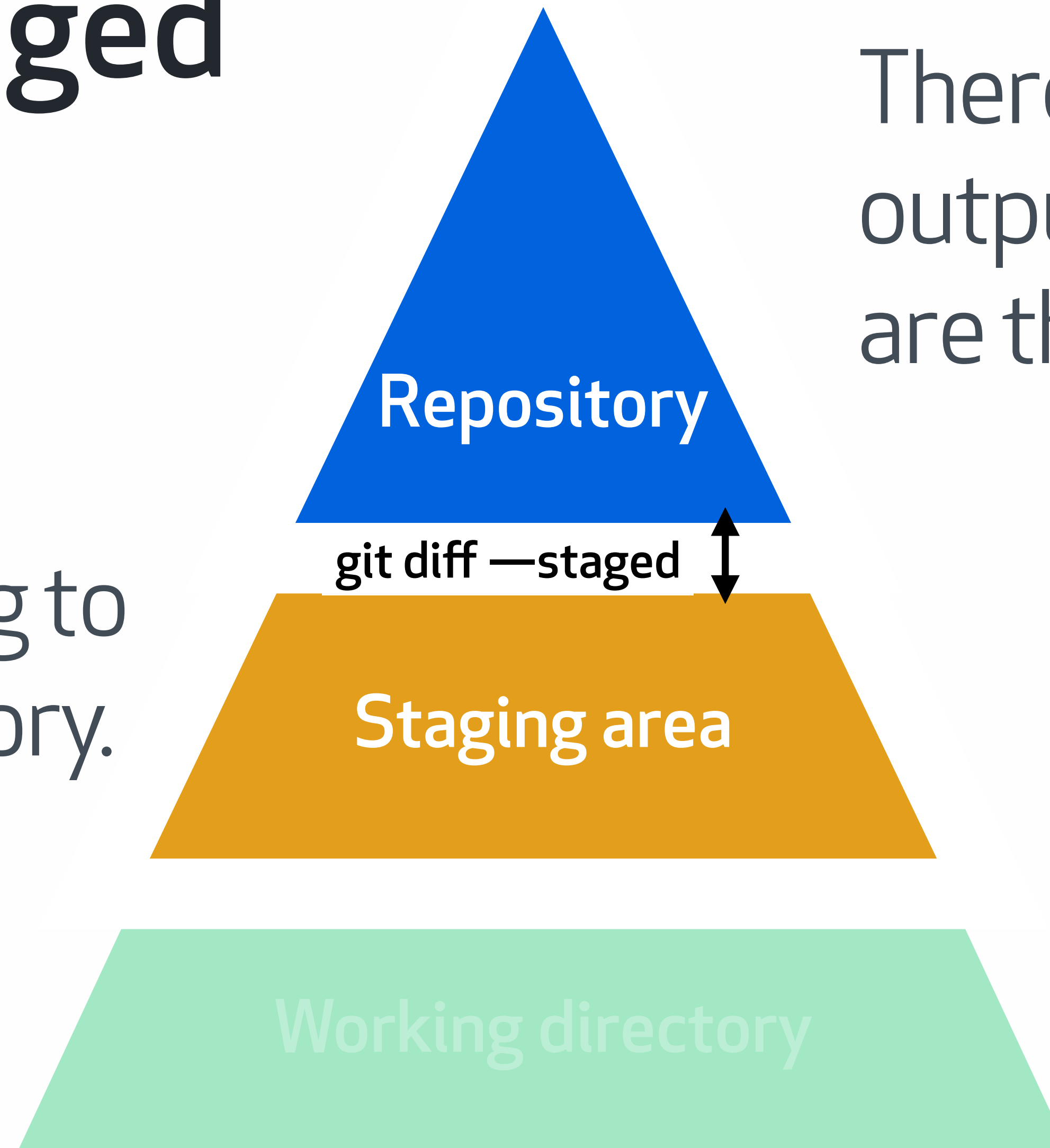
There's no
output if they
are the same.



git diff --staged

There's no output if they are the same.

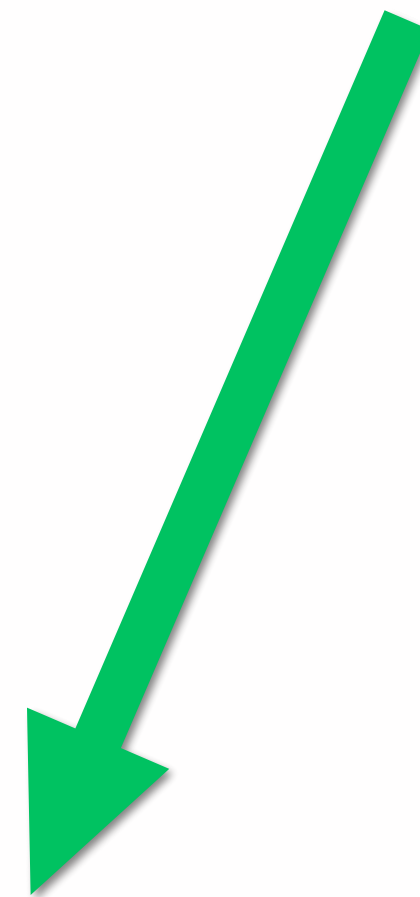
Compares staging to repository directory.



Git allows you to be selective

You can fix a bug across several different files in the same commit.

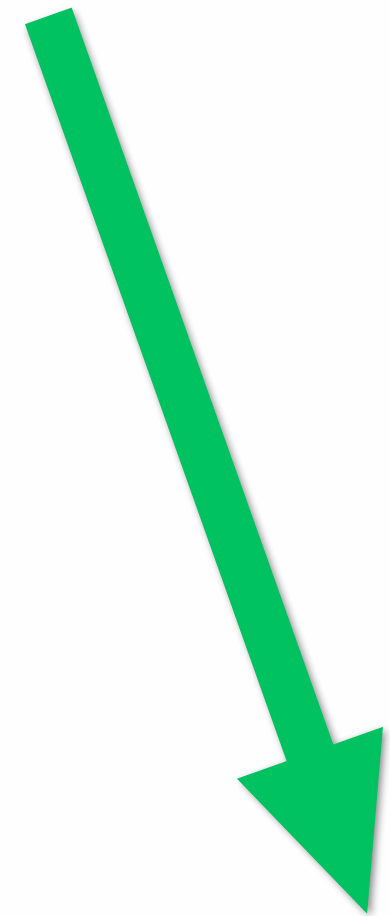
```
git commit -m 'new TA email'
```



Exercise-1



Exercise-2



Exercise-3



But commits should be logically grouped

Don't mix typo corrections and new features.

If the feature gets rolled back, you re-introduce the typo.

```
git commit -m 'typo in readme.md'
```



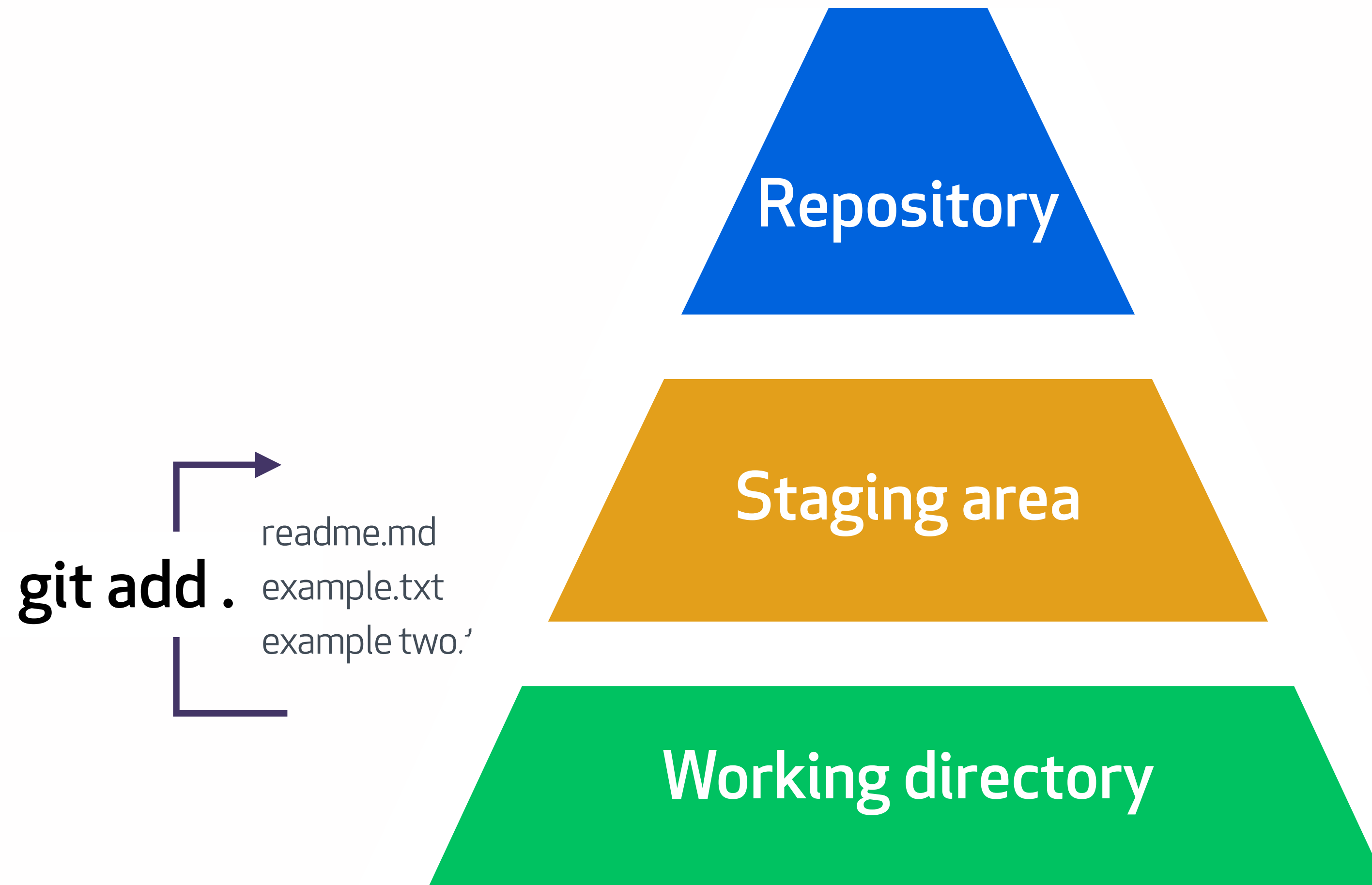
```
git commit -m 'new signup flow.'
```



```
git commit -m 'fix typo, add  
field to signup flow, create  
parallax effect'
```



It's why you should never use `git add .`



It's why you should never use `git add .`



it stages changes that aren't logically related...



Imagine if you revealed solutions in exercise-1

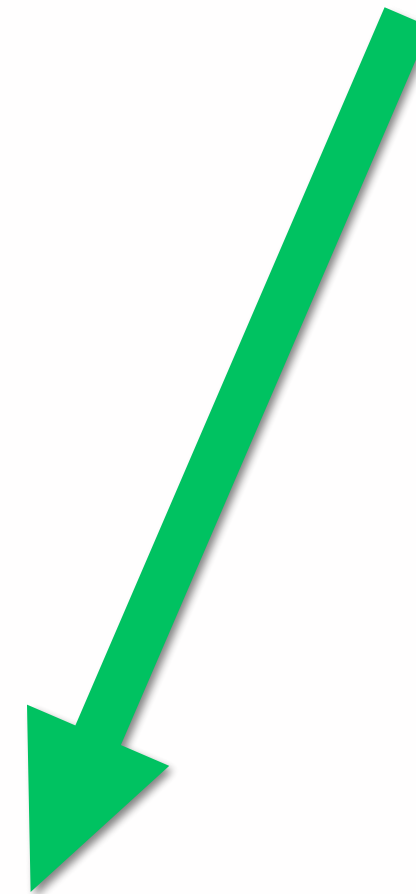
```
git commit -m 'remove key data'
```

You'd need to update Exercise-1, but you don't need to touch 2 or 3.

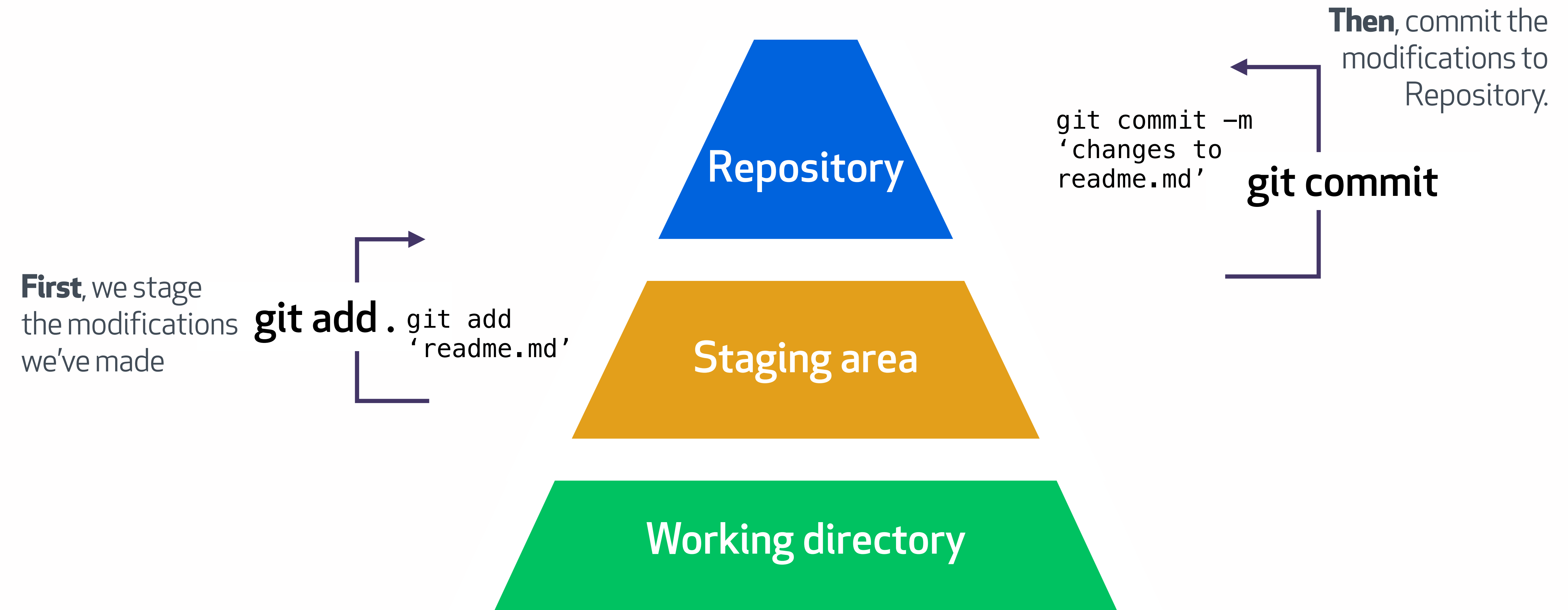
Exercise-1

Exercise-2

Exercise-3

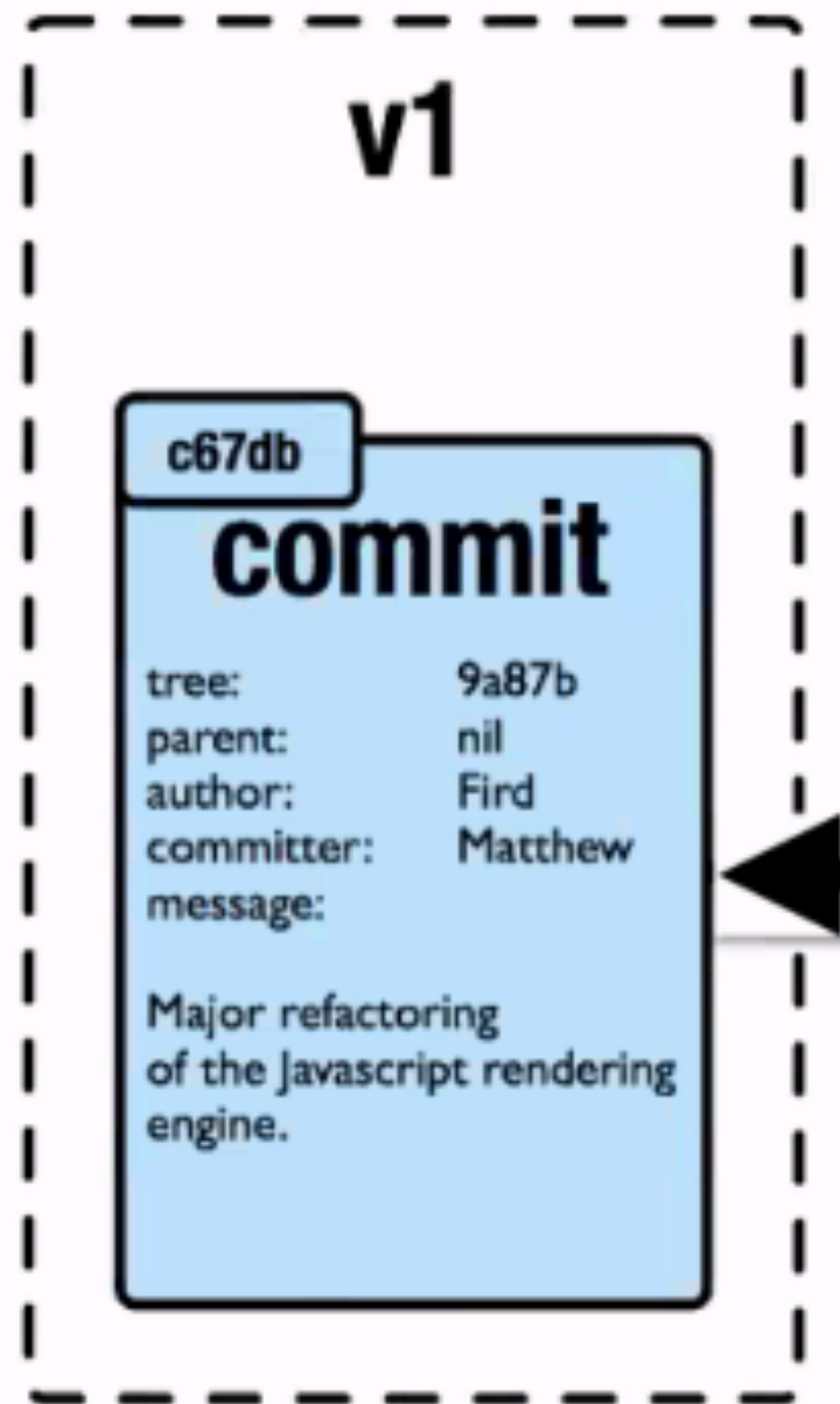


Order of operations:

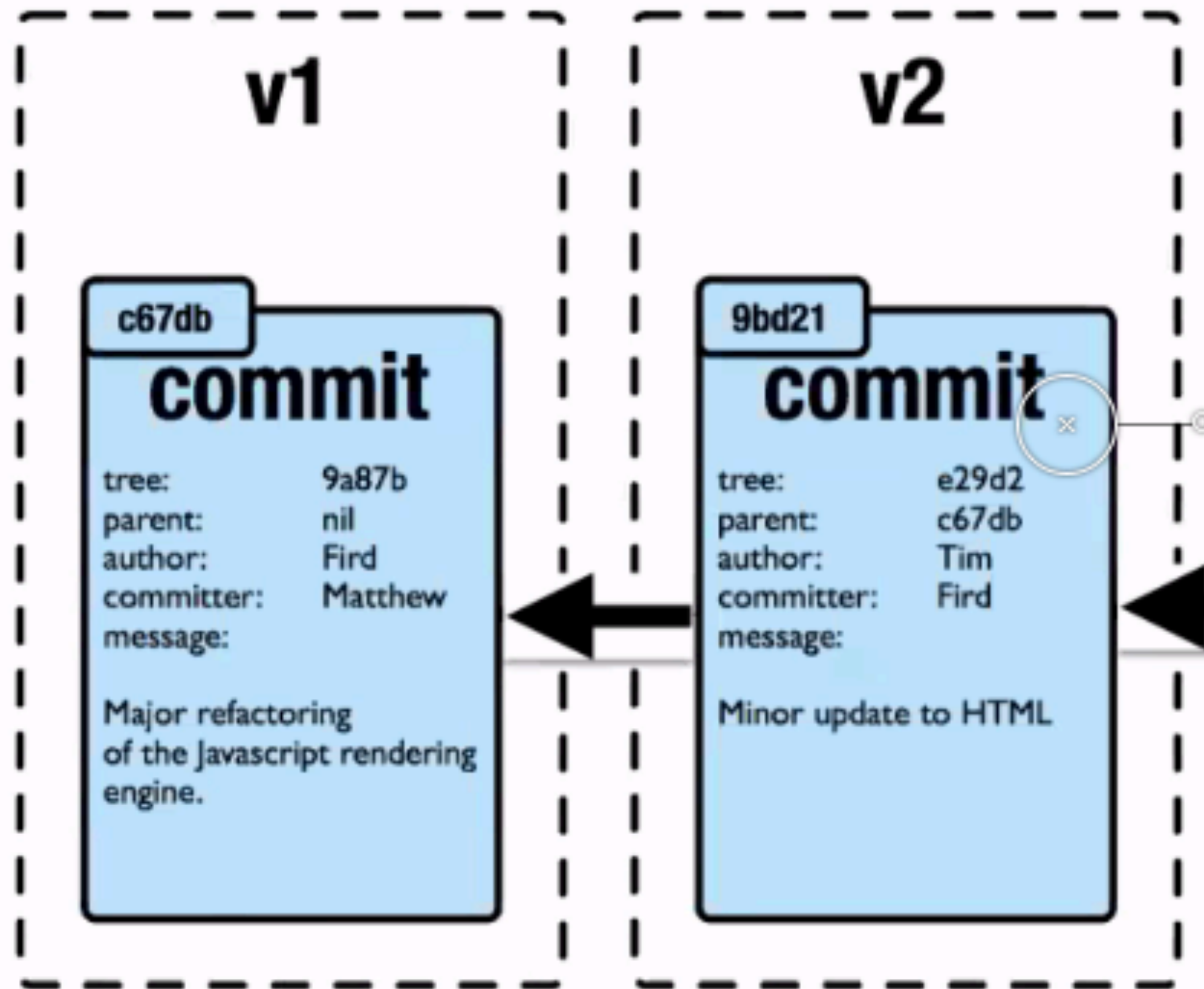


Now..
a bit of theory

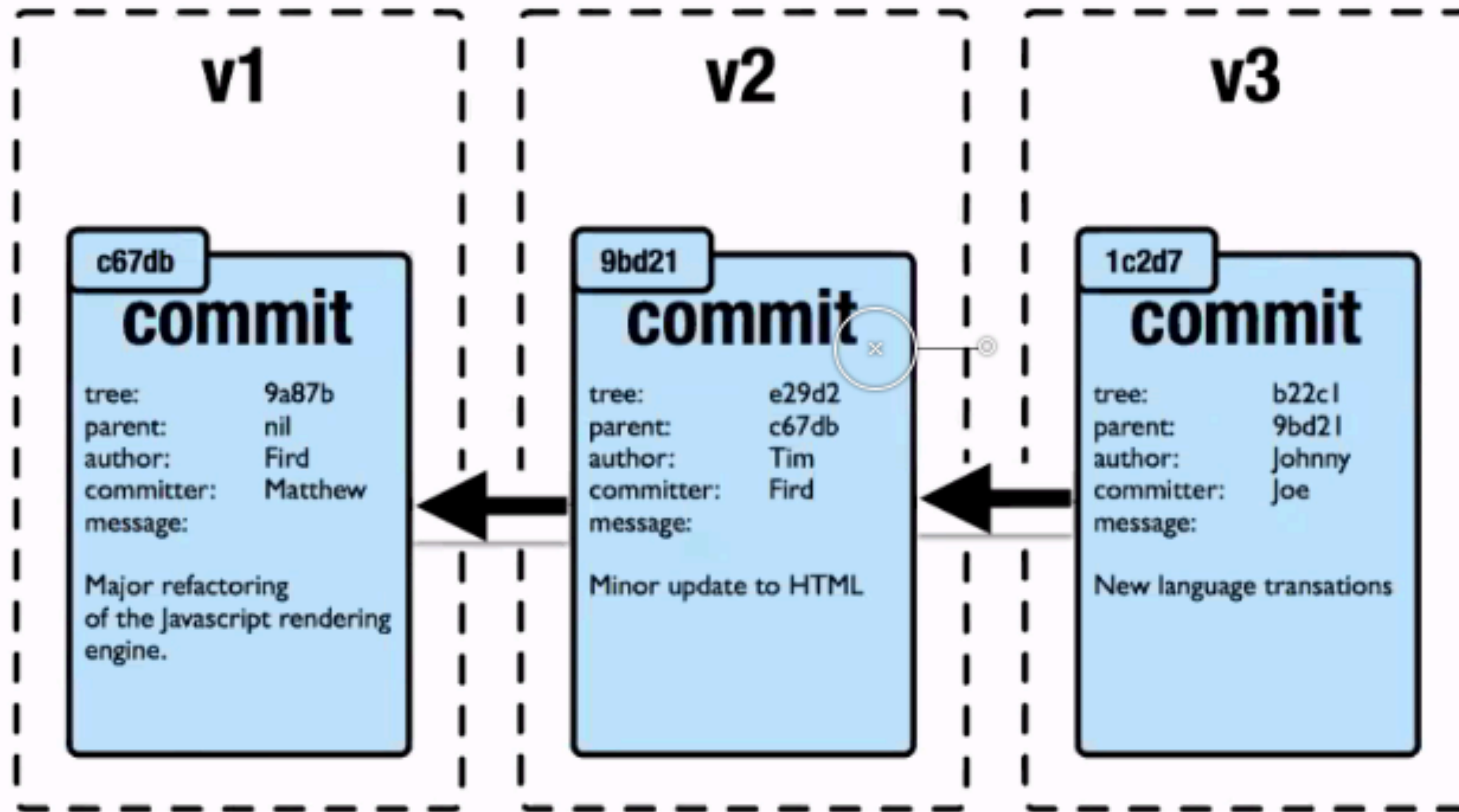
Inside a repository



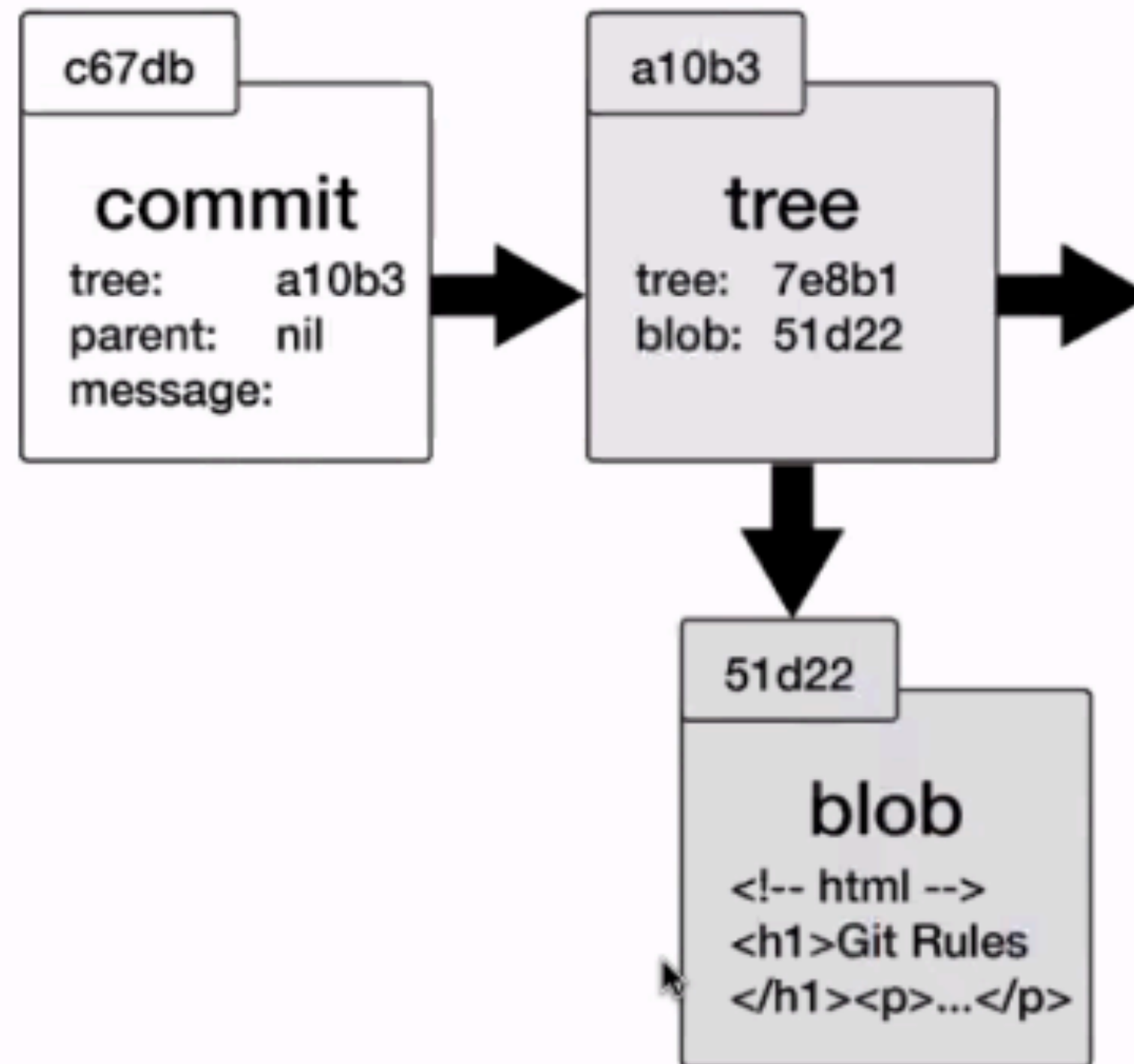
Inside a repository



Inside a repository



Inside the commit



It's a Merkle tree if that's your thing

