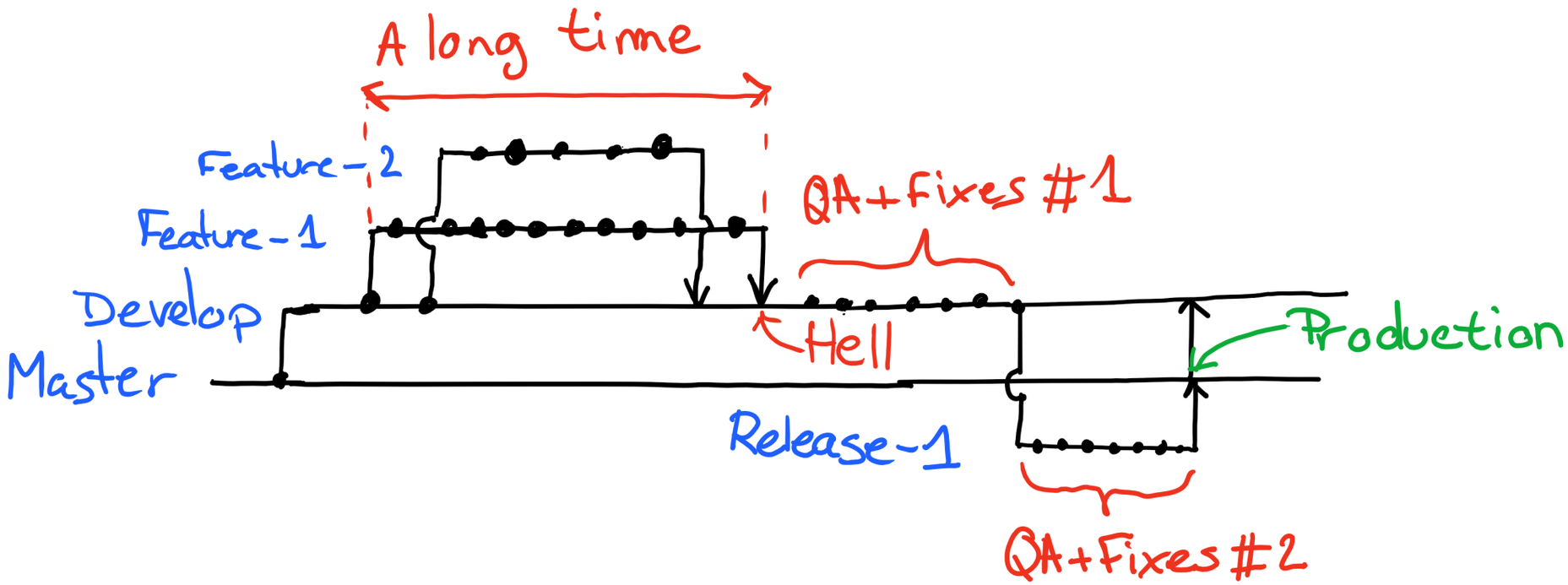


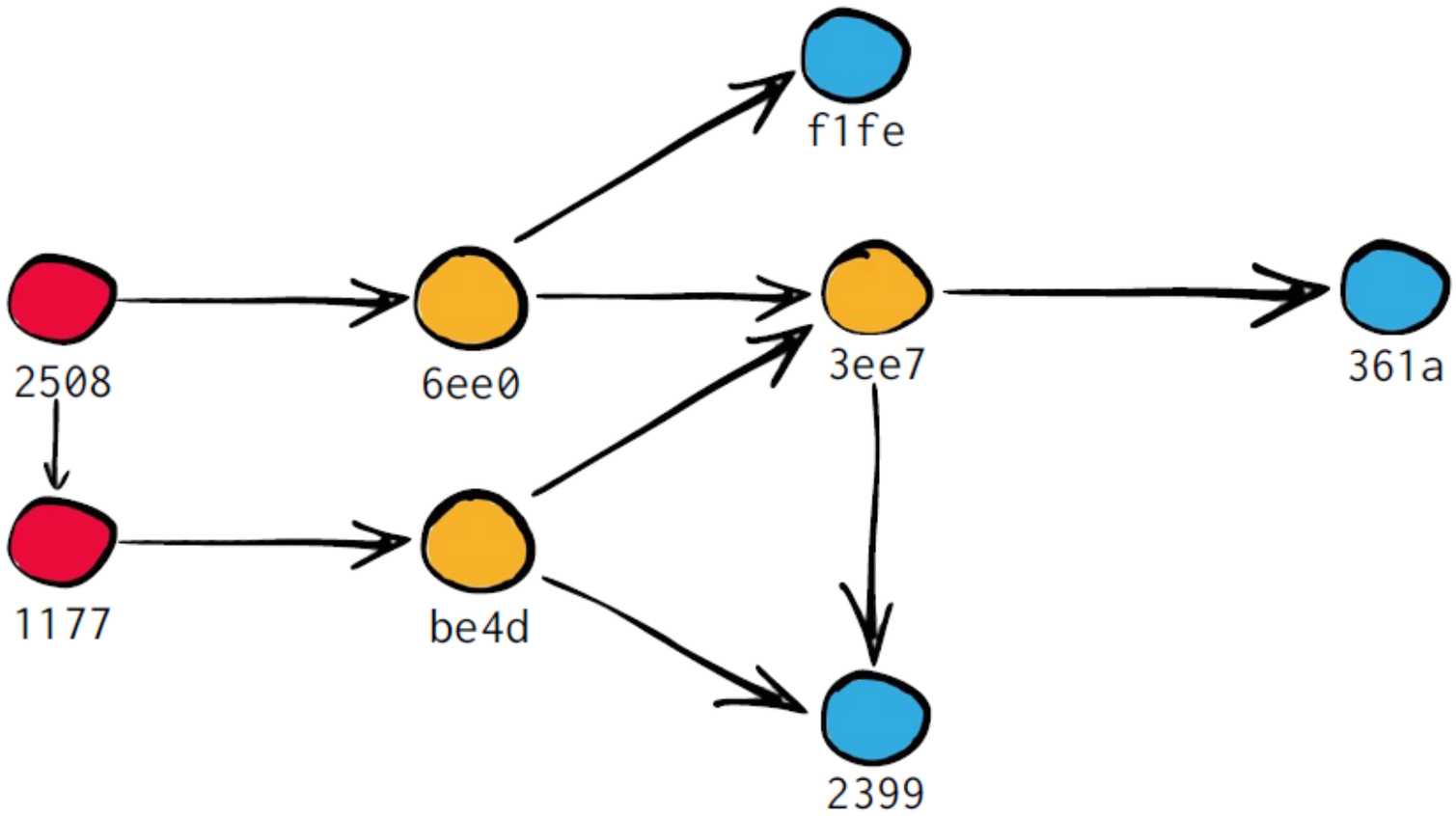
More about Git

Outline

- Reverting Changes
- Branching and Merging
- Conflict Resolution
- Inspecting History

But first…

A quick note



Reverting Changes

What can be reverted?

- **Commits** that are on *any* branch will not get lost.
- **Files** which were added and later removed can always be recovered.
- In Git we can *modify*, *reorder*, *squash*, and *remove* **commits** and also these actions can be undone.
- *Some commands* can **permanently** delete uncommitted changes. When in, **doubt always commit first.**

git revert

```
$ git log --oneline

f960dd3 (HEAD -> master) not sure this is a good idea
dd4472c we should not forget to enjoy
2bb9bb4 add half an onion
2d79e7e adding ingredients and instructions
```

- We examine our git history.

- We realize with commit **`f960dd3`** we made a mistake and want to revert the changes.

- A safe way to revert this commit is by running

  - `git revert f960dd3`

- This creates a new commit that does the opposite of the reverted commit. The old commit remains in the history.

Make a Commit

```
$ git log --oneline

f960dd3 (HEAD -> master) not sure this is a good idea
dd4472c we should not forget to enjoy
2bb9bb4 add half an onion
2d79e7e adding ingredients and instructions
```

Revert your Commit

```
$ git revert f960dd3
```

Check the History

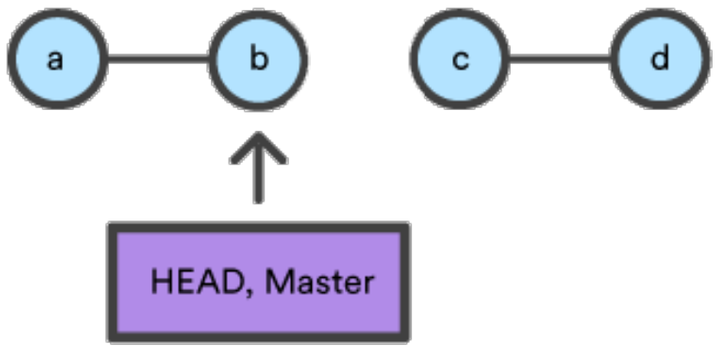
```
$ git log --oneline

d62ad3e (HEAD -> master) Revert "not sure this is a good idea"
f960dd3 not sure this is a good idea
dd4472c we should not forget to enjoy
2bb9bb4 add half an onion
2d79e7e adding ingredients and instructions
```

Adding to the previous commit

- Sometimes we commit but realize we forgot something.
- We can amend to the last commit:
  - `git commit --amend`
  - `git commit -am “My amended message.”`
- This can also be used to modify the last commit message.
- This will change the commit hash.

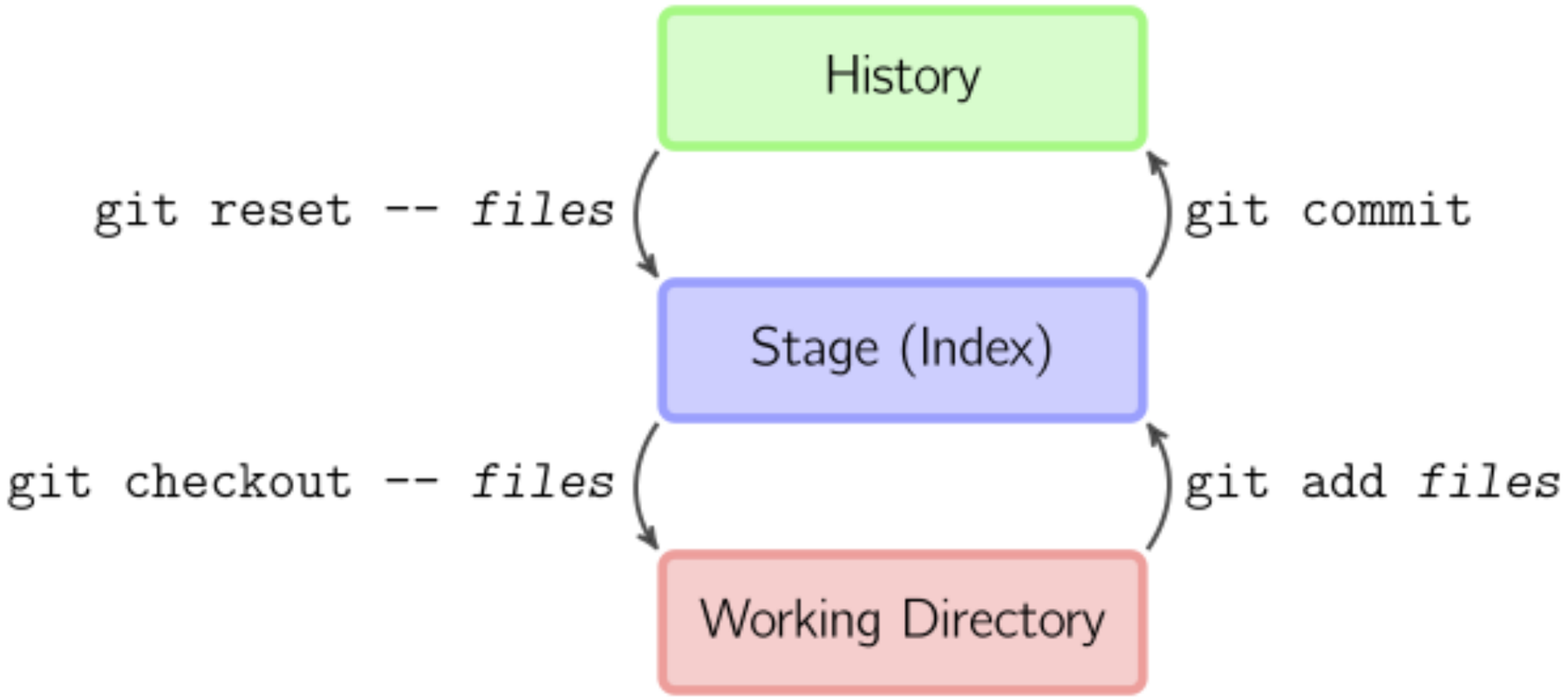
Clean the history

- We can also completely remove commits from the working tree by rewriting the history.
  - `git reset --hard dd4472c`
- **Use this command with caution!**
  This may result in data loss.
- Example from before, continued.

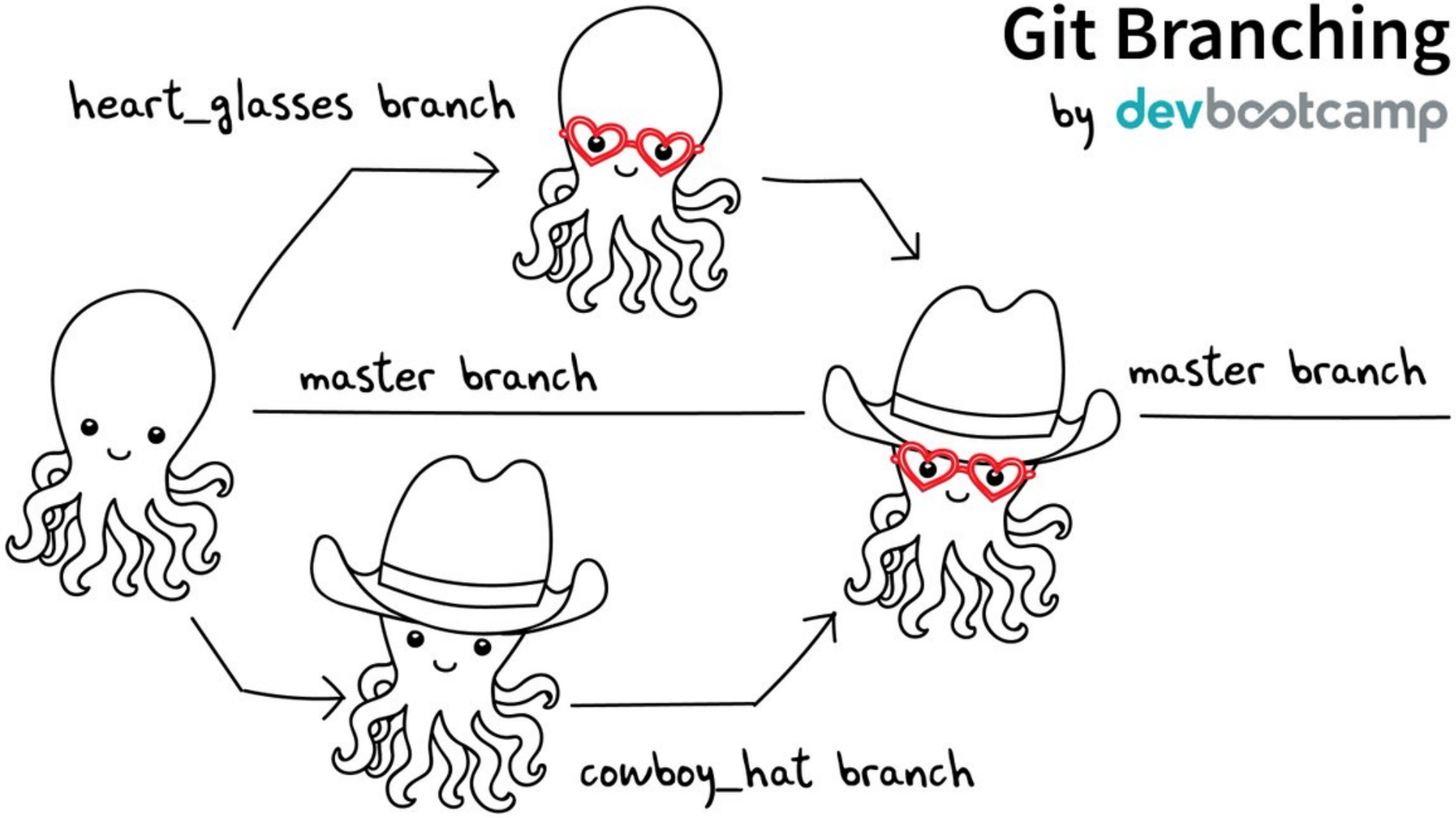


Reset History

```
$ git log --oneline

d62ad3e (HEAD -> master) Revert "not sure this is a good idea"
f960dd3 not sure this is a good idea
dd4472c we should not forget to enjoy
2bb9bb4 add half an onion
2d79e7e adding ingredients and instructions

$ git reset --hard dd4472c

HEAD is now at dd4472c we should not forget to enjoy

$ git log --oneline

dd4472c (HEAD -> master) we should not forget to enjoy
2bb9bb4 add half an onion
2d79e7e adding ingredients and instructions
```



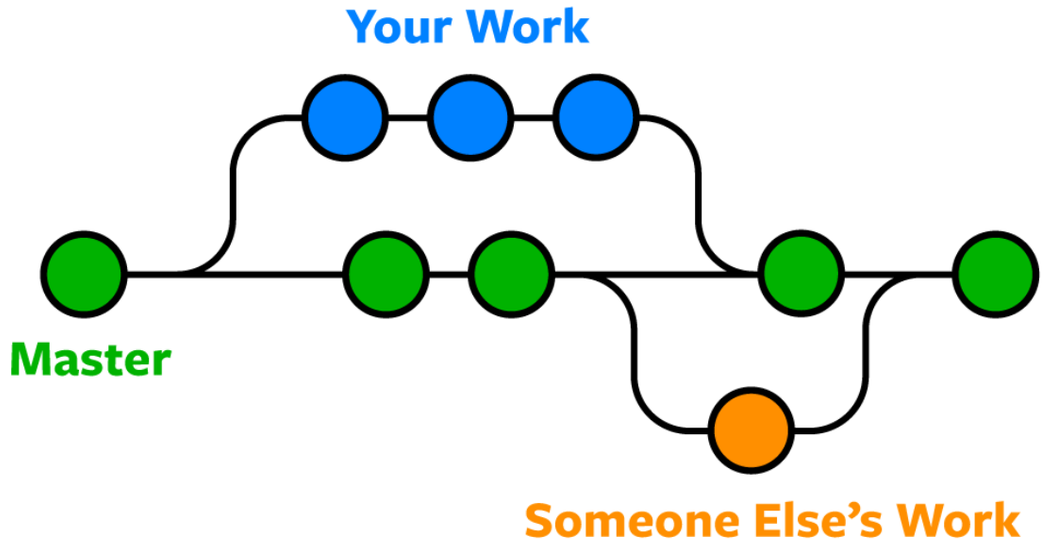
States Summary



Branching and Merging

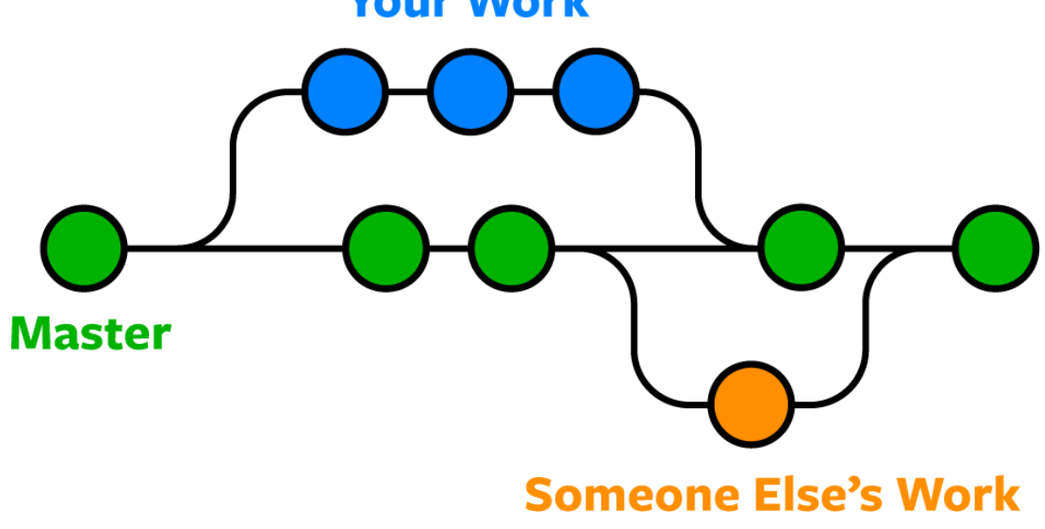
What is a branch?

- Branches are just like labels that point to a commit.
  - **Commits** are depicted here as little circles.
  - **Branches** have different colours.
- When we talk about branches we often mean all parent commits, not only the commit pointed to.



Motivation for Branches

- Software development is often not linear.
- We need separate different lines of work.
- We typically need at least one stable version of the code.
- Other than this convention there is nothing special about **master**, it is just a branch.



Where is `git HEAD`?

```
$ git branch

* master
```

- To see where we are (where HEAD points to), we use
  - `git branch`
- This command shows where we are, it does **not** create a branch.
- There is only **master** and we are on **master** (star represents the HEAD).

Creating a Branch

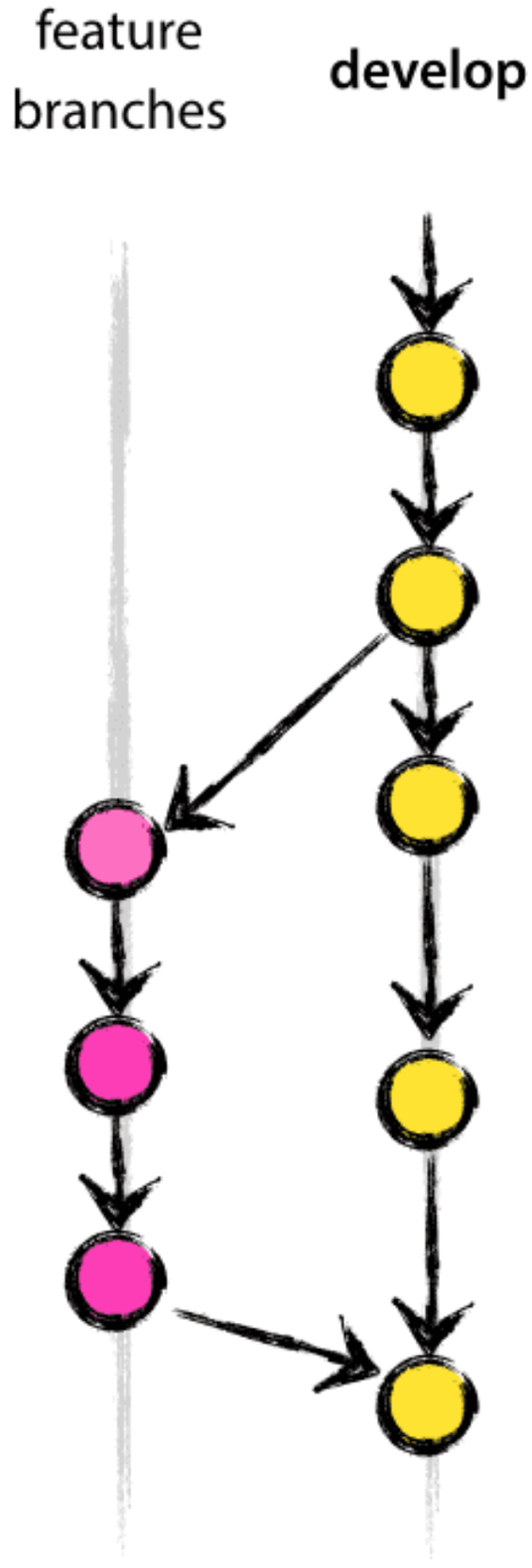
```
$ git branch experiment    # create branch called "experiment" pointing to the present c
ommit
$ git checkout experiment  # switch to branch "experiment"
$ git branch               # list all local branches and show on which branch we are
```

- We can verify that a new branch **`experiment`** was created using:
  - `git branch experiment`
- We can verify that `HEAD` was switched to **`experiment`** using:
  - `git checkout experiment`

```
$ git branch

* experiment
  master
```

Or the faster way…

```
git checkout -b
   new_branch
```

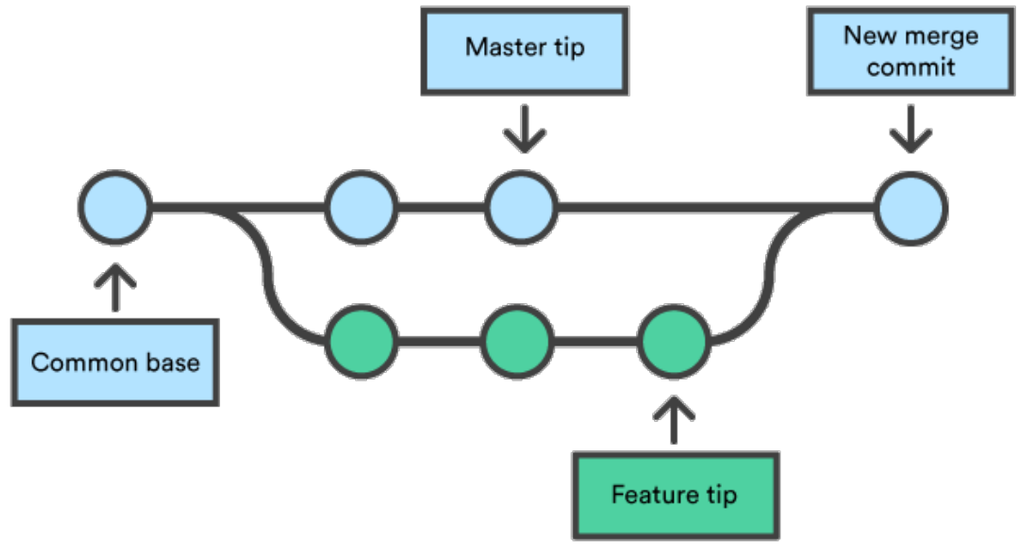


What is `git checkout`?

- `git checkout <branchname>`
  - Switch to a branch.
- `git checkout <hash>`
  - Switch to a specific commit.
- `git checkout <path/file>`
  - Set a file/path to a specific state.
  - Throws away all unstaged/ uncommitted changes.

Merging

- Now that we are happy with our changes, we can merge our changes back into the branch we diverged from.
- We can use
  - `git merge <branch>`

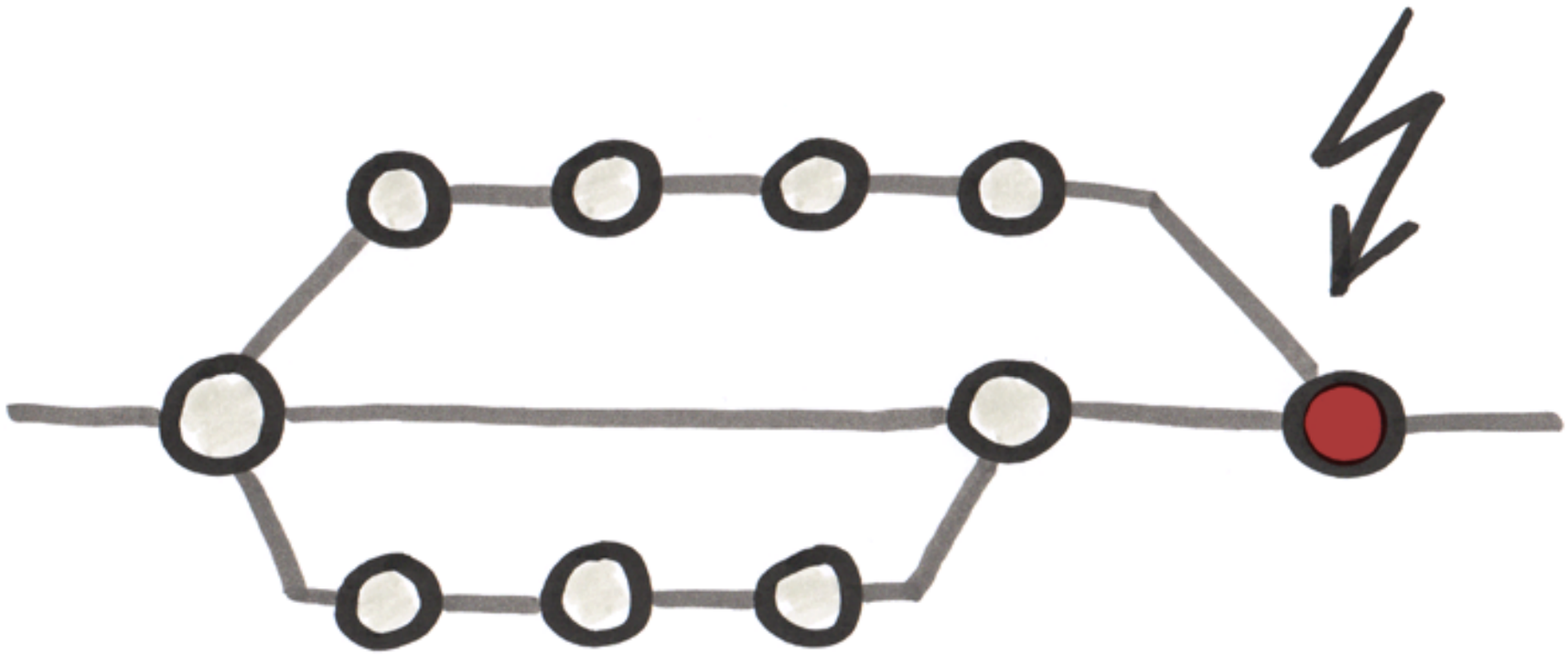


Checkout the branch you want to merge into

```
$ git branch

  experiment
  less-salt
* master
```

Then we merge `experiment` into `master`:

```
$ git merge experiment
```

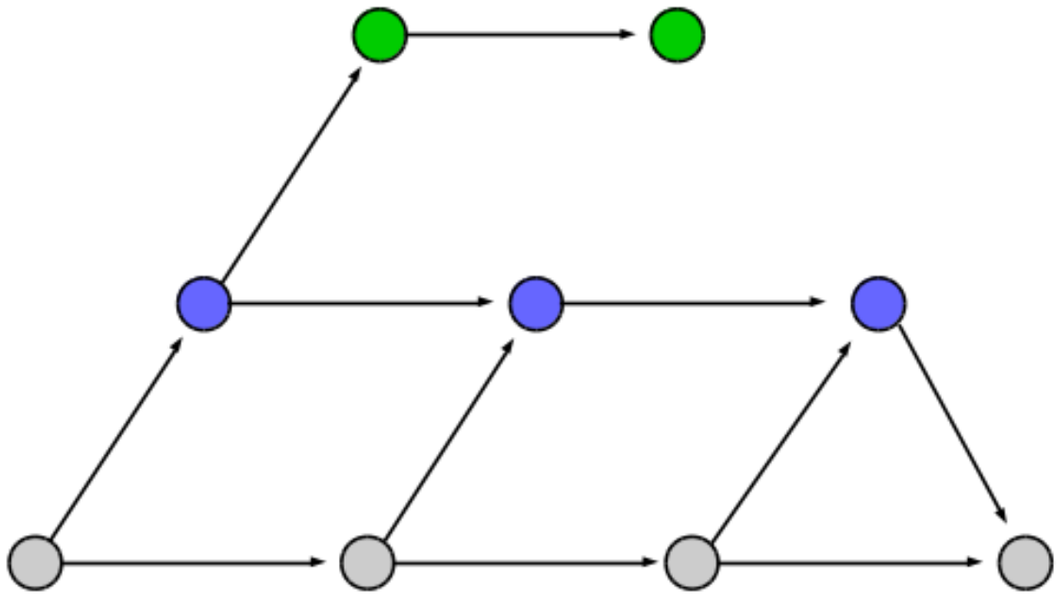


Conflict Resolution

Inspecting History

Getting Help

- If you need help, you can use **`git help [command]`**.
  - **`git help checkout`**
  - **`git help branch`**
  - **`git help reset`**
- Use online resources.
  - https://guides.github.com/

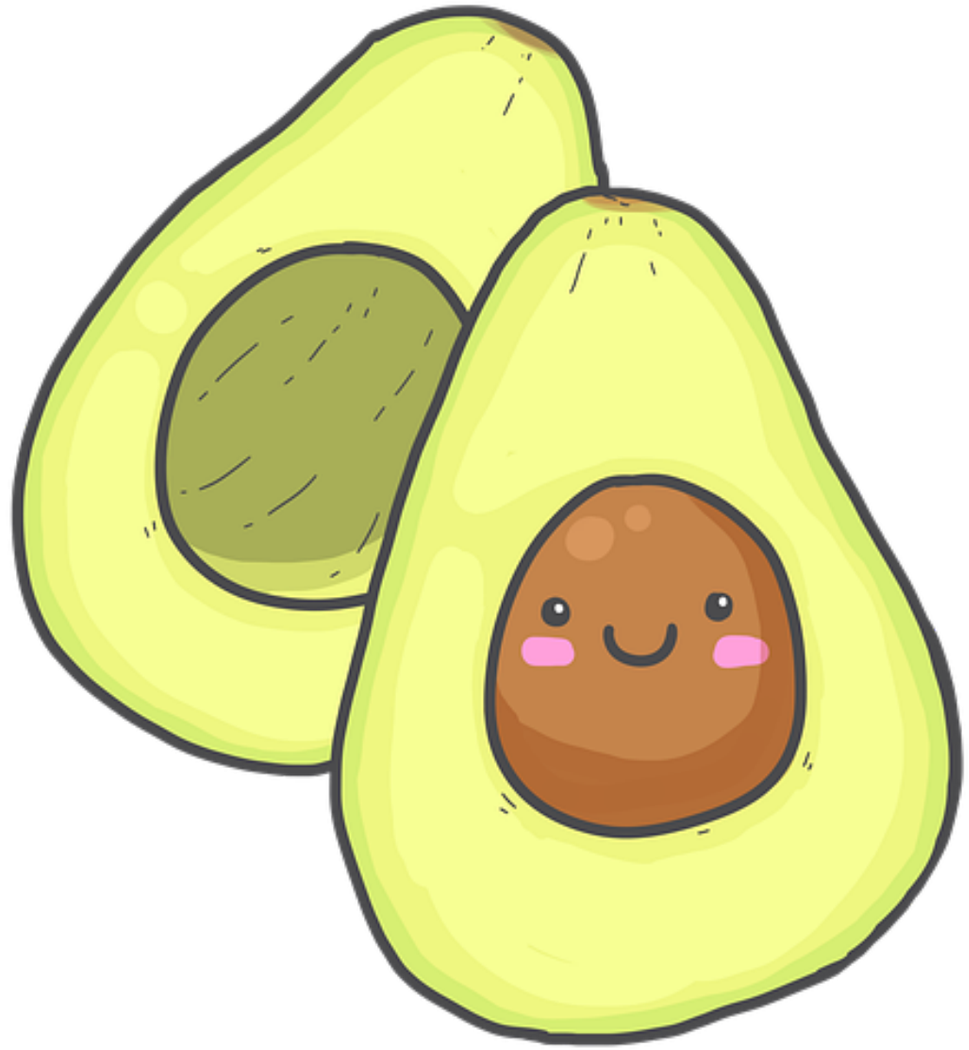


To-Do Task

Complete all lessons on https://learngitbranching.js.org/

To-Do Task

Tracking a guacamole recipe with Git



How to NOT use Git?