UPskill

Digital Skills & Jobs

Git – Branching

Metodologias de Trabalho em Equipa

P.PORTO

apdc
ASSOCIAÇÃO PORTUGUESA PARA O DESENVOLVIMENTO DAS COMUNICAÇÕES

INSTITUTO DO EMPREGO E FORMAÇÃO PROFISSIONAL

CONSELHO COORDENADOR DOS INSTITUTOS SUPERIORES POLITÉCNICOS

What is a branch?

- Branches allow to diverge from your current development and try something new without altering the history of your main work.
- For example, you could implement a new code feature whilst leaving the fully functional (hopefully working and tested) code intact for others to checkout.
- It is a fantastic way to test ideas, try new things and safely develop your repository.

Creating branches

- A new repositories, by default, start on a branch called *master*.
- To create a branch called *testing* use:

```
$ git branch testing
```

- To start working with new branch:

```
$ git switch testing
Switched to branch 'Testing'
```

Creating branches

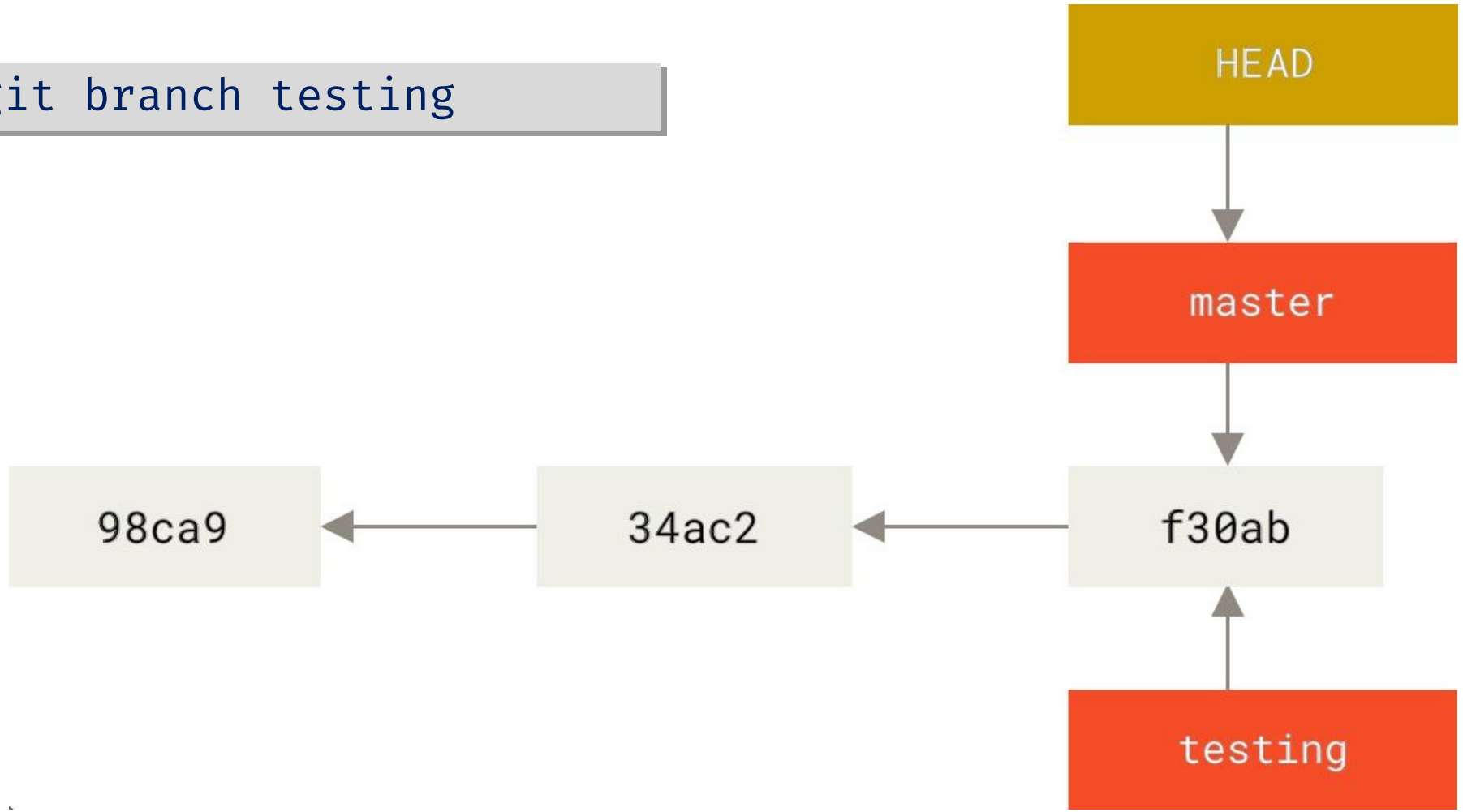
- To create a new branch and switch to it at the same time you can use the `git switch` command with the `-c` switch:

```
$ git switch -c testing
```

- In the past, the same steps were performed using the `git checkout` command, but after version 2.23, this command was split into two seperate pieces (switch and restore).

Create a new branch

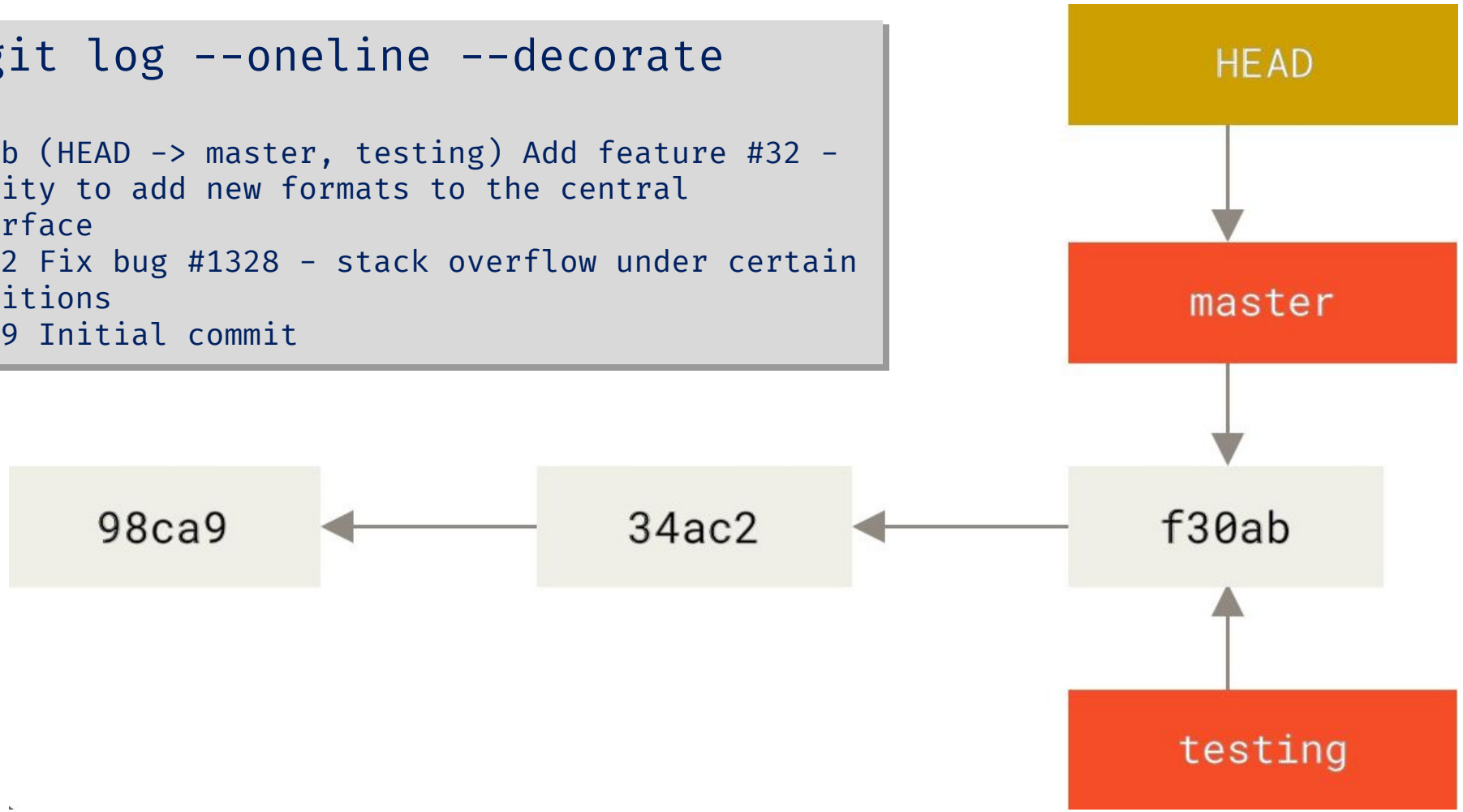
```
$ git branch testing
```



How to visualize the existing pointers

```
$ git log --oneline --decorate

f30ab (HEAD -> master, testing) Add feature #32 -
ability to add new formats to the central
interface
34ac2 Fix bug #1328 - stack overflow under certain
conditions
98ca9 Initial commit
```

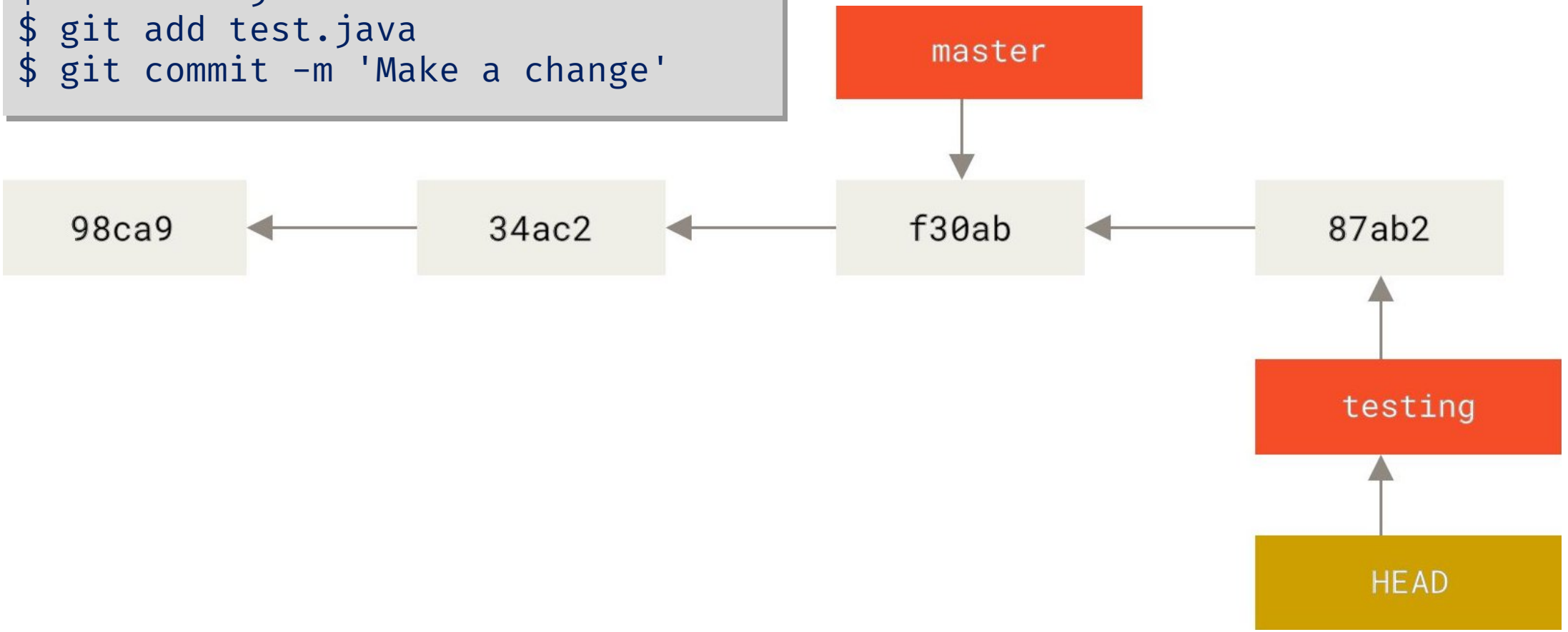


Switch to the new branch

```
$ git switch testing
```

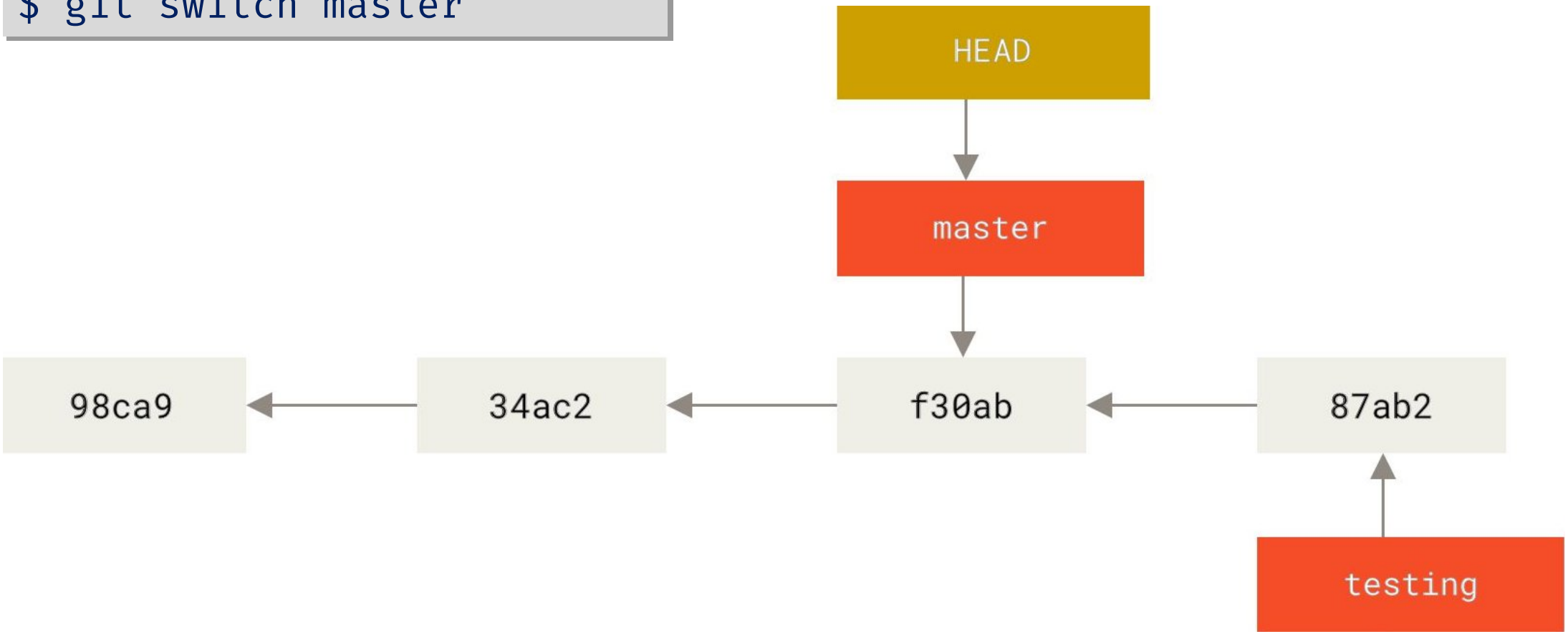
Commit to the new branch

```
$ vim test.java
$ git add test.java
$ git commit -m 'Make a change'
```



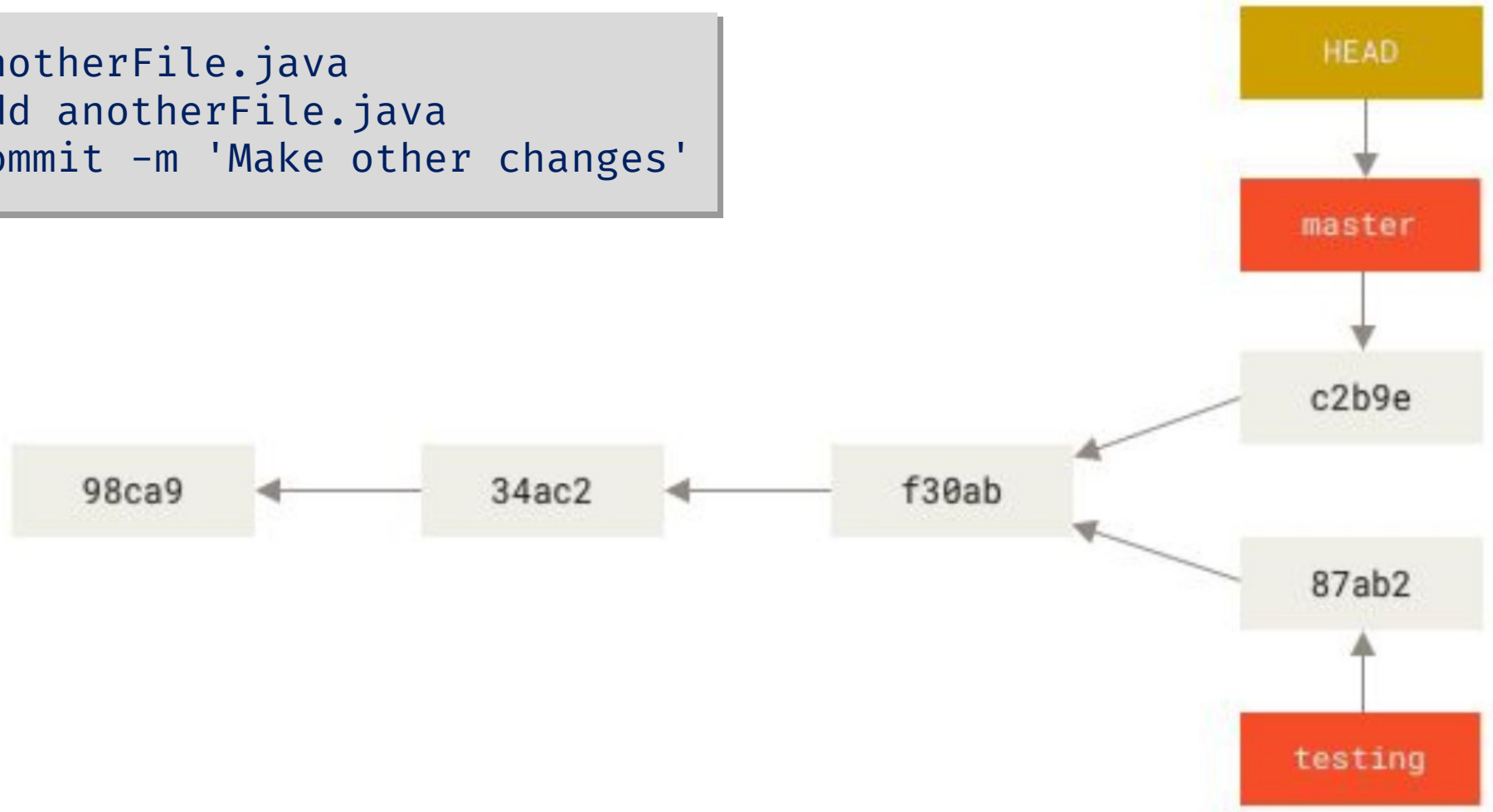
Switch to the master branch

```
$ git switch master
```



Switch to the master branch

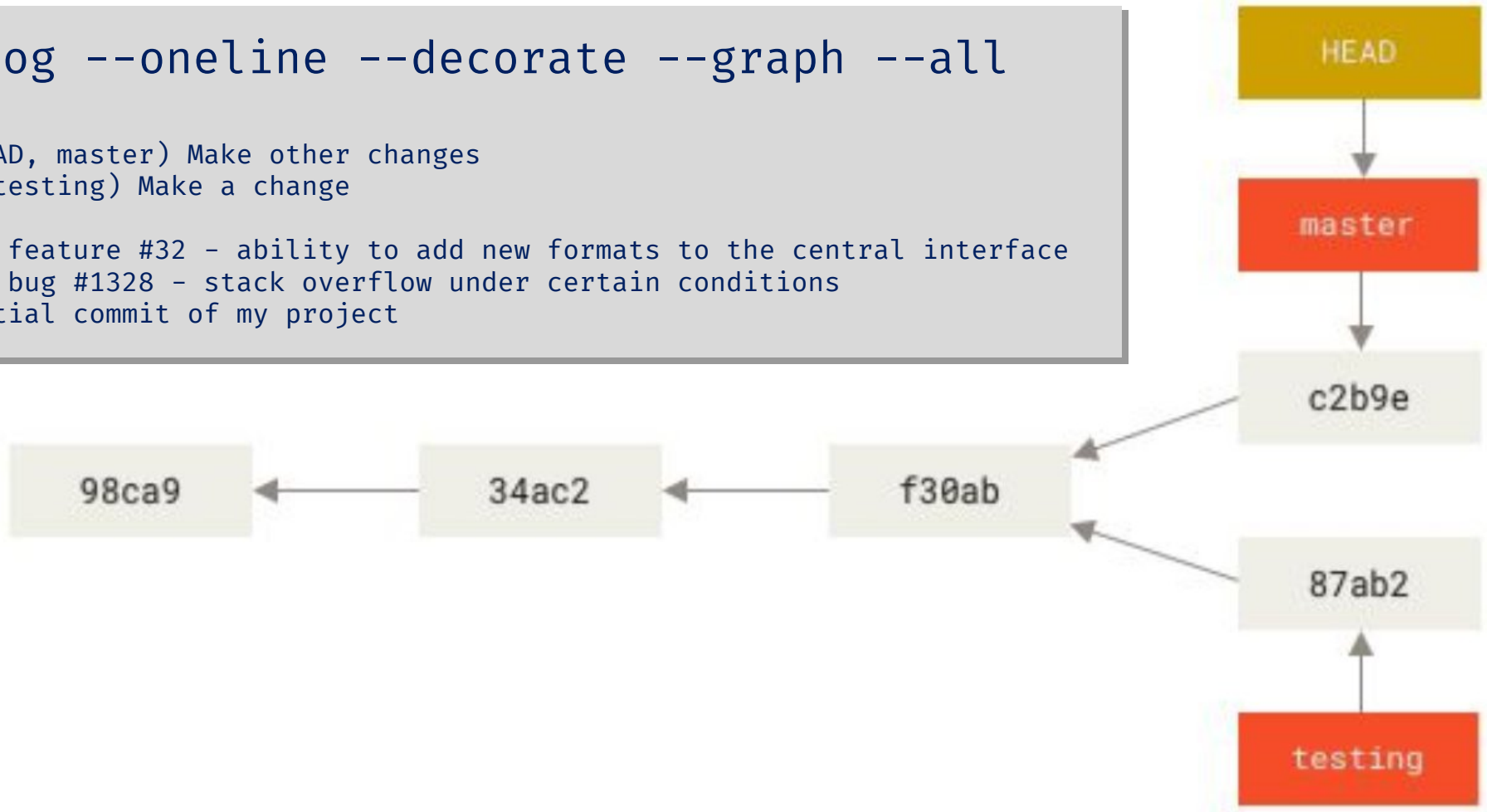
```
$ vim anotherFile.java
$ git add anotherFile.java
$ git commit -m 'Make other changes'
```



Visualize history of commits with branching

```
$ git log --oneline --decorate --graph --all

* c2b9e (HEAD, master) Make other changes
| * 87ab2 (testing) Make a change
|/
* f30ab Add feature #32 - ability to add new formats to the central interface
* 34ac2 Fix bug #1328 - stack overflow under certain conditions
* 98ca9 Initial commit of my project
```



Merging

- At some stage we will want fold our changes in the *testing* branch back into the *master* branch.
- We do this by merging the *testing* branch into *master*.

```
$ git switch master
$ git merge testing
```

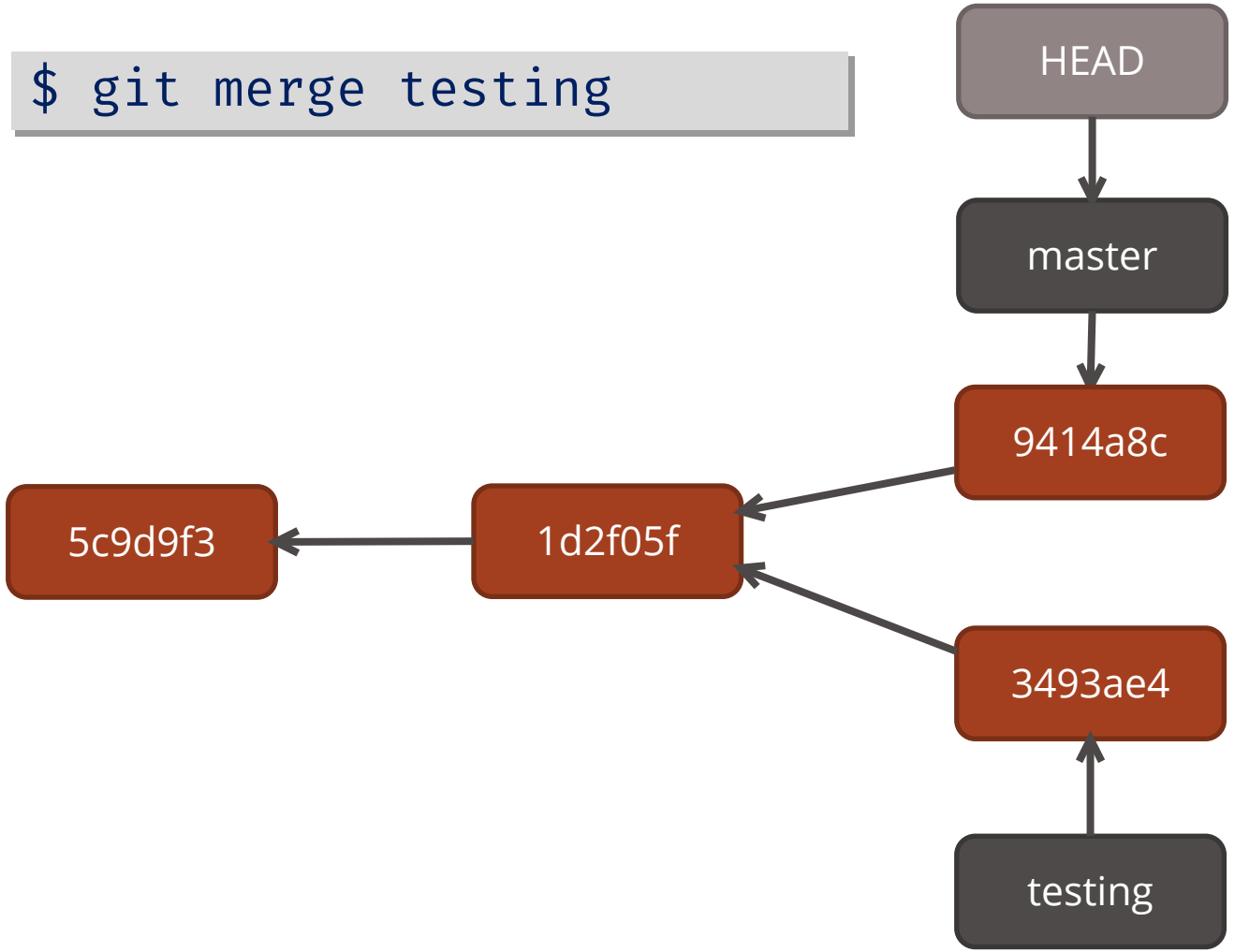
- To delete a no longer needed branch use:

```
$ git branch –d testing
```

How git merge works

Before

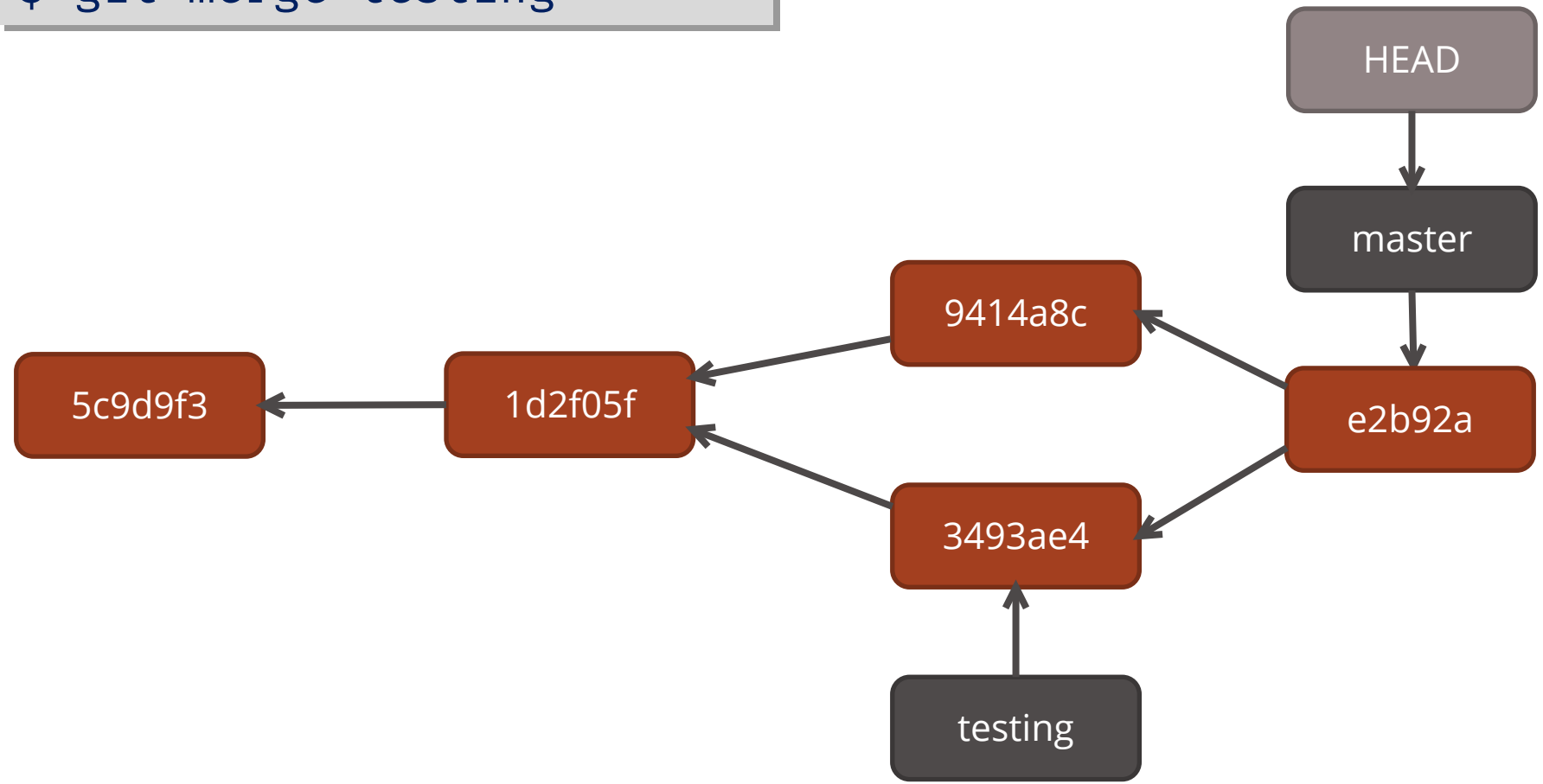
```
$ git merge testing
```



How git merge works

After

```
$ git merge testing
```



Conflicts

- A version control system lets multiple users simultaneously edit their own copies of a project.
- Usually, the version control system can merge simultaneous changes by two different users:
  - for each line, the final version is:
    - the original version if neither user edited it;
    - is the edited version if one of the users edited it.

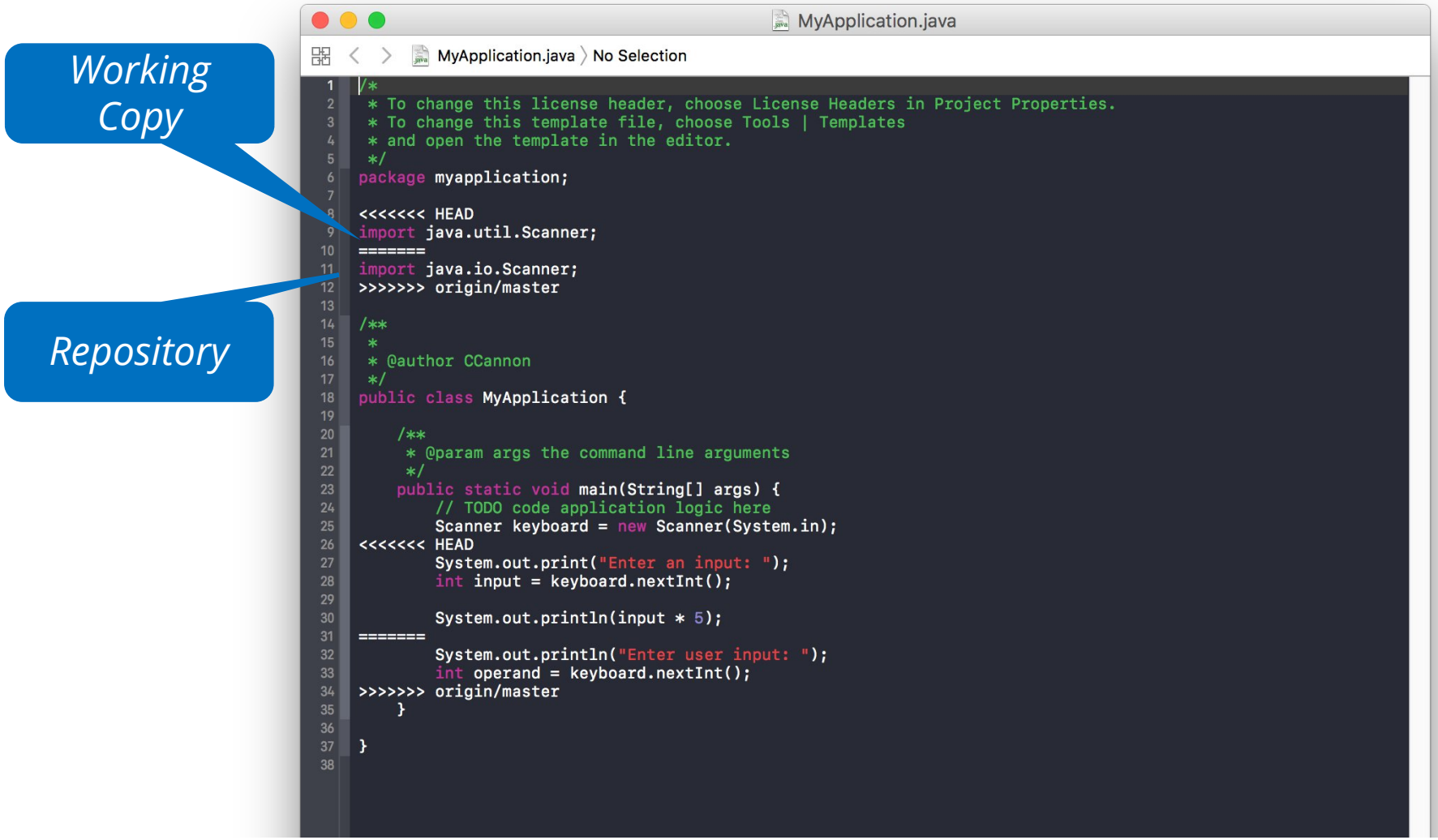
Conflicts


1 lime
2 avocados
1 cilantro
2 salt
2 avocados
2 cilantro
2 salt
1 lime
2 avocados
????????????
2 salt
4 onion
2 avocados
3 cilantro
2 salt
4 onion

Conflicts

- A conflict occurs when two different users make simultaneous, different changes to the same line of a file.
- In this case, the version control system cannot automatically decide which of the two edits to use (or a combination of them, or neither!).
- Manual intervention is required to resolve the conflict.

Git conflicts

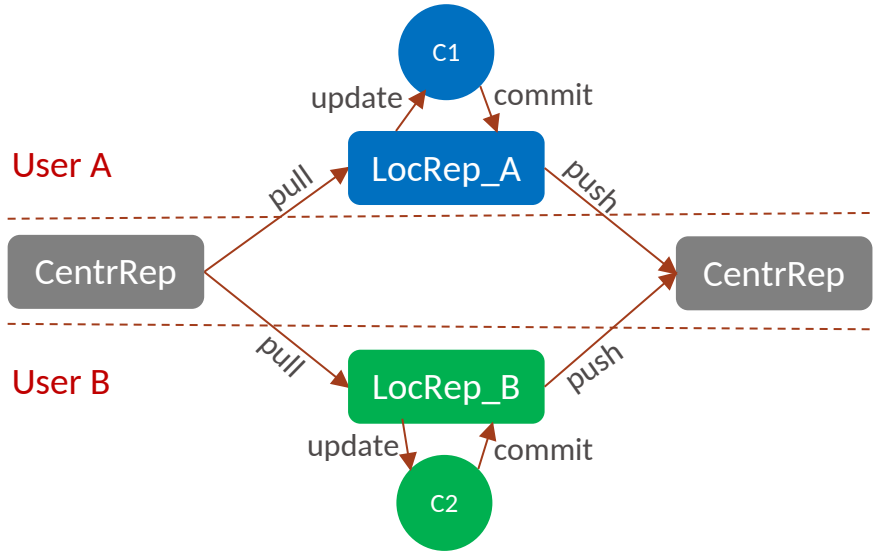


Conflicts

- Merge operation combines simultaneous edits by different users.
- Sometimes merge completes automatically, but if there is a conflict, merge requests help from the user by running a merge tool.
- In centralized version control, merging happens implicitly every time you do update

Conflicts

- Change1 (C1) and Change2 (C2) are considered simultaneous if:
  - User A makes C1 before User A does an update that brings C2 into User A's working copy;
  - User B makes C2 before User B does an update that brings C1 into User B's working copy.



Solve conflicts

- If a file has conflictual information when merging (e.g. when the same variable has different values attributed), Git won't be able to merge them cleanly.

```
$ git switch master
$ git merge testing

CONFLICT (content): Merge conflict in
conflictFile.java
Automatic merge failed; fix conflicts and
then commit the result.
```

Check status

```
$ git status

On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
        both modified:   conflictFile.java
```

conflictFile.java

```
<<<<<<< HEAD
count = 12
=======
count = 23
>>>>>>> testing

( ... )
```

Proceed with the merge

```
$ git add conflictFile.java
$ git merge --continue

[master 991c091] Merge branch 'testing'
```

Remove testing branch if no longer needed

```
$ git branch -d testing

Deleted branch testing (was f29e067).
```

Rebasing

- In Git, there are two main ways to integrate changes from one branch into another: the merge and the rebase
- With the rebase command, you can take all the changes that were committed on one branch and replay them on a different branch

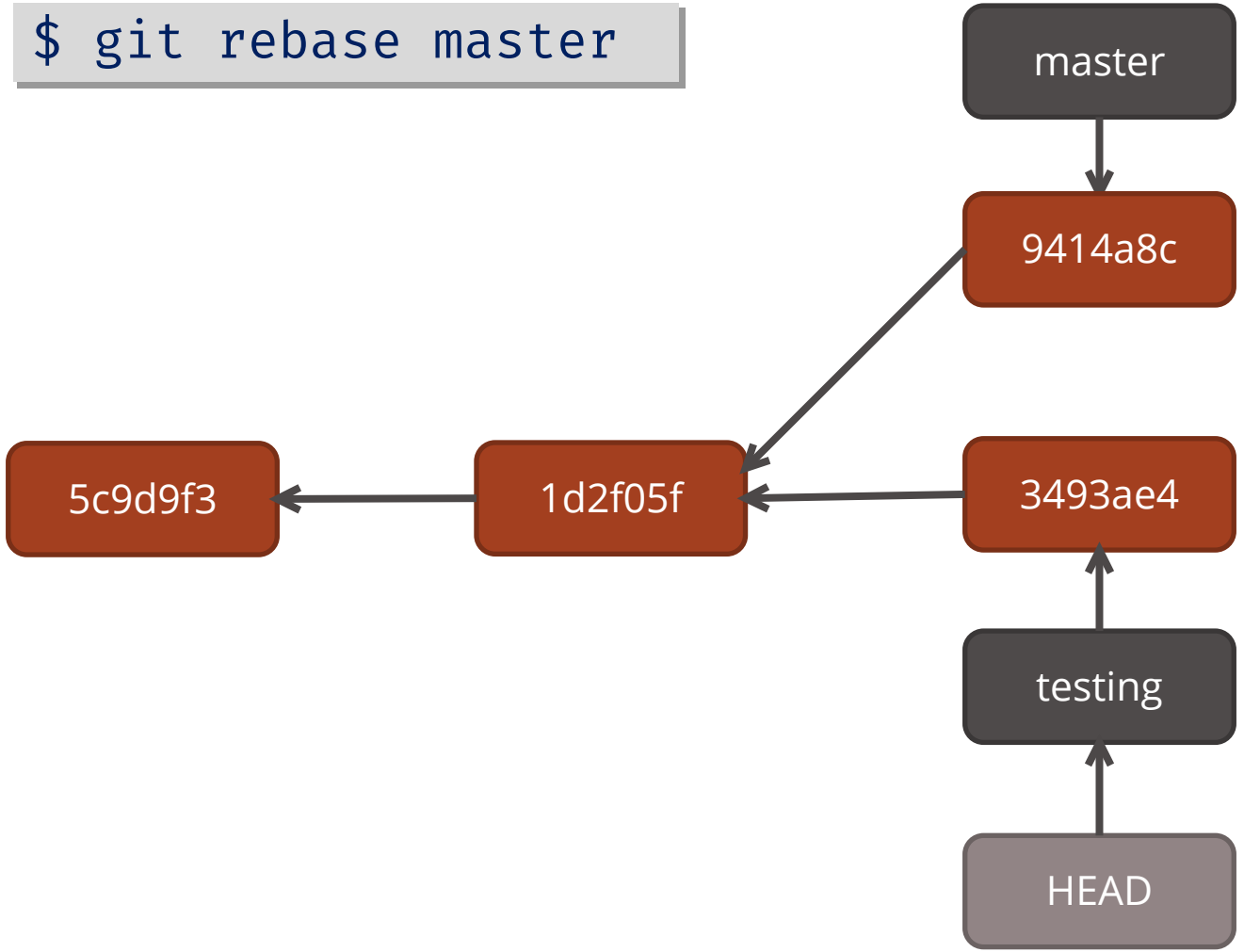
```
$ git switch master
Switched to branch 'master'

$ git rebase master
Successfully rebased and updated
refs/heads/master.
```

How git rebase works

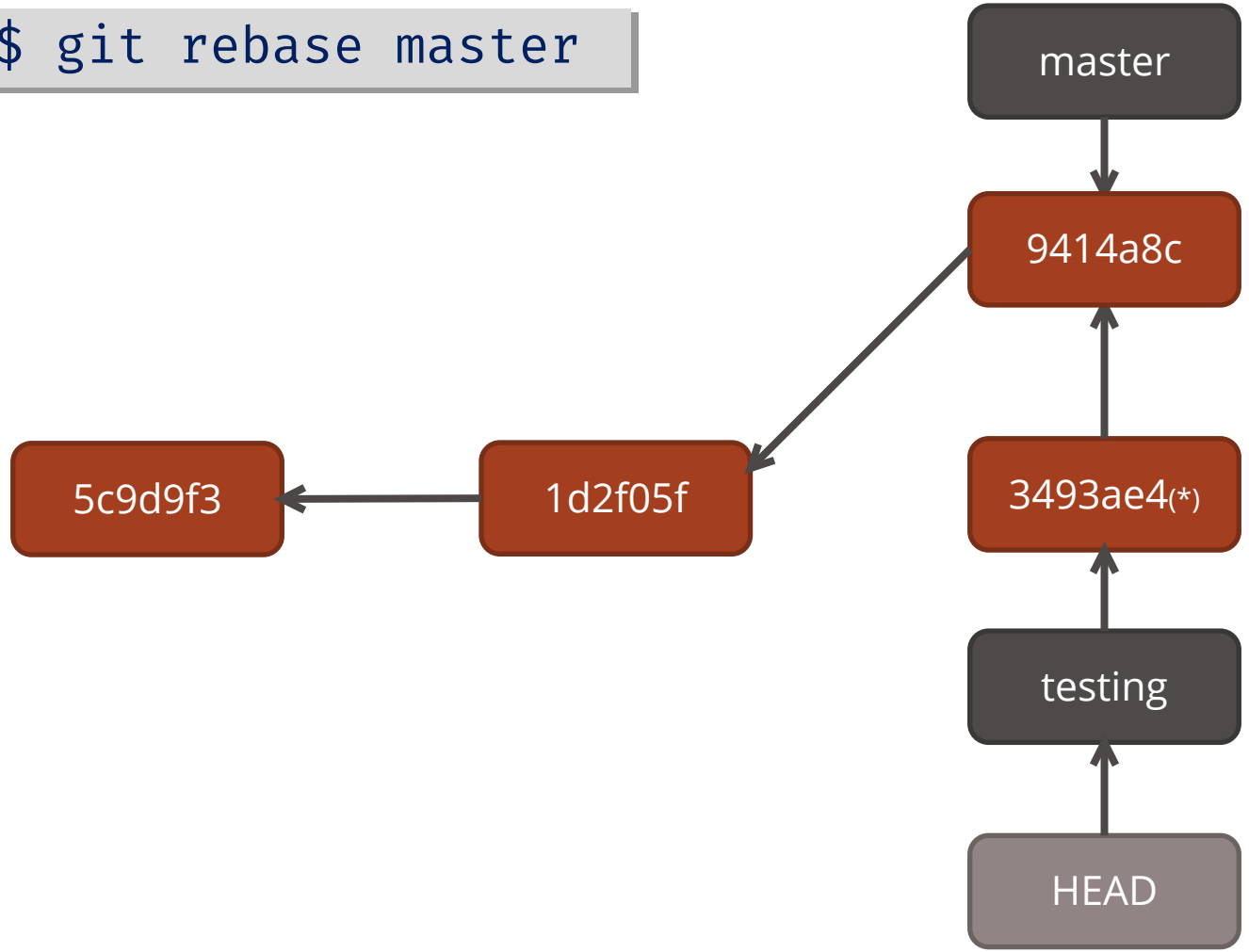
Before

```
$ git rebase master
```



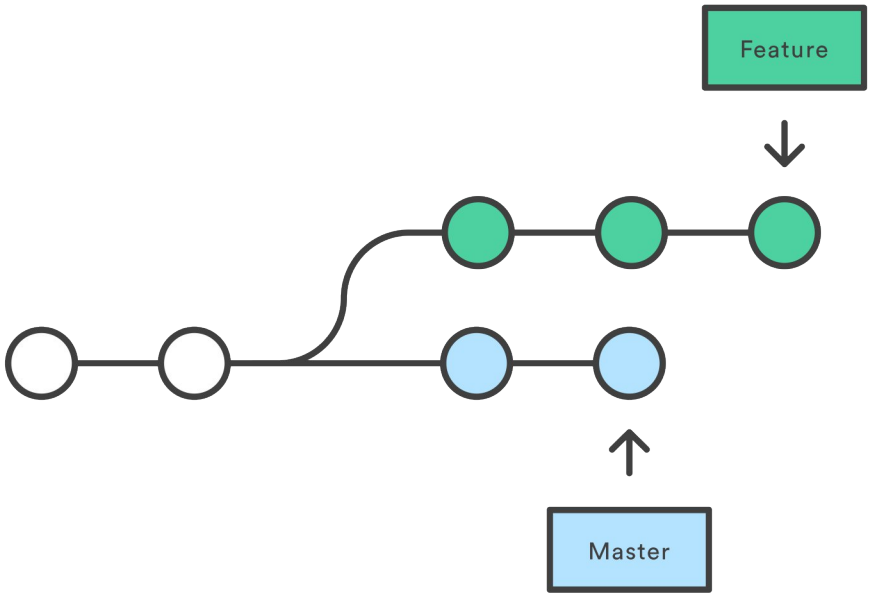
How git rebase works

After `$ git rebase master`

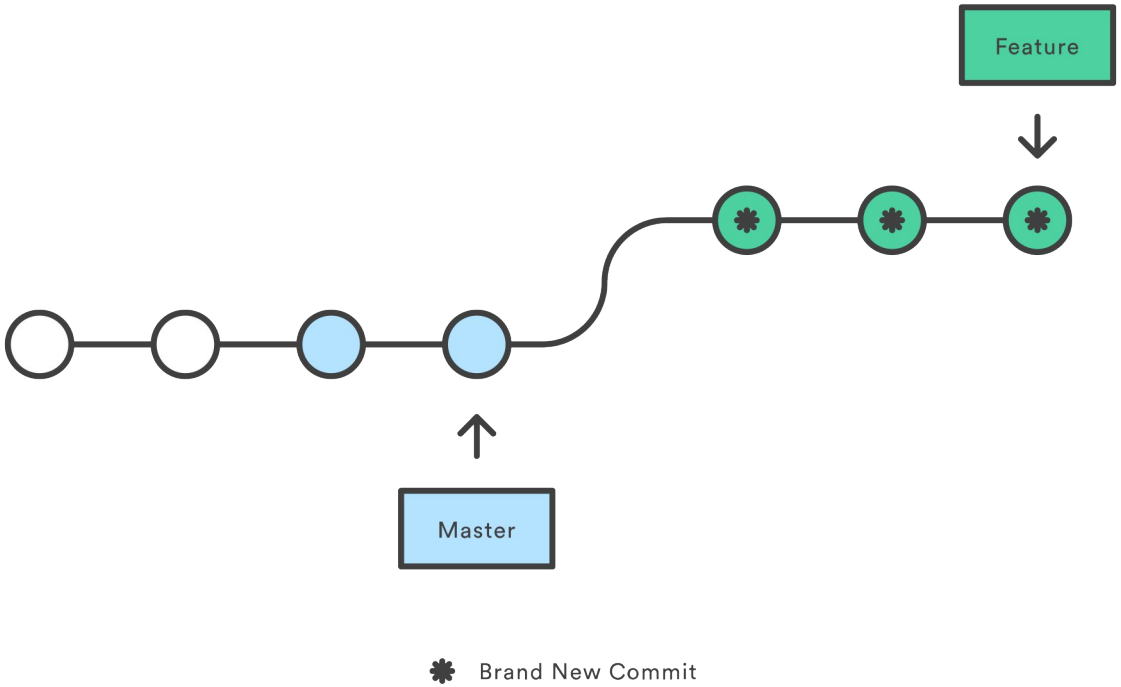


How git rebase works

A forked commit history



Rebasing the feature branch onto master




```
$ git rebase master
```

Merge vs Rebase

- Merge maintains an historical view of the repository's commit history, and a record of what actually happened.
- Rebase allows to tell the story of how your project was made, and tell a more coherent story of how to get from A to B.
  - Git is a powerful tool, and allows you to do many things to and with your history, but every team and every project is different.
  - It's up to you to decide which one is best for your particular situation.
  - You can get the best of both worlds: rebase local changes before pushing to clean up your work, but never rebase anything that you've pushed somewhere.

Branch Management

- If you run the `git branch` command with no arguments, you get a simple listing of your current branches (the * indicates the current branch that we have checked out)

```
$ git branch
* master
  testing
```

Branch Management

- To see the last commit made to each branch we can run `git branch -v`

```
$ git branch -v
* master  7a98805 Fix javascript issue
  testing 782fd34 Add scott to the author
list in the readme
```

Branch Management

- To rename a branch locally we can use `git branch –-move [previous name] [new name]`

```
$ git branch --move master main
```

  - this change is only local for now.

UPskill

Digital Skills & Jobs

Git – Branching

Metodologias de Trabalho em Equipa

P.PORTO

apdc
ASSOCIAÇÃO PORTUGUESA PARA O DESENVOLVIMENTO DAS COMUNICAÇÕES

INSTITUTO DO EMPREGO E FORMAÇÃO PROFISSIONAL

CONSELHO COORDENADOR DOS INSTITUTOS SUPERIORES POLITÉCNICOS