Lab 12: GIT: Version Control System

Introduction to Web Programming

AA 2017/2018

Stefano Chirico

What is Version Control? What is Git?

- **Version Control** is the management of the changes made within a system, that it has not to be software necessarily.
- Even if you have never used before a tool like Git, you will probably have ever carried out a version control. A very used and **bad** practice in software developing is, when the software has reached a stable situation, saving a local copy of it, identifying it as stable, and then following with the changes in other copy.

What is Version Control? What is Git?

- The **version control systems** (**VCS**) are designed for carrying out a proper management of the changes. These tools provide the following features:
  - **Reversibility.**
    The reversibility is the **main capability** of a VCS, allowing to return to any point of the history of the source code, for example, when a bug has been introduced and the code has to return to a stable point.
  - **Concurrency.**
    The concurrency allows to have **several people making changes on the same project**, facilitating the process of the integration of pieces of code developed by two or more developers.
  - **Annotation.**
    The annotation is the feature that allows to **add additional explanations and thoughts about the changes made**, such us a resume of the changes made, the reason that has caused these changes, an overall description of the stability, etc.

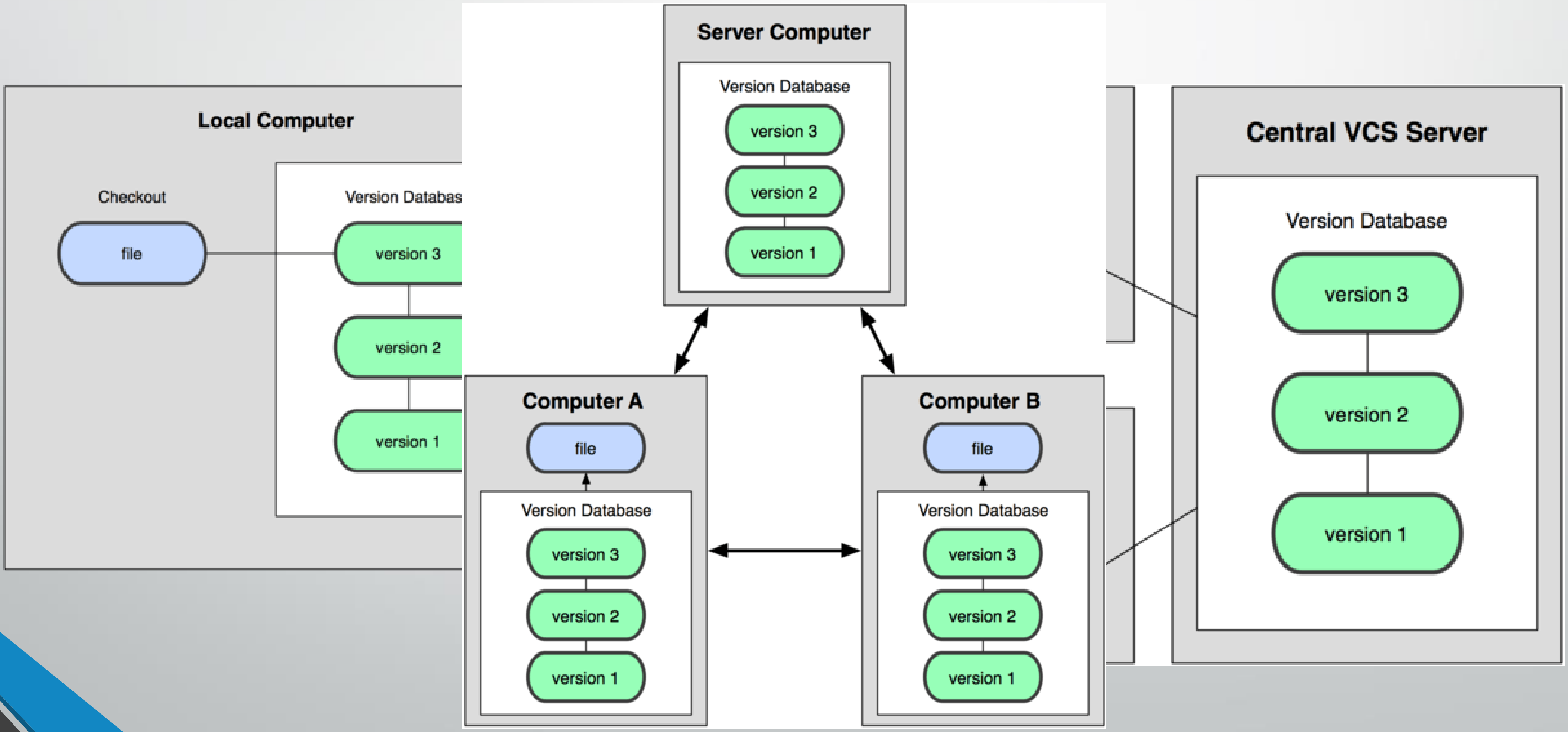
What is Version Control? What is Git?

With this, **the VCSs solve one of the most common problems of software development: the fear for changing the software**. You will be probably be familiar to the famous saying "if something works, don’t change it". Which is almost a joke, but, actually, is like we act many times. **A VCS will help you to get rid of being scared about changing your code**.

What is Version Control? What is Git?

- There are several models for the version control systems.
  - The one we mentioned, the **manual process**, can be considered as a **LVCS** (**local** version control system), since the changes are only saved locally.
  - Before the DVCSs burst into the version controlling world, the most popular VCS was, probably Apache Subversion (known also as **SVN**). This VCS was **centralized** (**CVCS**). A centralized VCS is a system designed to have a single full copy of the repository, hosted in some server, where the developers save the changes they made.
  - **Git** is a **distributed** version control system (**DVCS**), also known as **decentralized**. This means that every developer has a full copy of the repository, which is hosted in the cloud.

Local vs Centralized vs Distributed



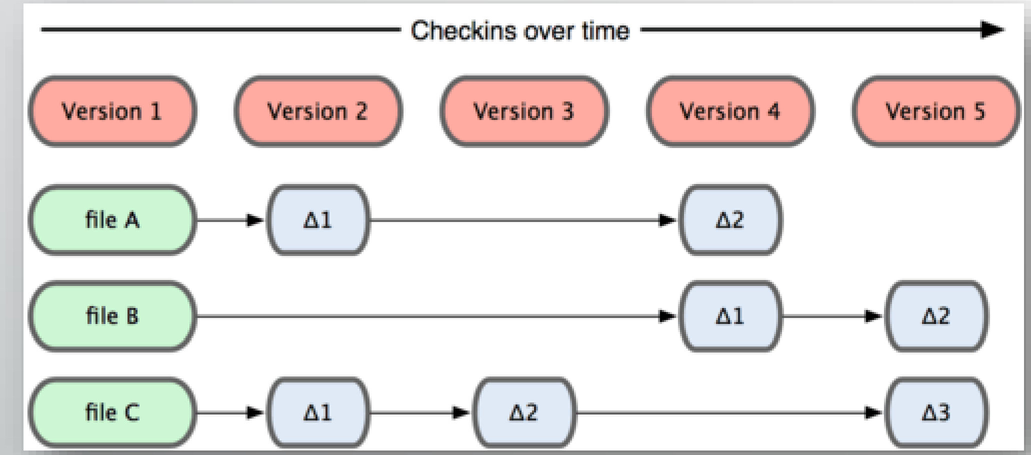
Git vs SVN (DVCS vs CVCS)

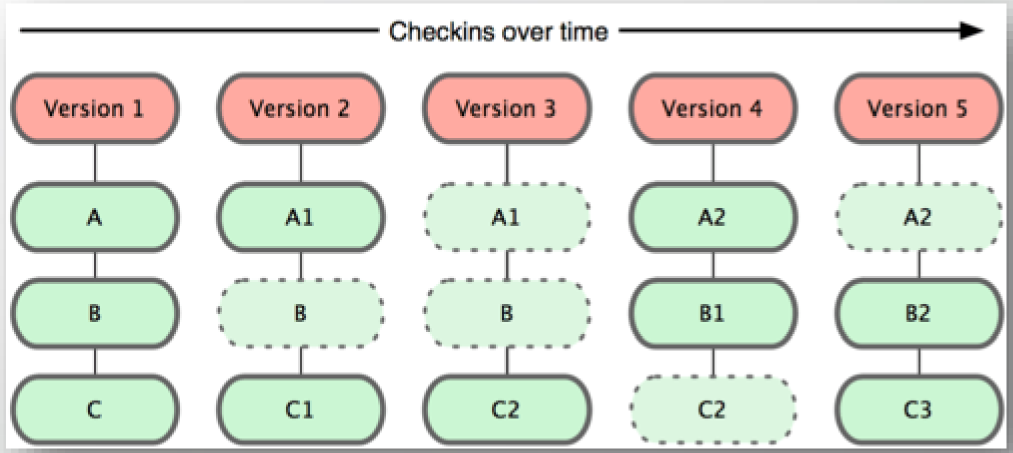
- Before the DVCSs burst into the version controlling world, the most popular VCS was, probably Apache Subversion (known also as SVN). This VCS was centralized (CVCS). A centralized VCS is a system designed to have **a single full copy of the repository, hosted in some server, where the developers save the changes they made**.
- Of course, using a CVCS is better than having **a local version control**, which **is incompatible with teamwork**. But having a version control system that completely depends on a centralized server has an obvious implication: if the server, or the connection to it goes down, the developers won't be able to save the changes. Or even worse, if the central repository gets corrupted, and no backup exists, the history of the repository will be lost.
- **CVCSs can also be slow**. Recording a change in the repository means making effective the change in the remote repository, so, it relies on the connection speed to the server.

Git vs SVN (DVCS vs CVCS)

- Returning to Git and DVCSs, with it, every developer has the full repository locally. So, the **developers can save the changes whenever they want**. If at certain moment the server hosting the repository is down, the developers can continue working
- Another difference with CVCSs, is that DVCSs, specially Git, are **much more faster**, since the changes are made locally, and the disk access is faster than network access, at least in normal situation.
- The differences between both systems could be summed up to the following: with a **CVCS you are enforced to have a complete dependency on a remote server** to carry out your version control, whereas **with a DVCS the remote server is just an option to share the changes**.

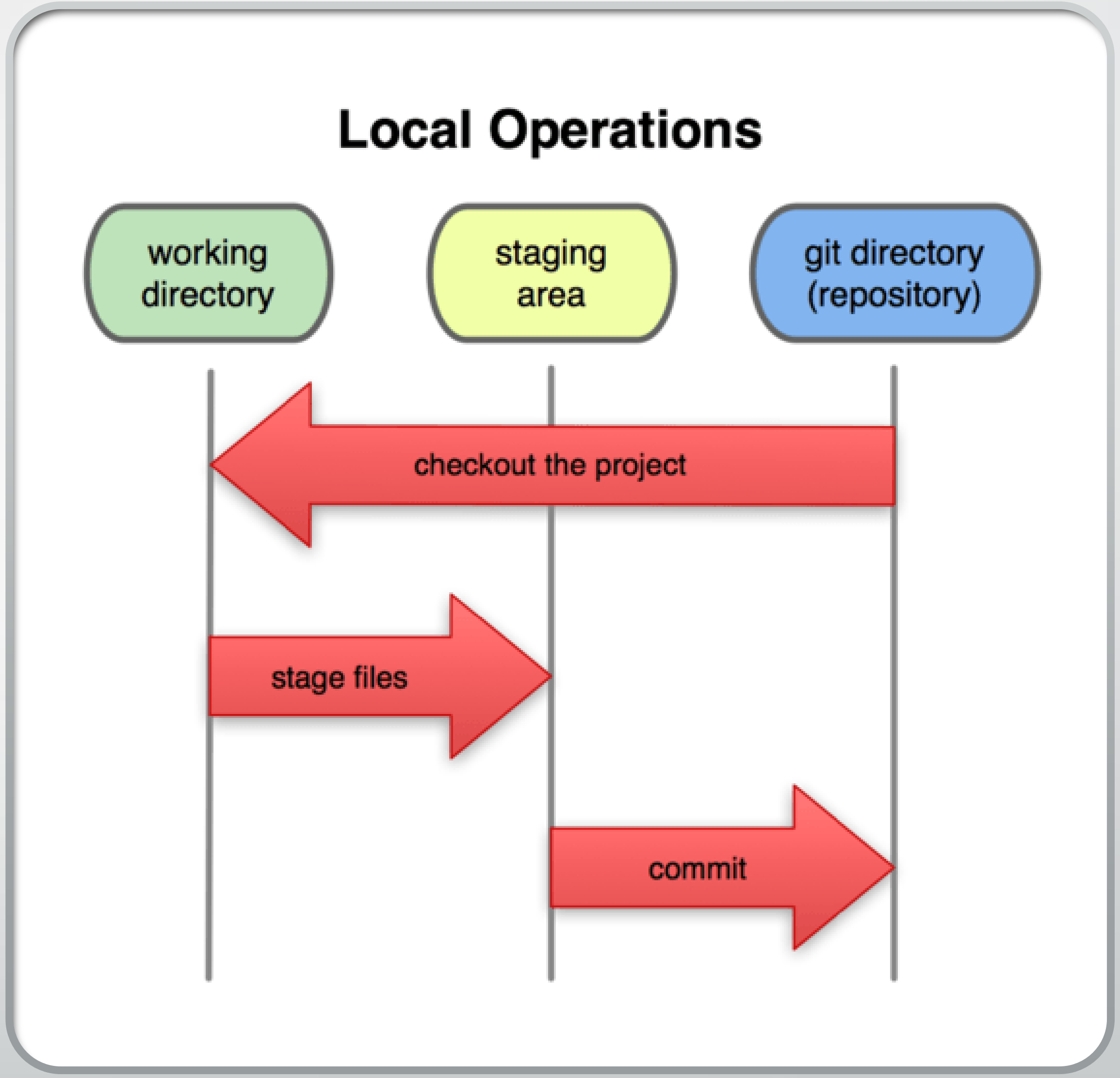
Git vs SVN (DVCS vs CVCS)





Git states

- Modify the files in the working directory
- Add snapshot on the staging area
- Commit the staged files (the snapshot) into the Git repository



Creating/Cloning a repository

```
git init
```

We have a Git repository! Note that a folder named `.git` has been created. The repository will be the directory where the .git folder is placed. This folder is the repository metadata, an embedded database. It's better not to touch anything inside it while you are not familiarized with Git.

```
git clone https://github.com/SCUniTN/Lab12.git
```

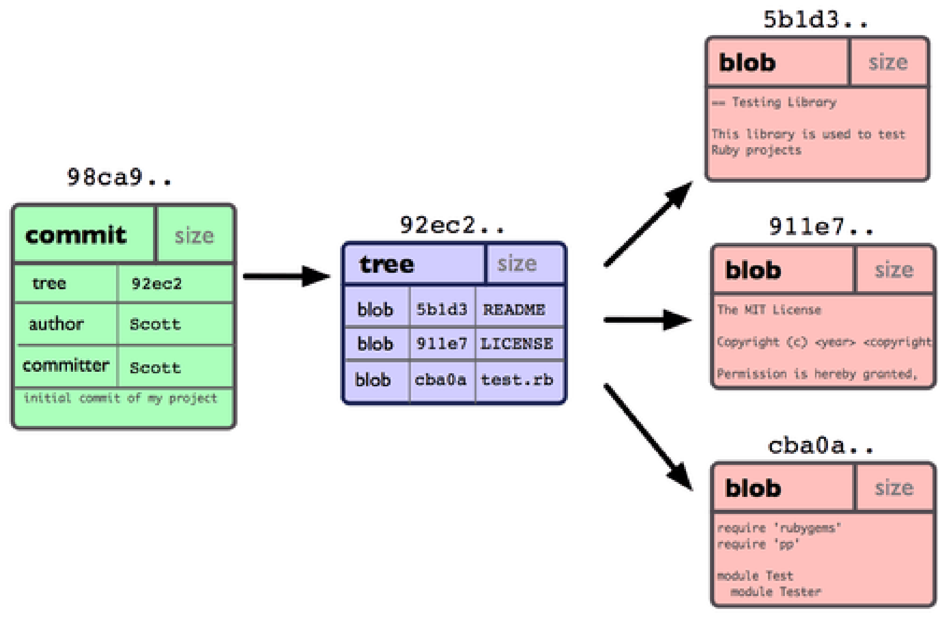
Which would create a local directory with the repository, with the reference to the remote it has been cloned from.

Branches

- Branching is probably the most powerful feature of Git. **A branch represents an independent development path**. The branches coexist in the same repository, but each one has its own history. In the previous section, we have worked with a branch, Git's default branch, which is named **master**.

Branches

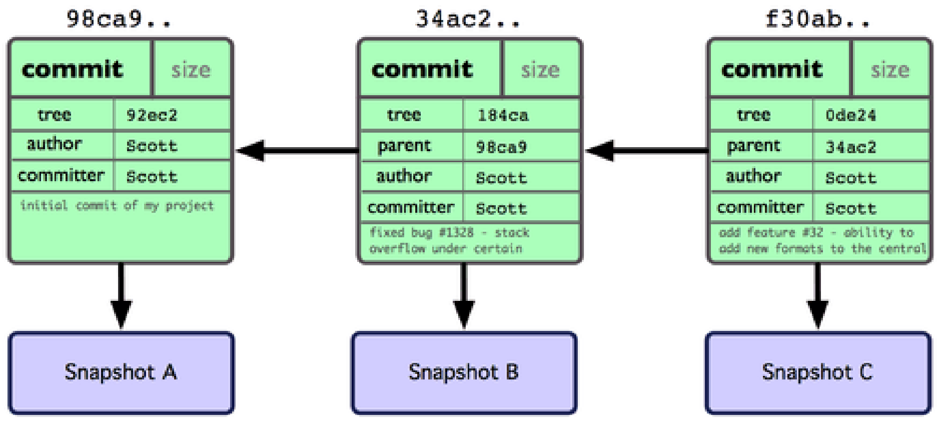
- **How Git stores data?**
- In the example, the repository stores 5 objects:
  - 3 blob: one for each file content
  - 1 tree: lists directory content and specifies the file names
  - 1 commit: stores meta-data of the snapshot and has a link to the tree-root



```
> git add README test.rb LICENSE
> git commit -m 'initial commit of my project'
```

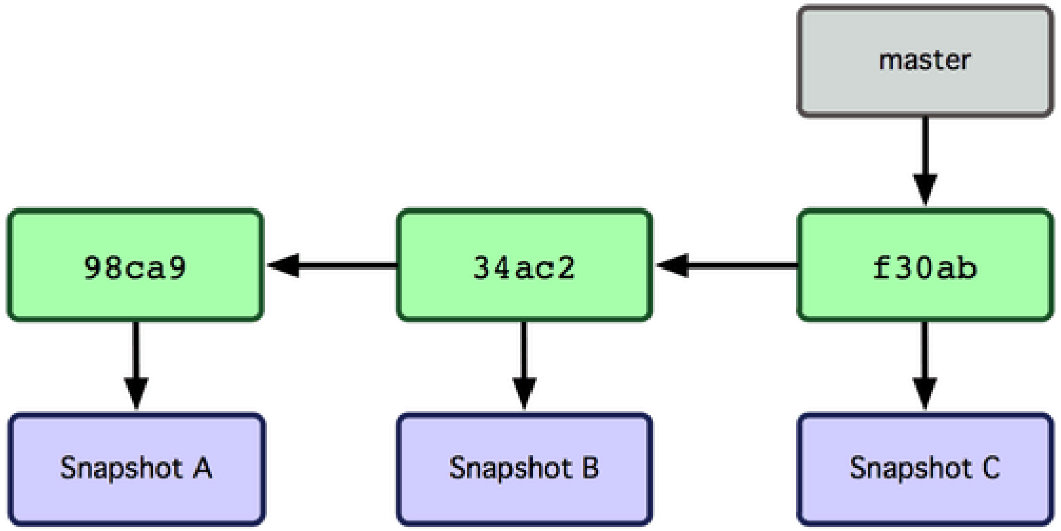
Branches

- **How Git stores data?**
- Each new commit stores a link (pointer) to the previous commit



Branches

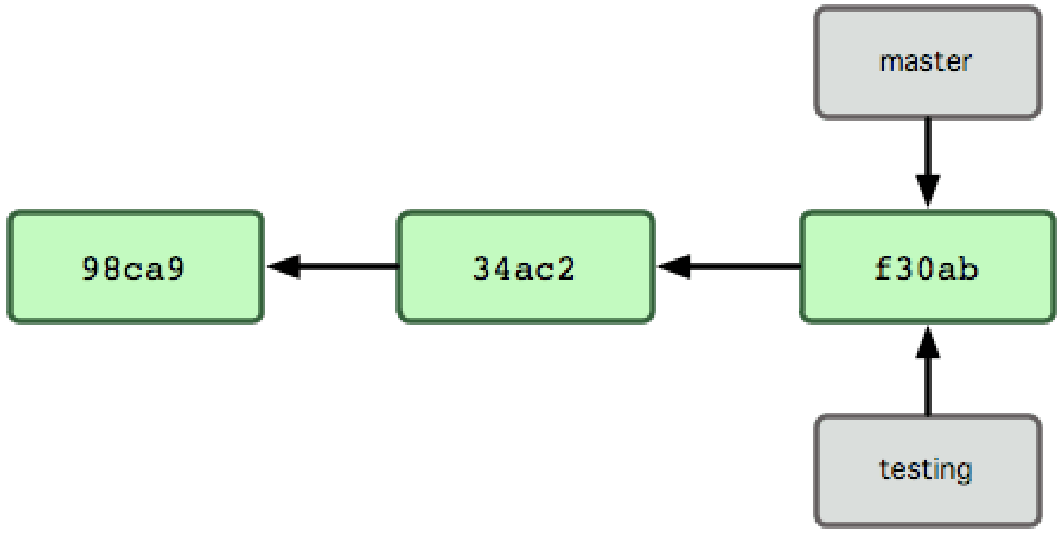
- **What is a branch?**
- A branch is a pointer to a commit. Usually the last commit.
- The default branch is **master**



Branches

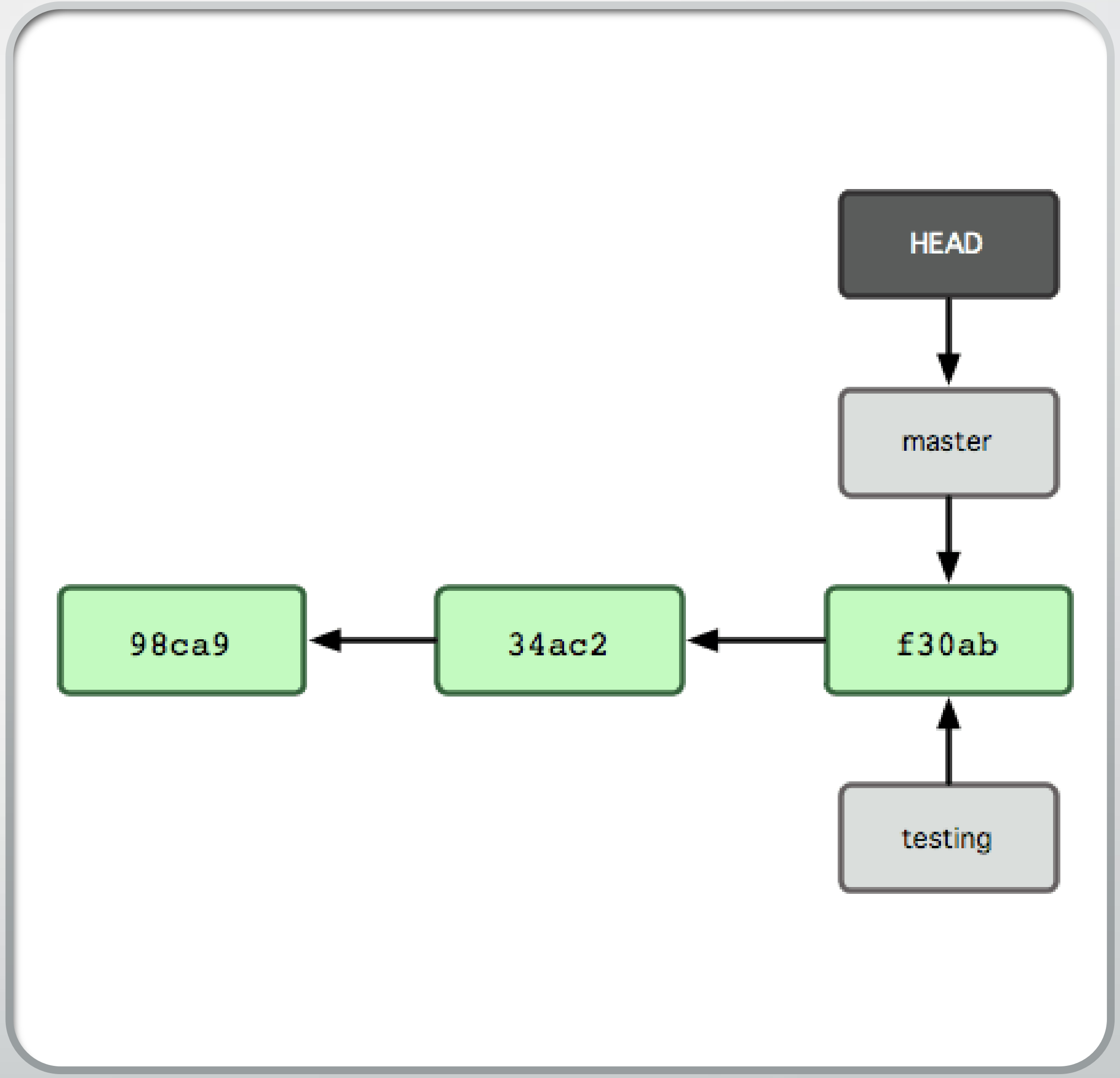
- **Create new branch**
- Creating new branch, we add a pointer to the last commit.

```
> git branch testing
```



Branches

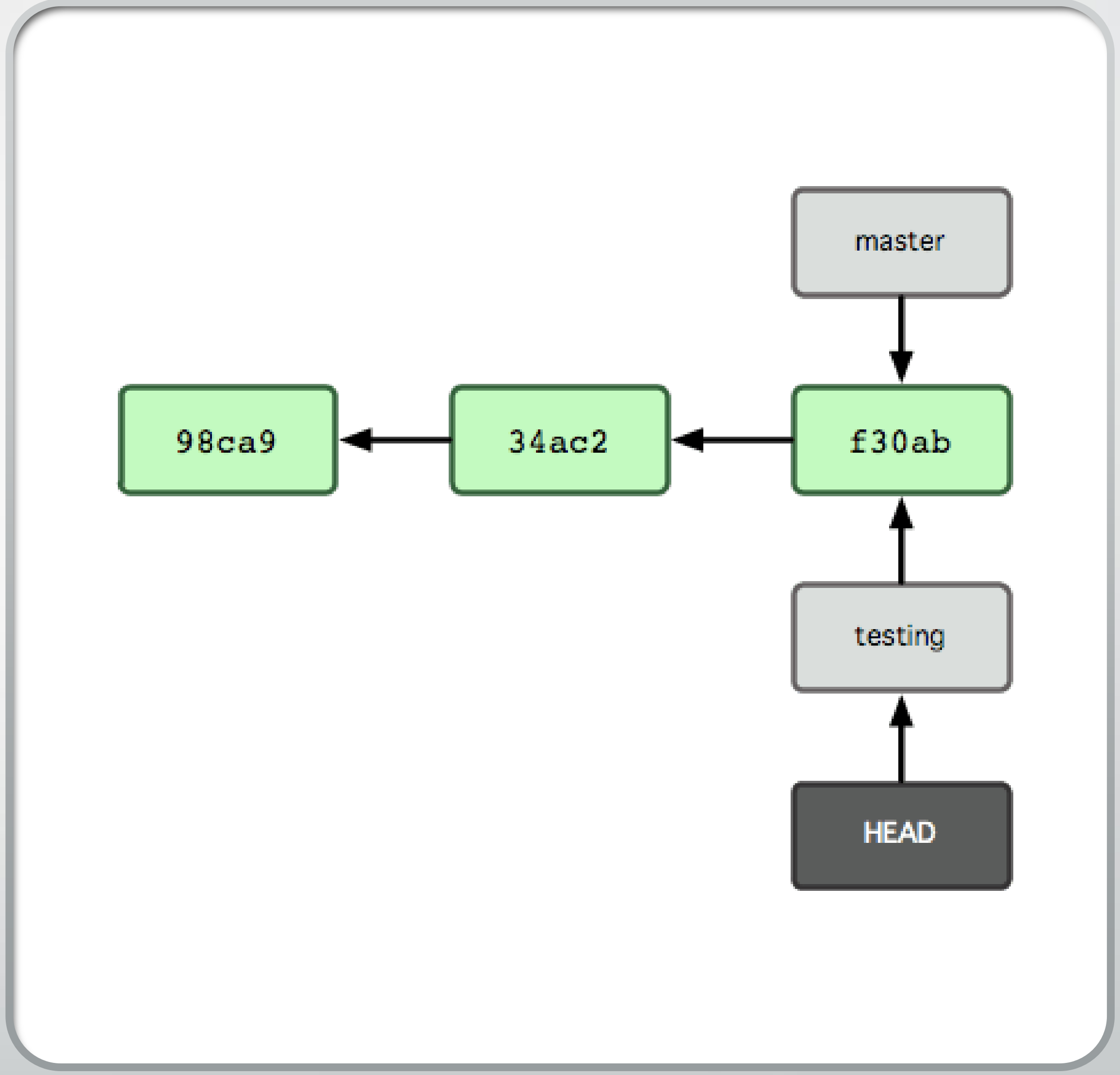
- **In which branch am I?**
- Git uses a special pointer to store which is the active branch: **HEAD**.



Branches

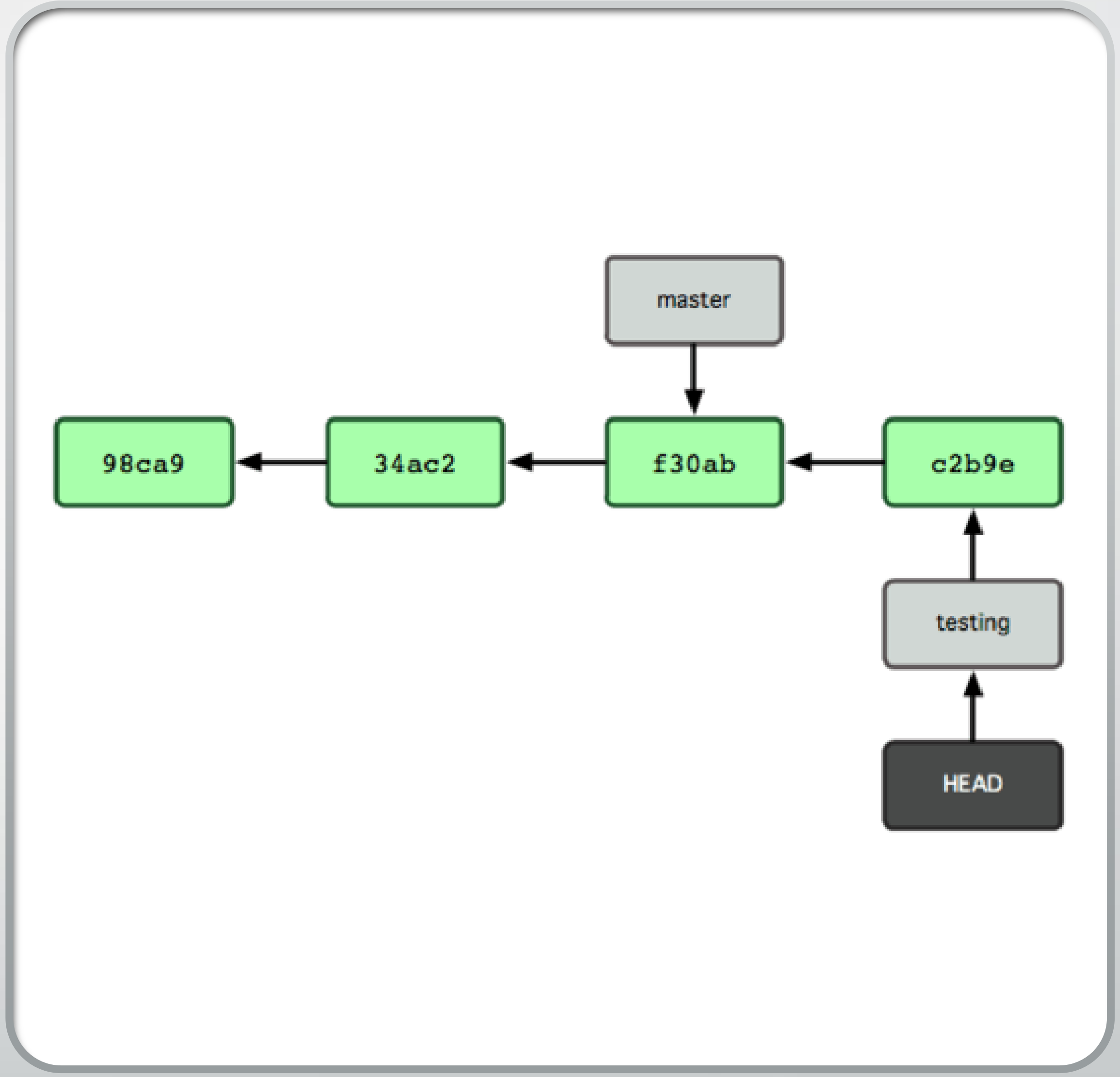
- **How to change branch?**
- You can change the branch via **checkout**.

```
> git checkout testing
```



Branches

- **Working with branches**
- Commit file modifications

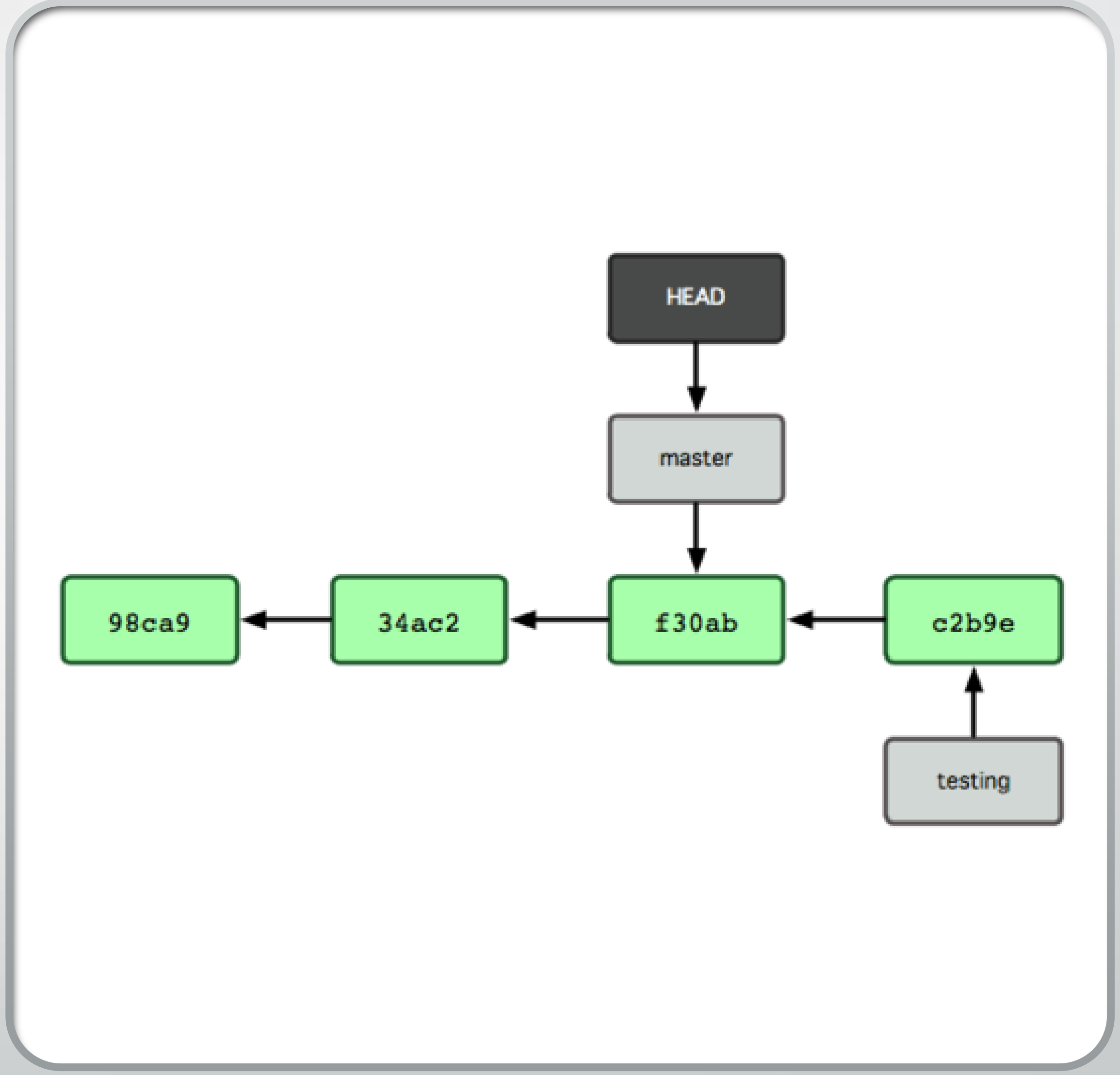


```
> … modified a file …
> git commit -a -m 'modified file …'
```

Branches

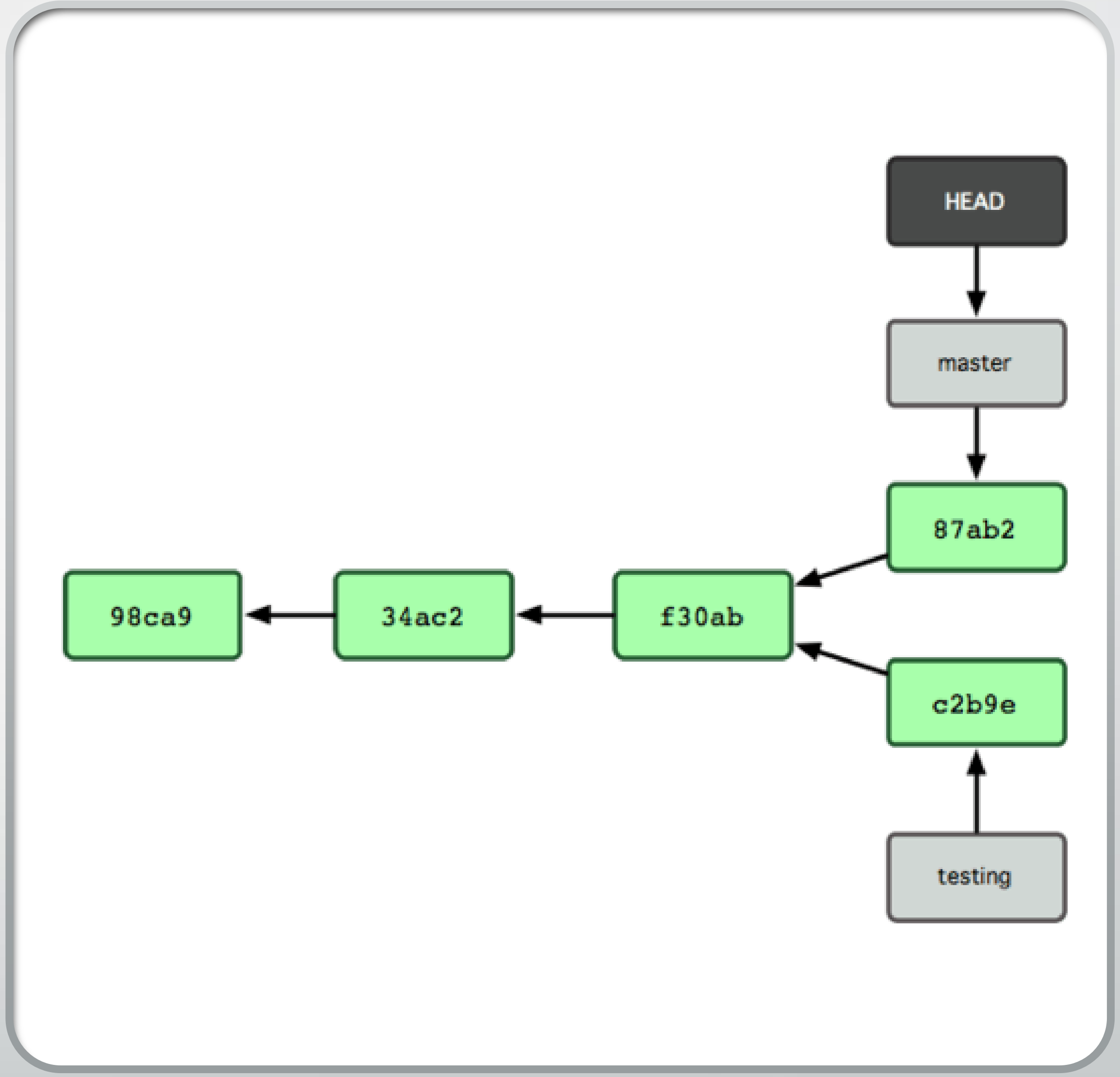
- **Working with branches**
- Return to master branch

```
> git checkout master
```



Branches

- **Working with branches**
- Commit new modifications



```
> … modified a file …
> git commit -a -m 'modified another file …'
```

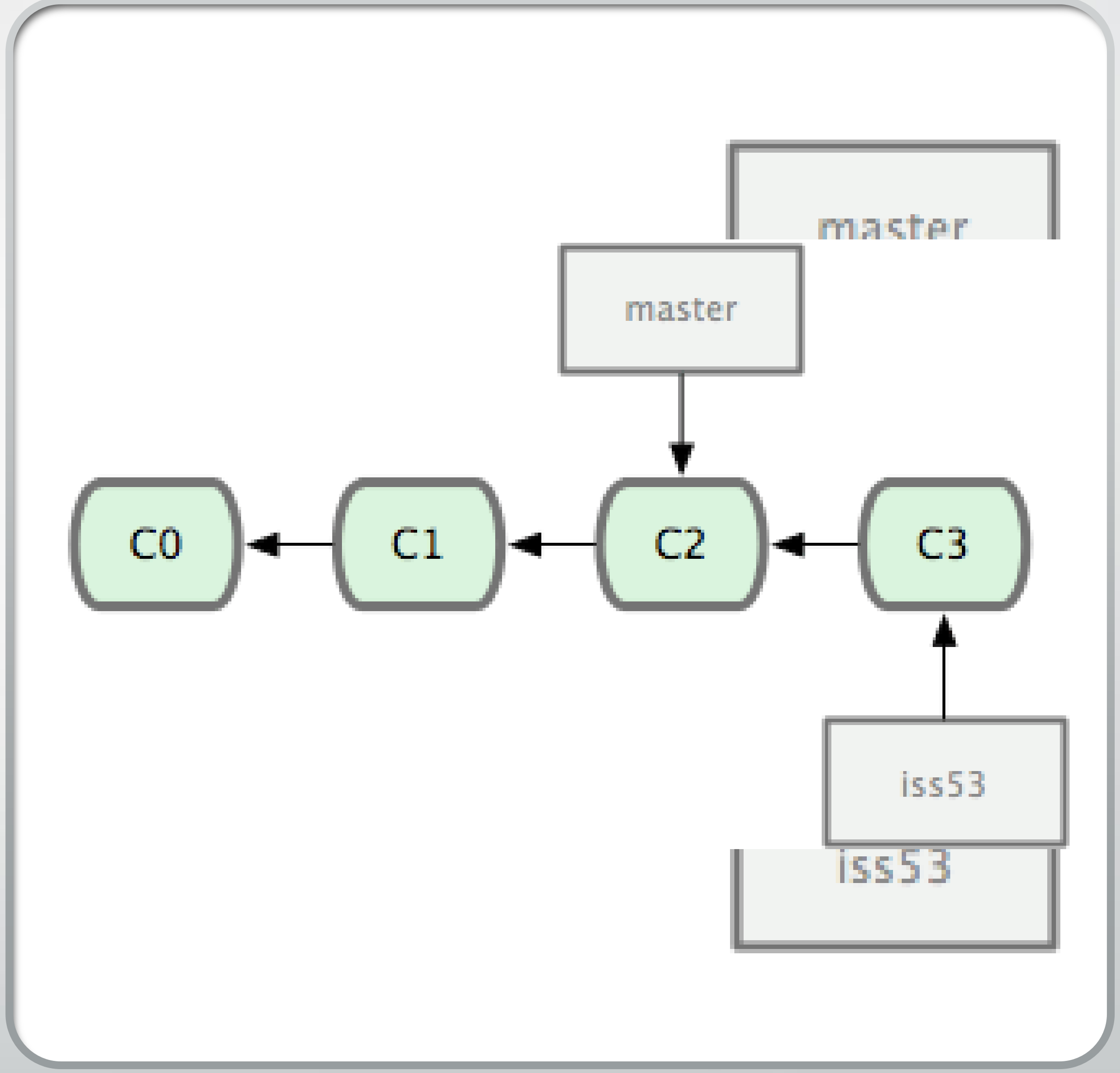
Branches

- **Example**

1. Working on a web app
2. Create a branch for new features (iss53)
3. Develop new features (iss53)

```
> git branch iss53
> git checkout iss53
modified a file …
> git commit -a -m 'modified some files on iss53'
```

```
> git checkout -b iss53
```

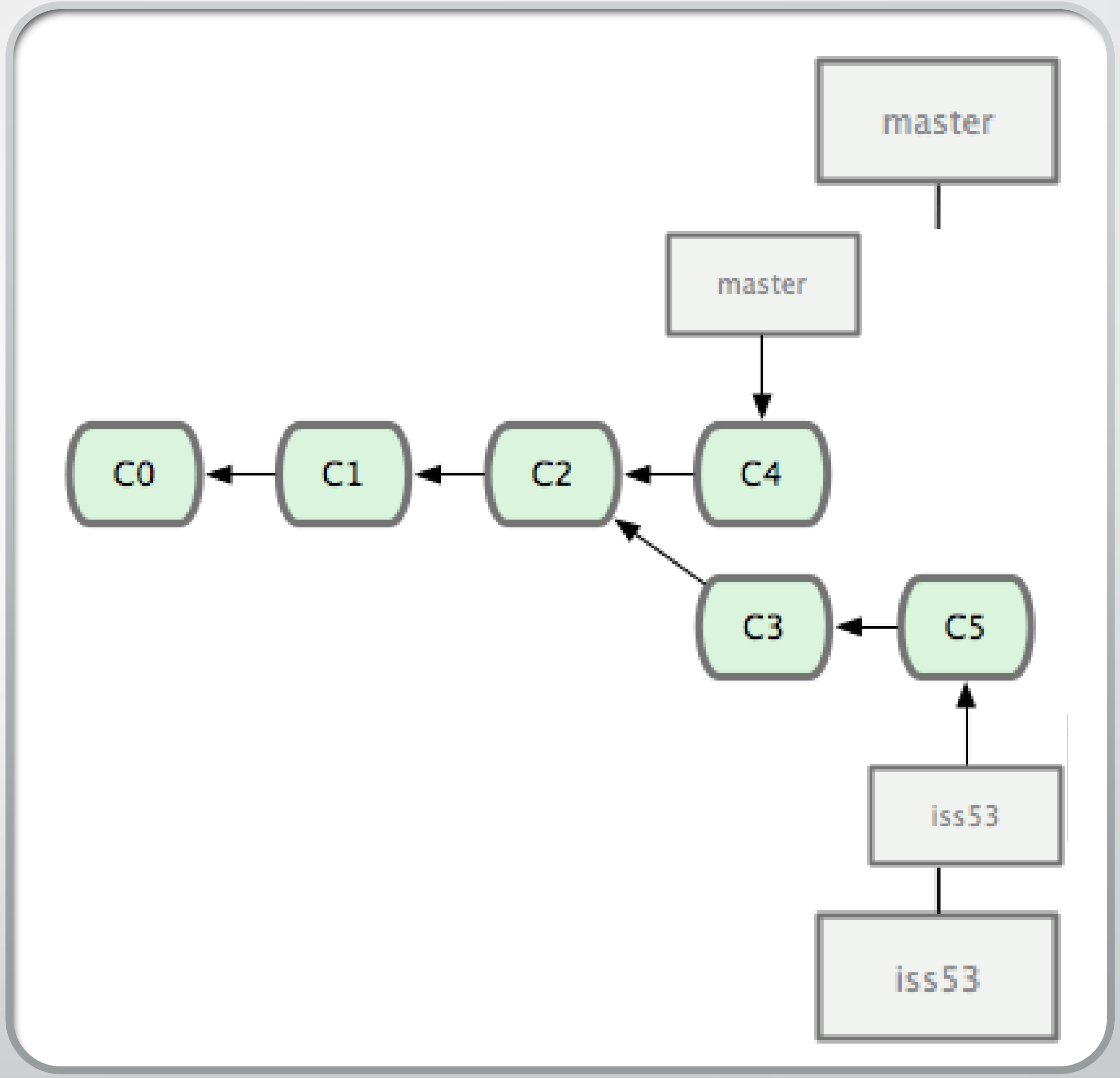


Branches

- **Example**
- Some problems on production version!!!

1. Back to production branch
2. Create an hotfix branch
3. Dev/Test and merge hotfix
4. Back to new feature iss53

```
> git checkout master
> git branch -d hotfix
> git checkout hotfix
> git checkout master
> git merge hotfix
> git checkout iss53
```


master
master
C0
C1
C2
C4
C3
C5
iss53
iss53