# GIT 101

Michał Zając

Akademickie Stowarzyszenie Informatyczne

## AGENDA

## GIT 101

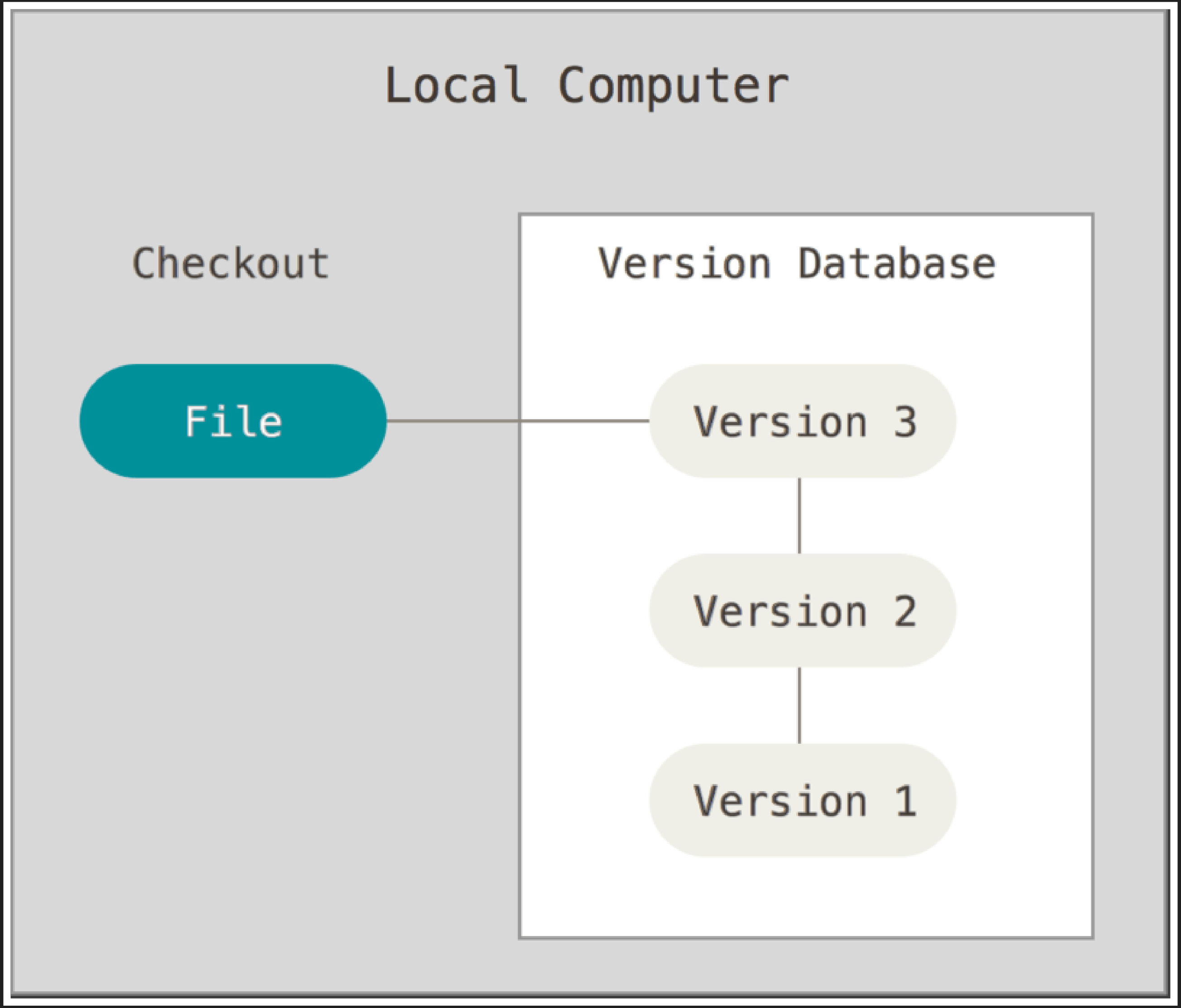
1. Brief history of VCSes
2. A few things before we begin
3. Git basics
4. Branching
5. Customizing Git

## BRIEF HISTORY OF VCSES

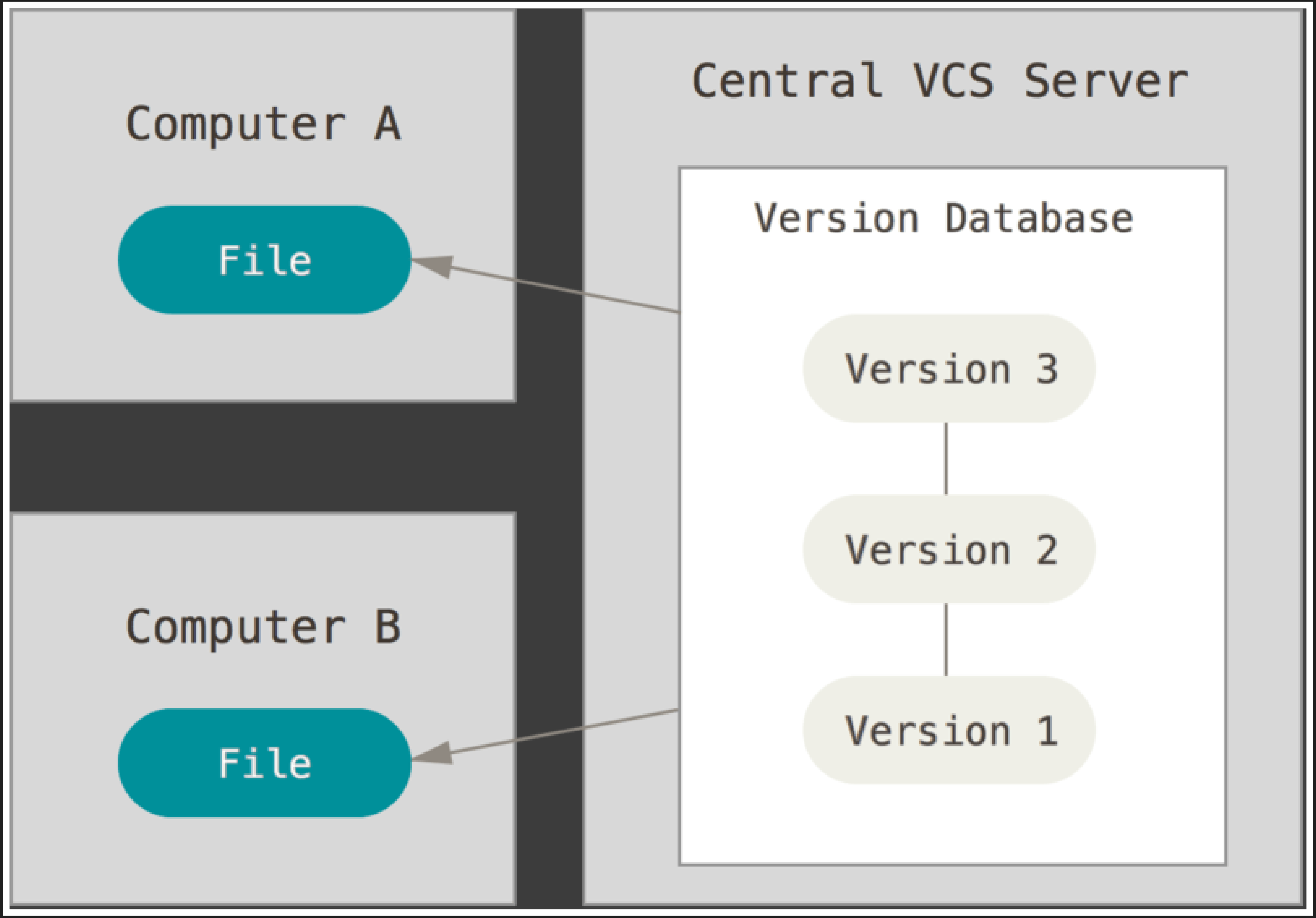
## SVCS

Simple (or stupid) Version Control System.

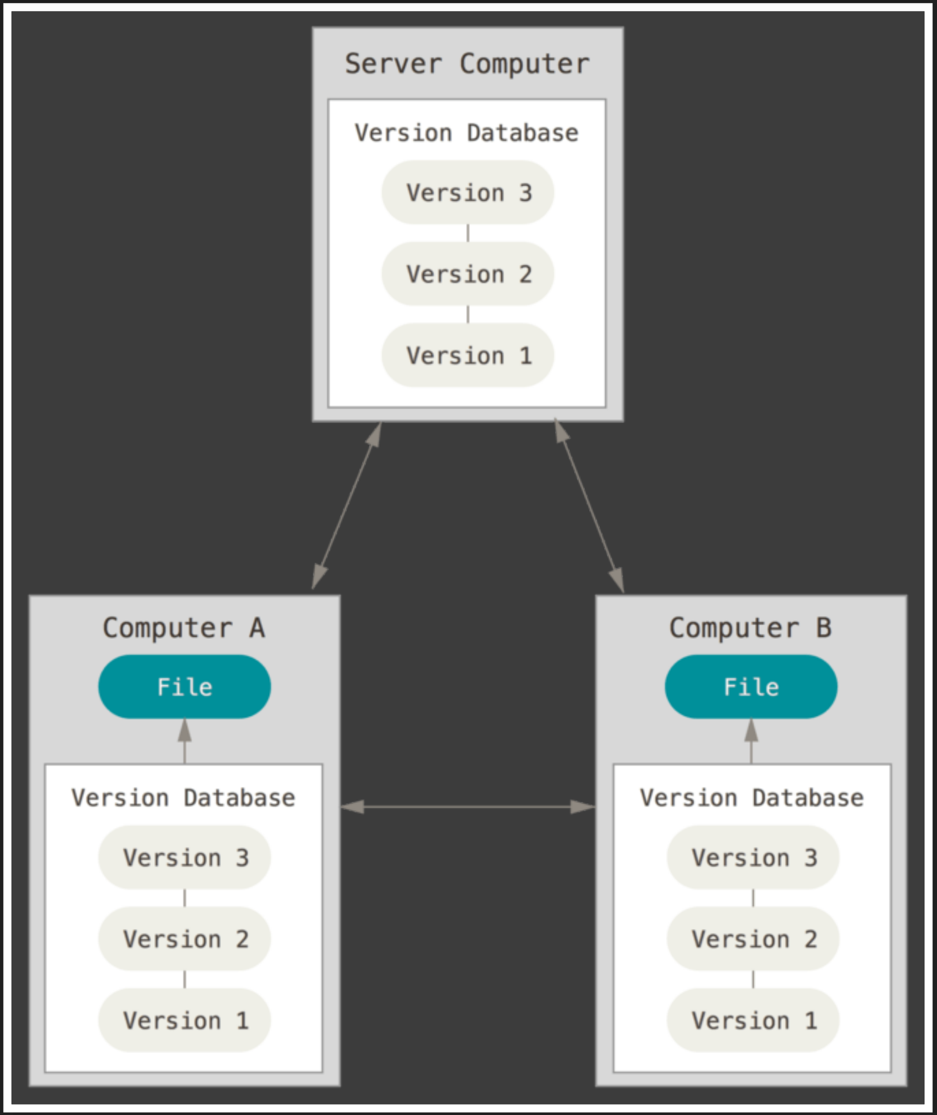
# LVCS


Local Computer
Checkout
File
Version Database
Version 3
Version 2
Version 1

# CVCS


Computer A
File
Central VCS Server
Version Database
Version 3
Version 2
Version 1
Computer B
File

# DVCS


Server Computer
Version Database
Version 3
Version 2
Version 1
Computer A
File
Version Database
Version 3
Version 2
Version 1
Computer B
File
Version Database
Version 3
Version 2
Version 1

# A FEW THINGS BEFORE WE BEGIN

## INTRODUCE YOURSELF

Git would like to know who you are so we have to introduce ourselves.

```
$ git config --global user.name "Michał Zając"
$ git config --global user.email "michal.zajac@gmail.com"
```

## EDITOR

Git uses you system's default editor unless you tell it to use something else:

```
$ git config --global core.editor vim
```

## VERIFYING SETTINGS

You can check your settings with
`git config --list`

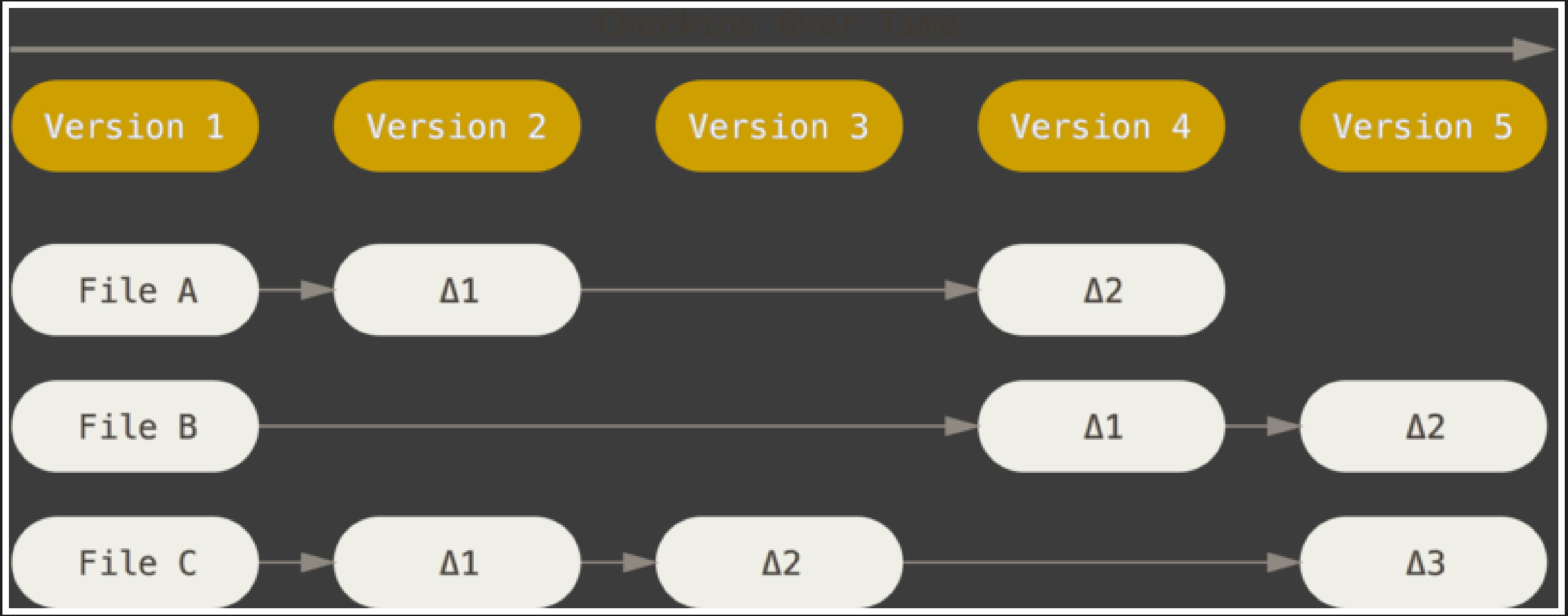
```
$ git config --list
user.name=Michał Zając
user.email=michal.zajac@gmail.com
user.signingkey=79313F09
push.default=simple
pull.rebase=preserve
alias.s=status -s
```

## I WILL BE USING AN ALIAS

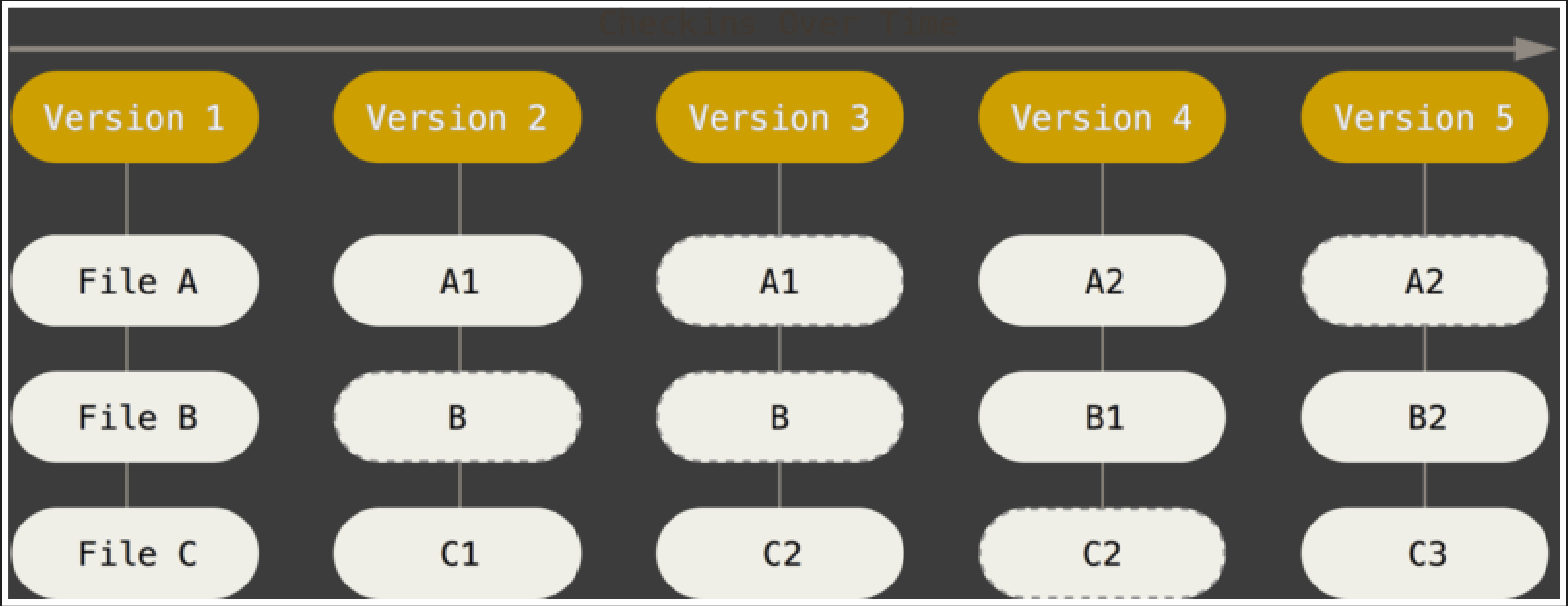
```
git glog
```

Check Facebook event page for details

# SNAPSHOTS INSTEAD OF DIFFERENCES



## SNAPSHOTS INSTEAD OF DIFFERENCES



## NEARLY EVERYTHING IS LOCAL

Don't have internet connection? Not a problem.

*cough* unlike Subversion *cough*

## GIT HAS INTEGRITY

## GIT GENERALLY ONLY ADDS DATA

THIS IS VERY IMPORTANT

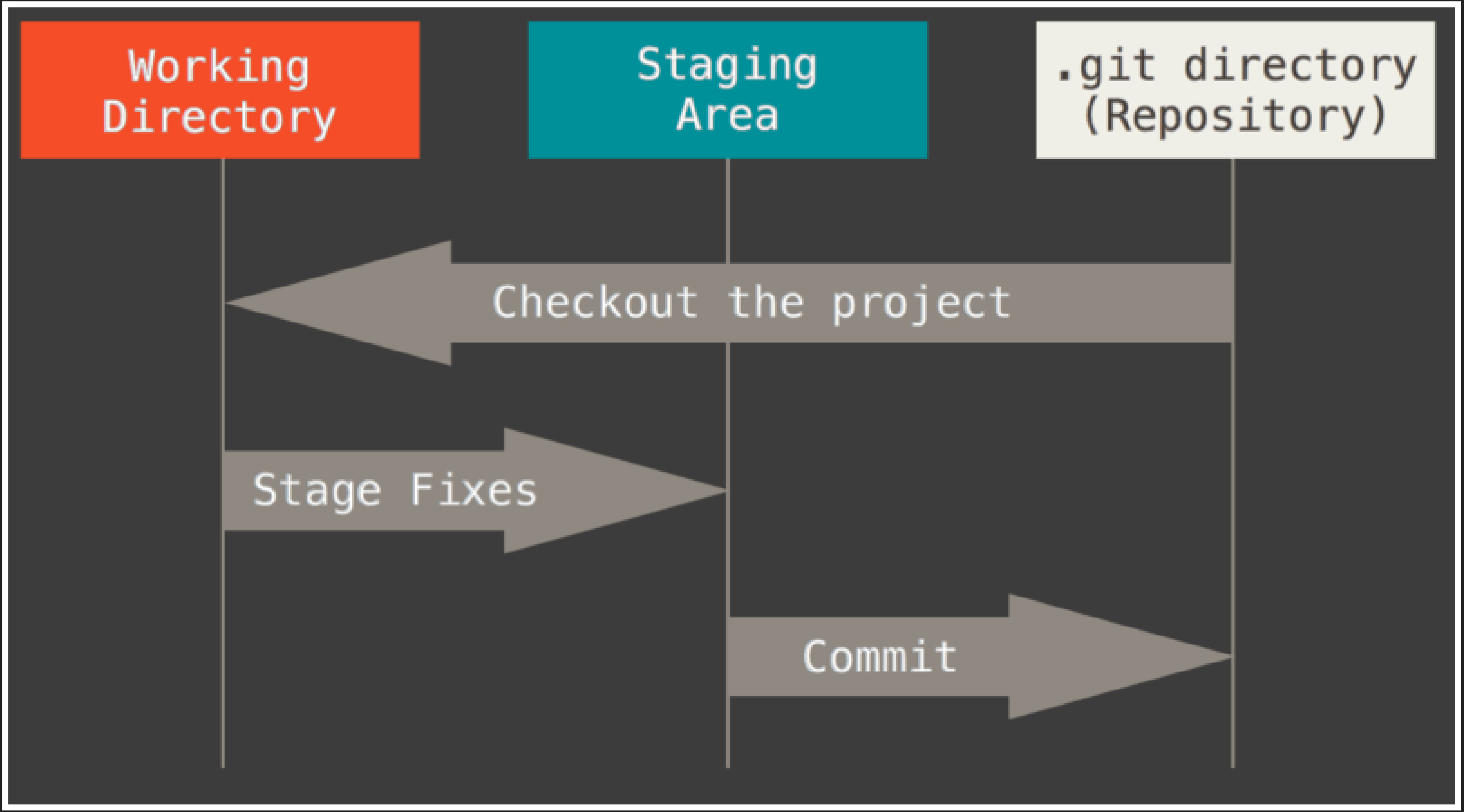
GIT GUD OR
GIT REKT

## THREE STATES

Hark for this is main thing to remember from this section if you want your learning to proceed smoothly.

Git has three states that your files can reside in: commited, modified, and staged.

# THREE STATES



## GIT BASICS

## CREATING A GIT REPOSITORY

```
$ cd my_project
$ git init
Initialized empty Git repository in /home/quintasan/my_project
$ git add .
$ git commit -m "Initial commit"
```

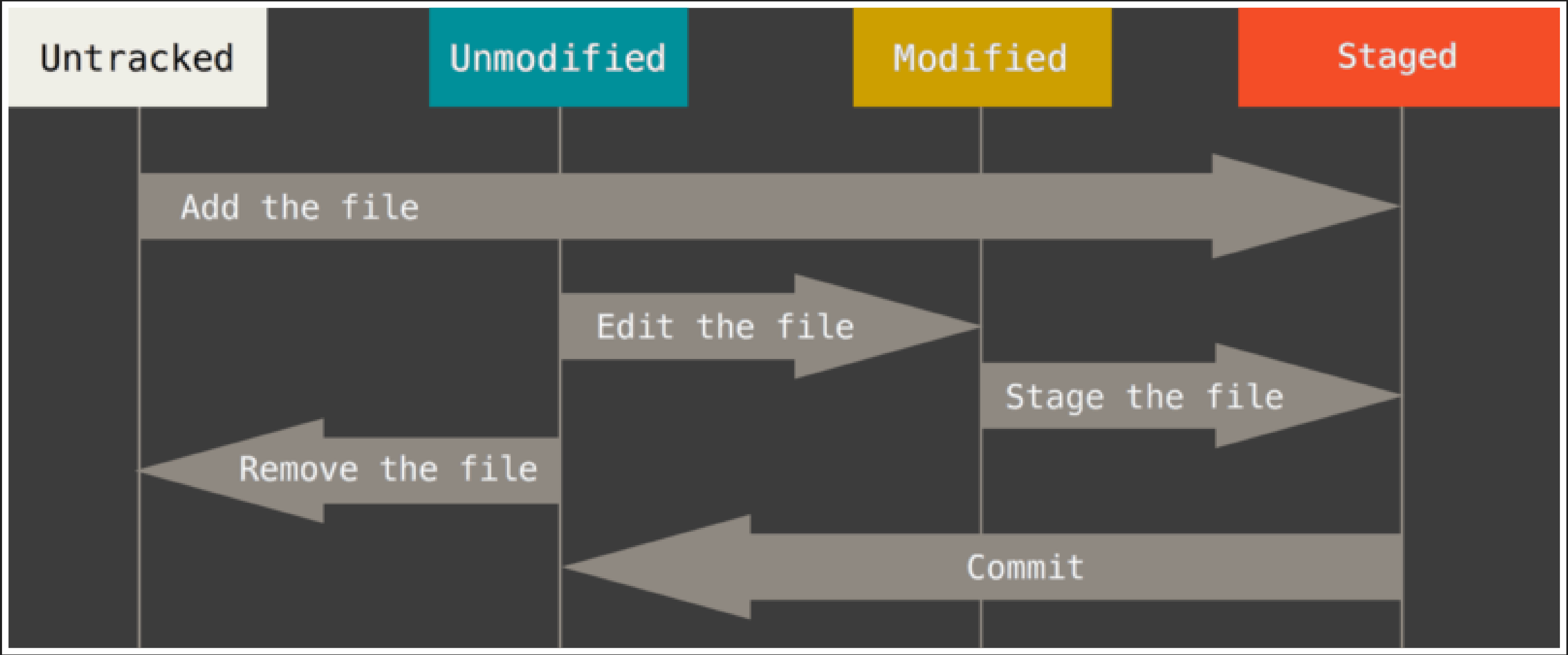
## GETTING A GIT REPOSITORY

```
$ git clone git@github.com:Quintasan/misc.git
```

## CHECKING STATUS OF YOUR FILES

`git status` is your friend. Be sure to visit your friends very often.

# POSSIBLE FILE STATES



## TRACKING NEW FILES

`git add` is the way

## STAGING MODIFIED FILES

`git add` is also the way

## IGNORING FILES

Add 'em to `.gitignore`

## VIEWING STAGED AND UNSTAGED CHANGES

To view unstaged changes: `git diff`

To view staged changes: `git diff --cached`

## COMMITING

This one is obvious - `git commit`

## REMOVING FILES

Piece of cake - `git rm`

## MOVING FILES AROUND

`git mv` which is equivalent to

```
$ mv README.md README
$ git rm README.md
$ git add README
```

## VIEWING THE HISTORY

`git log` allows you to browse the commit history

## BRIEF INTERMISSION

The following sections will touch upon things that can lead to data loss if handled incorrectly.

## ONE SIMPLE RULE

Do not rewrite history once you made it public.

## I FORGOT TO ADD THINGS TO A COMMIT

Use `git commit --amend`

## UNSTAGING A STAGED FILE

Added too much? Use `git reset HEAD <file>`.

For now this is the only reset command you need.

## UNMODIFYING A MODIFIED FILE

Your refactor was a bad idea afer all? Use `git checkout -- <file>`.

THIS CAN AND WILL RESULT IN DATA LOSS.

## WORKING WITH REMOTES

Since Git is distributed we often have to work with remote hosts.

## LISTING REMOTES

`git remote` will show all remotes

## ADDING REMOTES

```
git remote add <name> <url>
```

## FETCHING AND PULLING

`git fetch` vs. `git pull`

## PUSHING DATA

```
git push <remote name> <branch name>
```

## ADDING, REMOVING AND RENAMING REMOTES

```
git remote rm <name>,
git remote rename <name> <new_name>
```

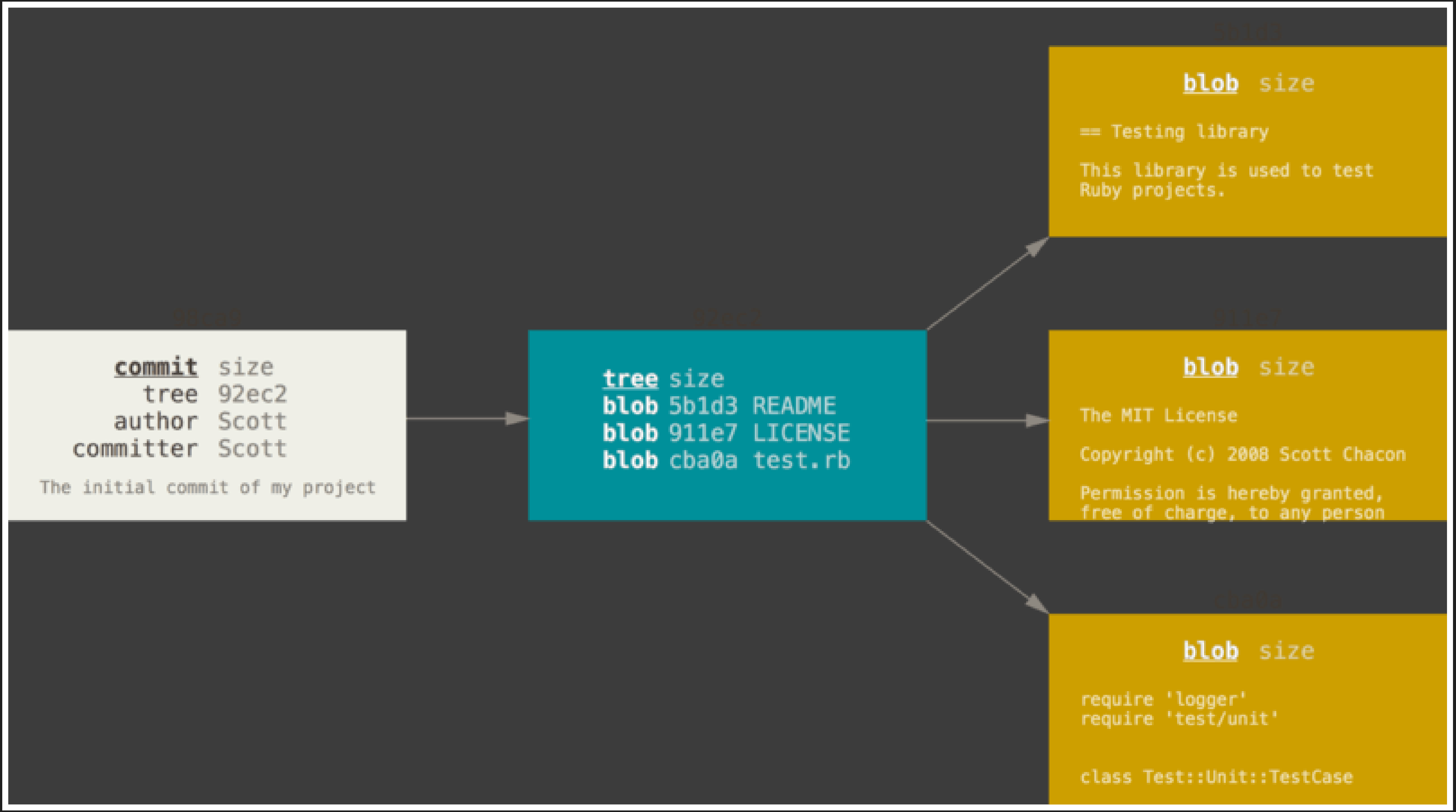
## BRANCHING

## BRANCHES IN A NUTSHELL

Let us a take a step back and examine how Git stores its data.

## EXAMPLE

# EXAMPLE

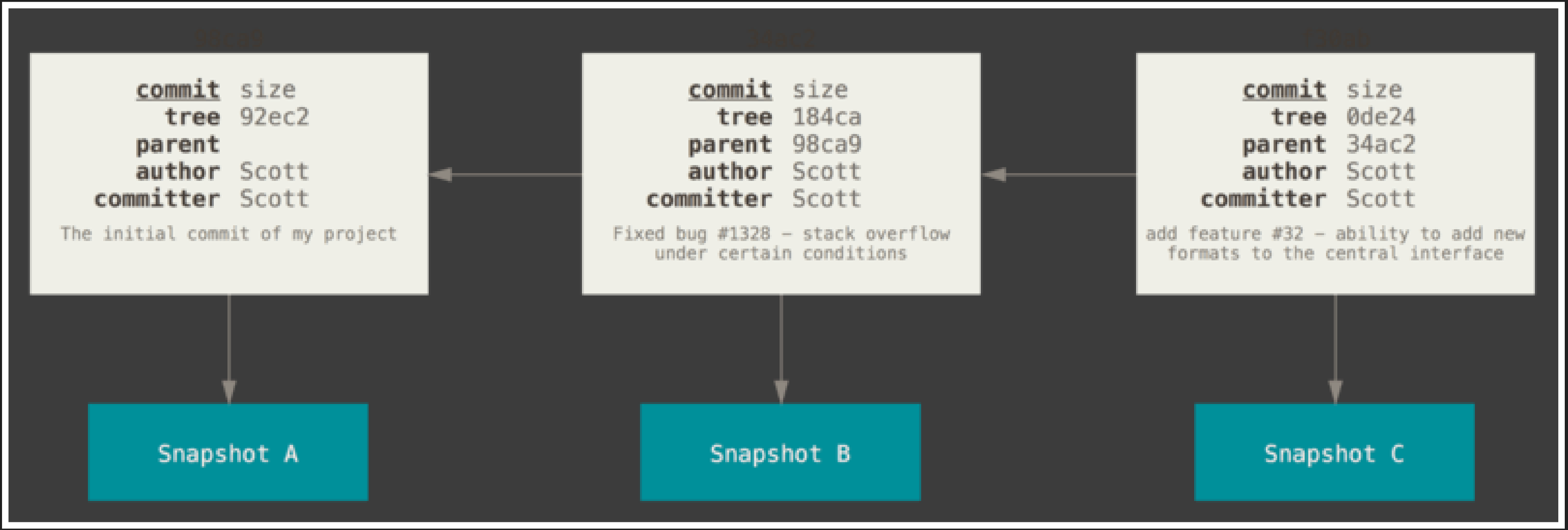


```
== Testing library

This library is used to test
Ruby projects.
```

```
The MIT License

Copyright (c) 2008 Scott Chacon

Permission is hereby granted,
free of charge, to any person
```

```
require 'logger'
require 'test/unit'


class Test::Unit::TestCase
```

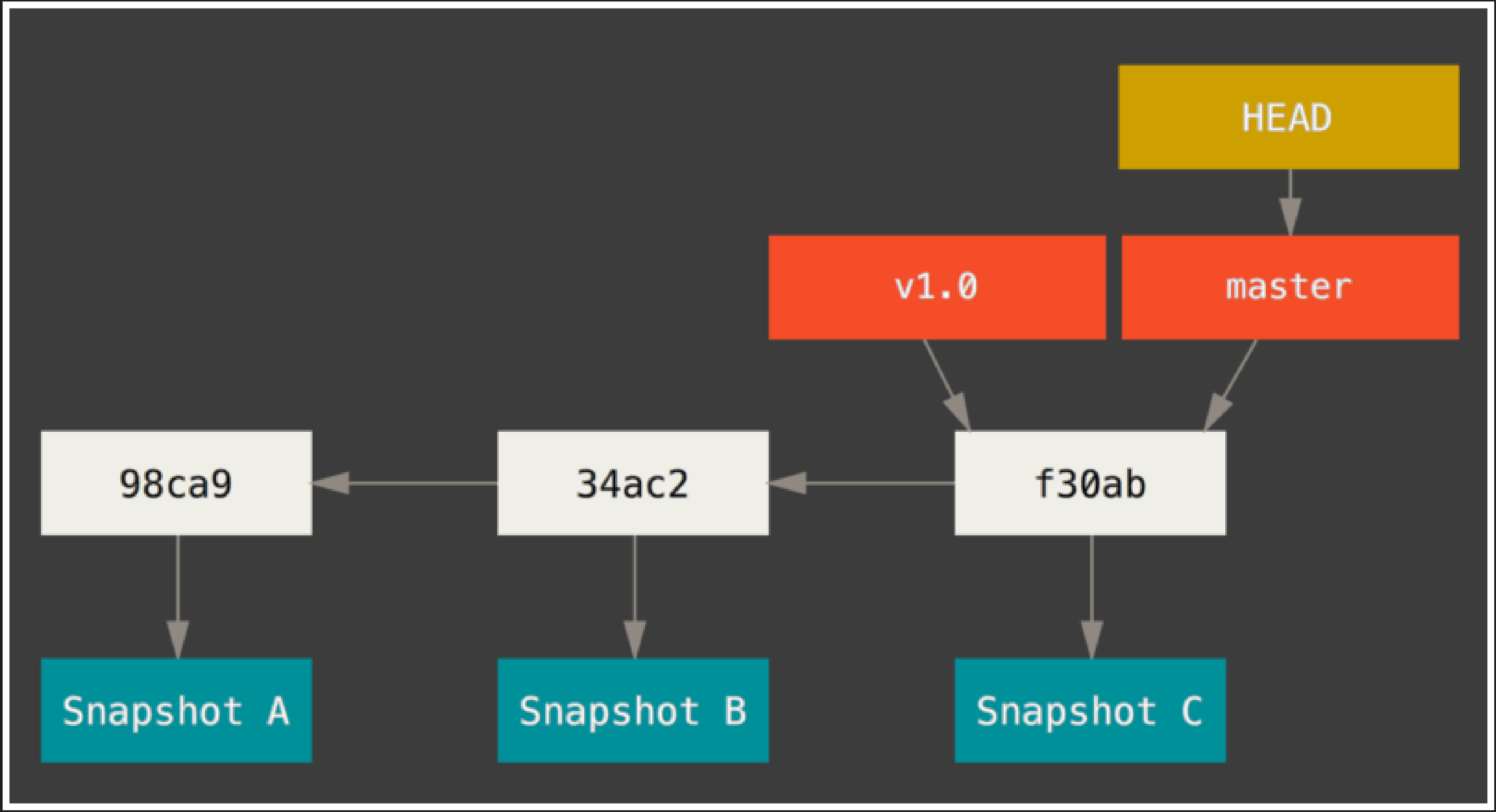
# EXAMPLE CONT.



## WHAT IS A BRANCH?

Essentially branch is a movable pointer to one of these commits.

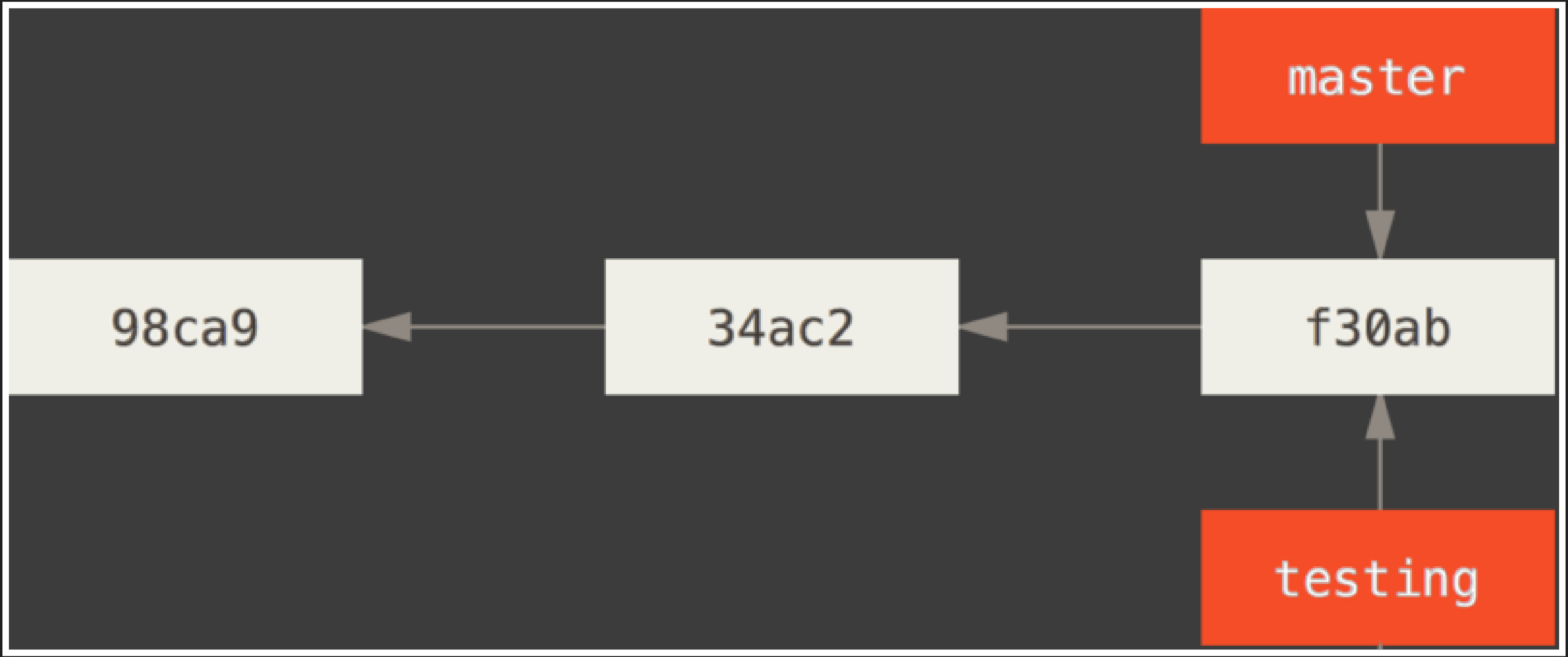
# A BRANCH AND ITS COMMIT HISTORY



## CREATING A NEW BRANCH

```
git branch <branch name>
```

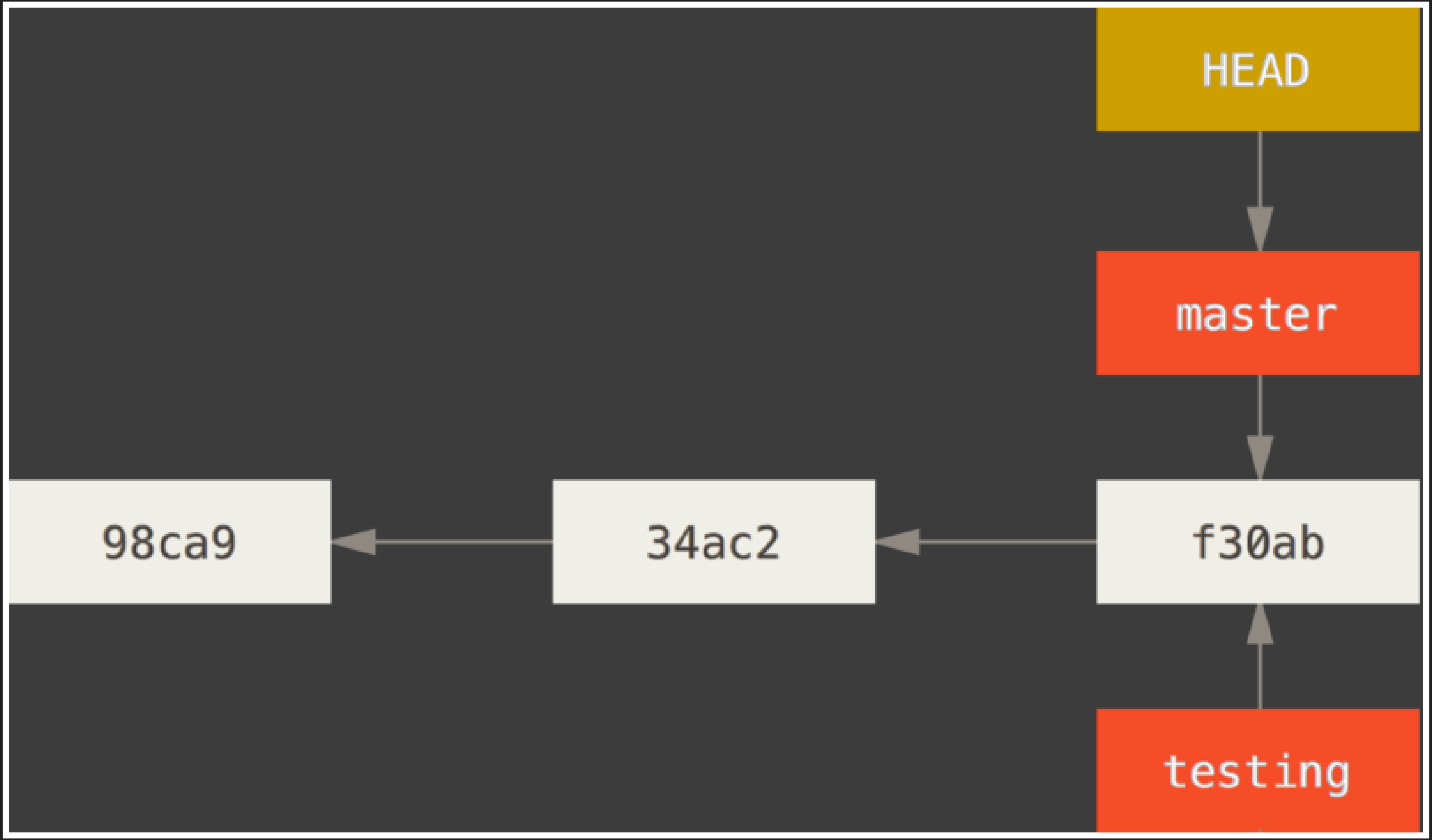
# CREATING A NEW BRANCH



## HOW DOES GIT KNOW WHERE AM I?

It uses it's HEAD (please don't judge me).

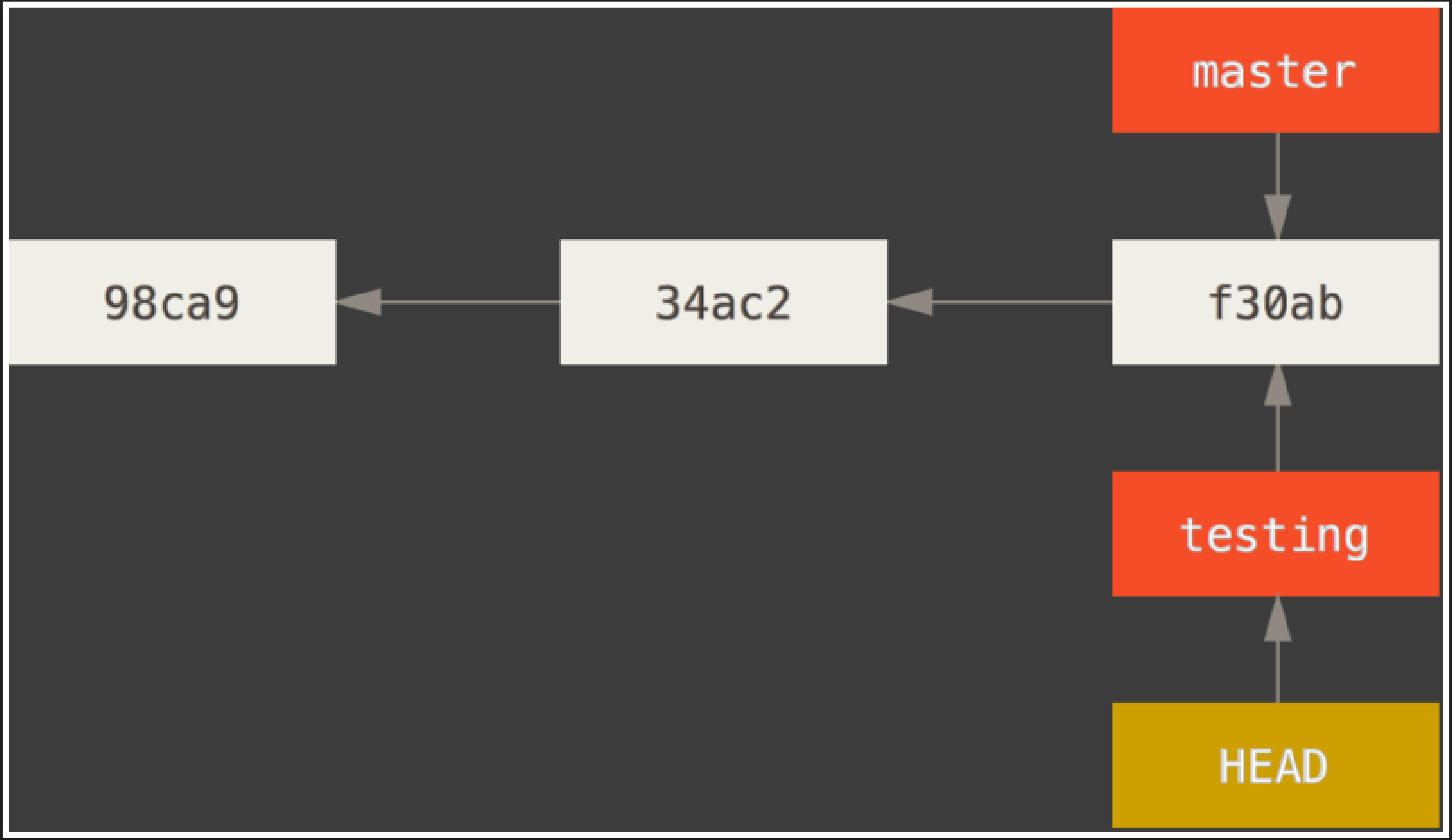
## GIT USING IT'S HEAD



## SWITCHING BRANCHES

```
git checkout <branch name>
```

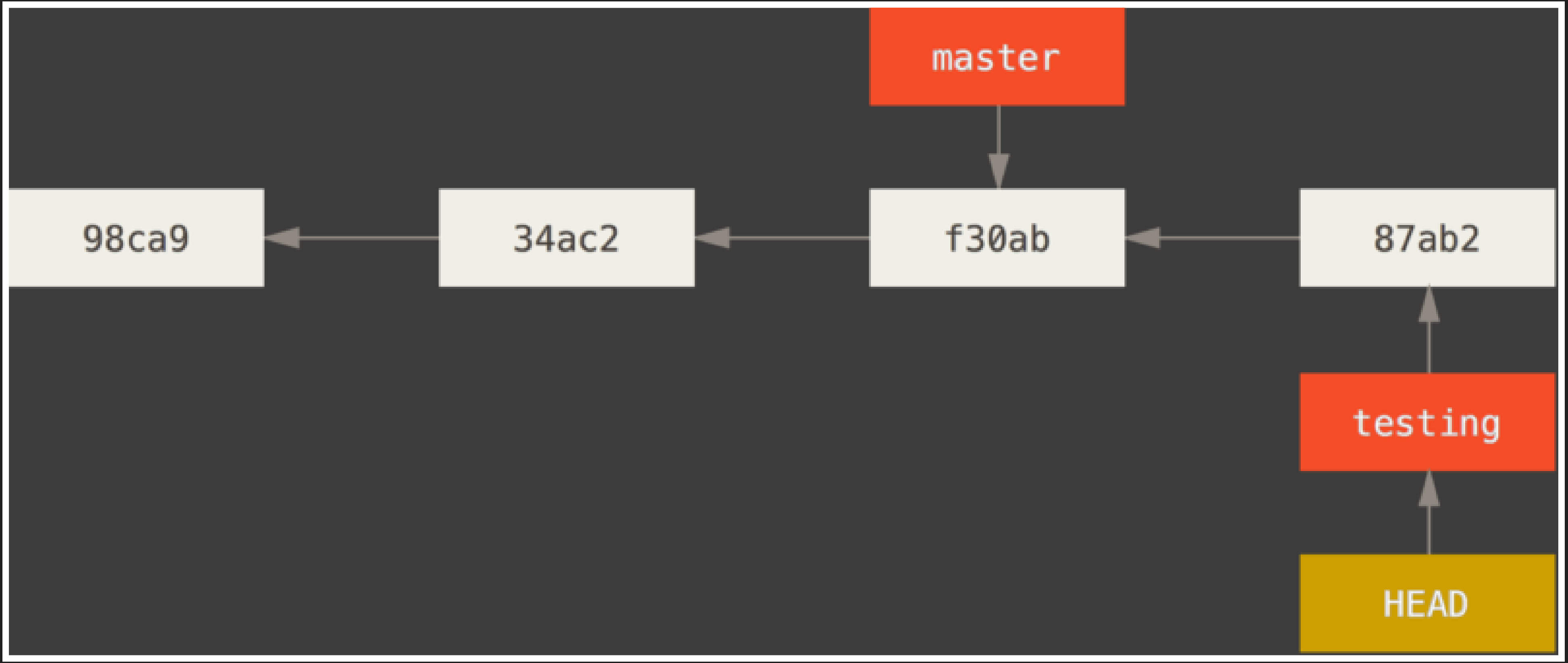
# SWITCHING BRANCHES



## ADVANCING BRANCHES

Let's change something and see the state of our branches.

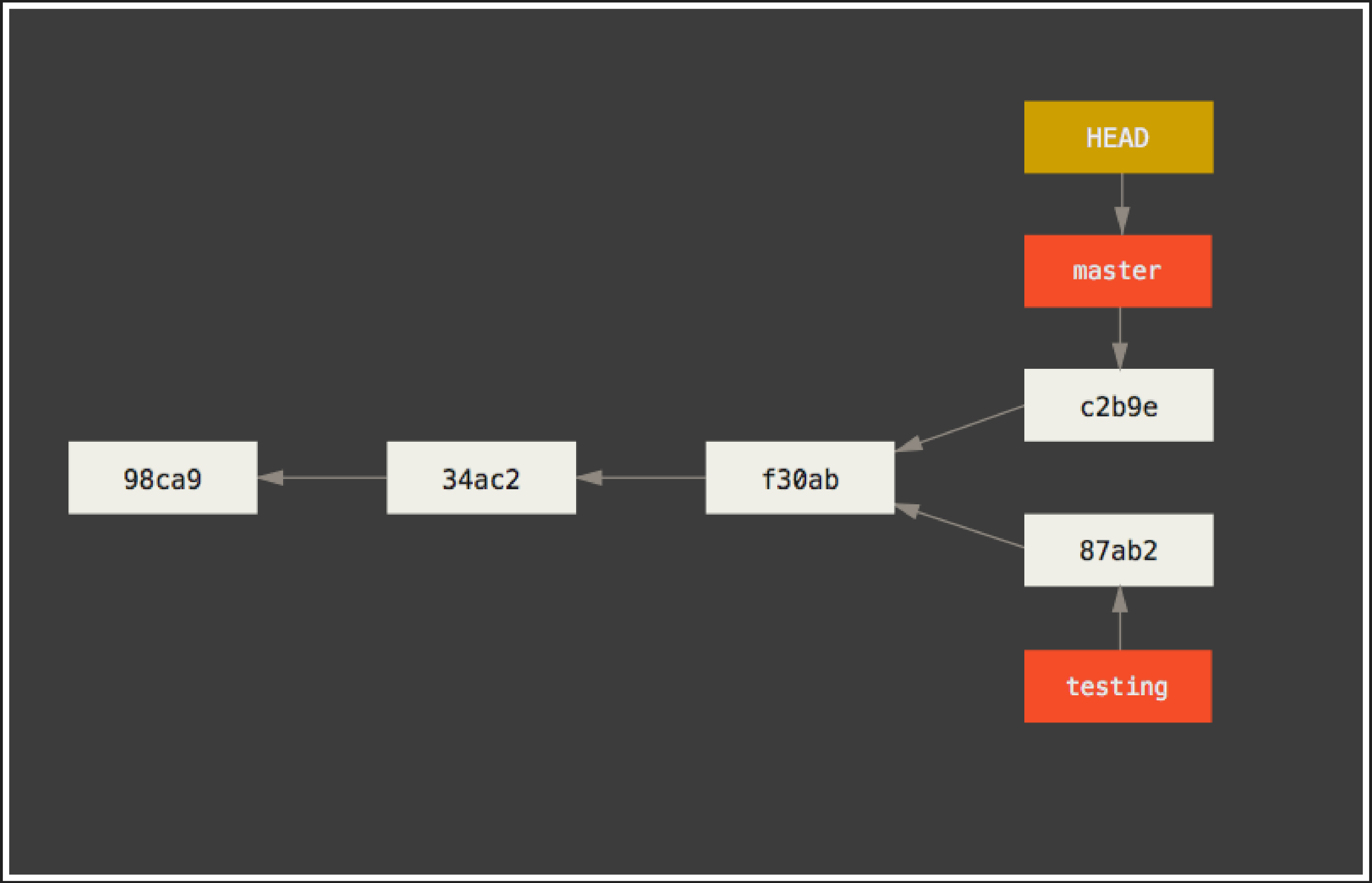
# ADVANCING BRANCHES



## DIVERGING MASTER

Let's go back to master and change something as well.

## DIVERGING MASTER



## BASIC BRANCHING AND MERGING

## SCENARIO

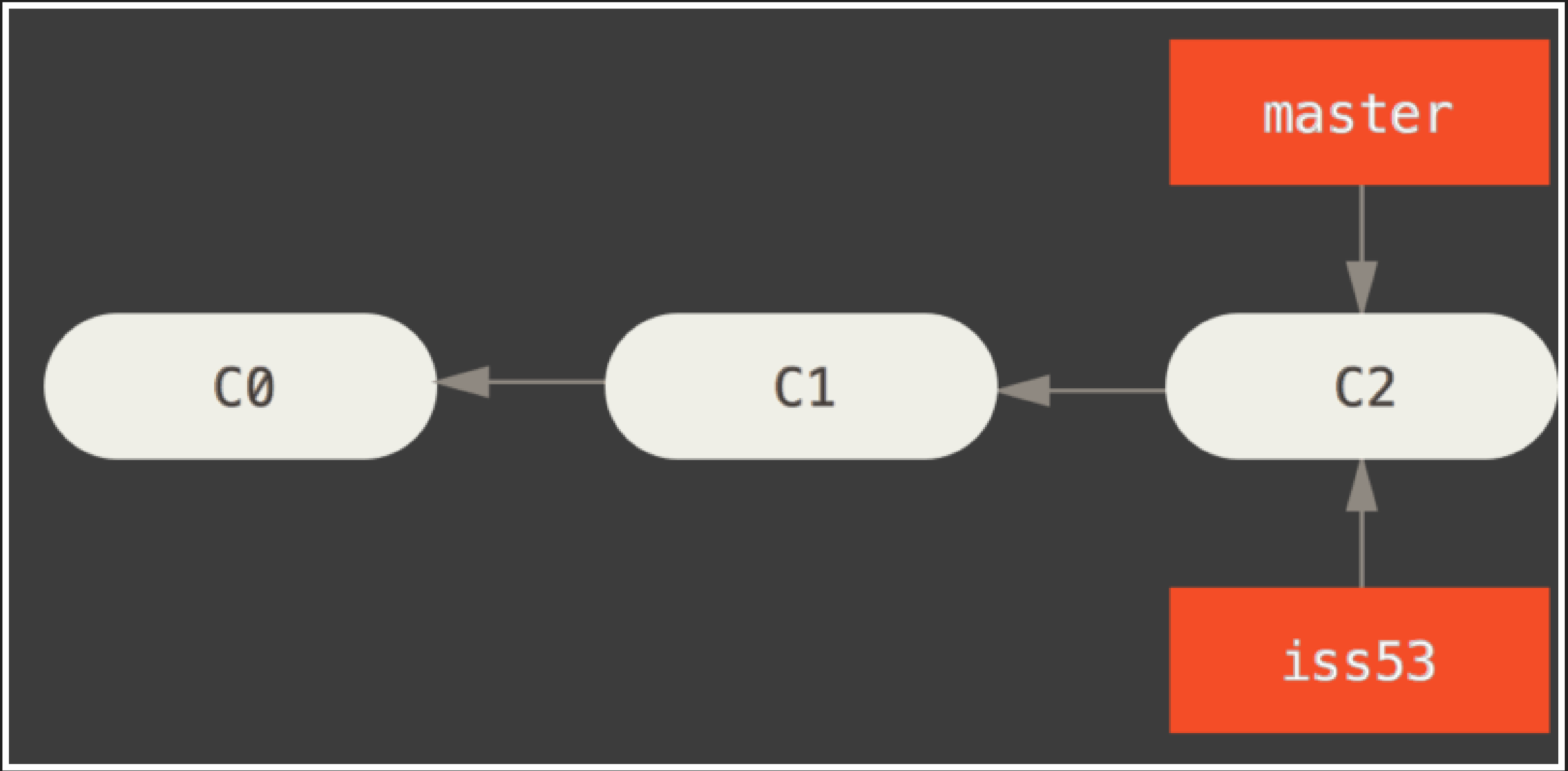
We're working on a website and we're writing a new story. Suddenly we receive a call about a critical issue that we have to fix.

## STEPS

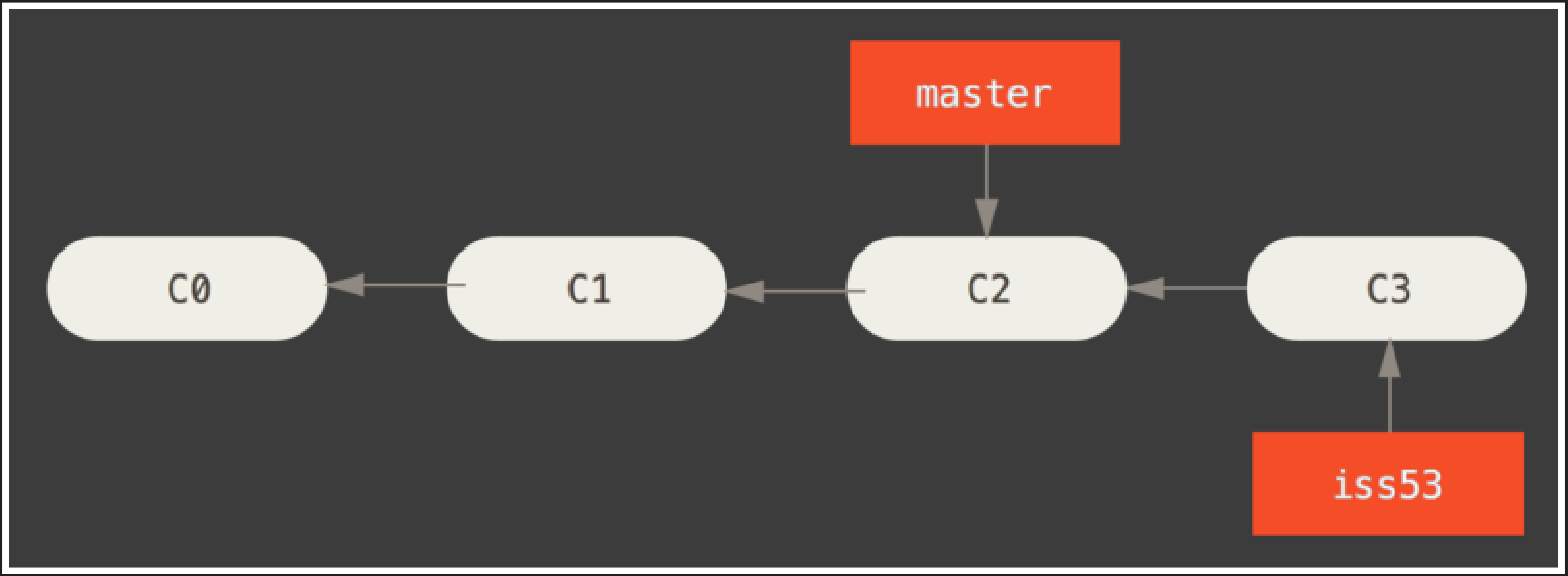
1. Create a new branch for a story
2. Start working in that branch
3. Switch to the production branch
4. Create a branch for the hotfix
5. Merge the hotfix branch and push to production
6. Switch back to our story branch and continue working
7. Merging our story

## A WORKED EXAMPLE

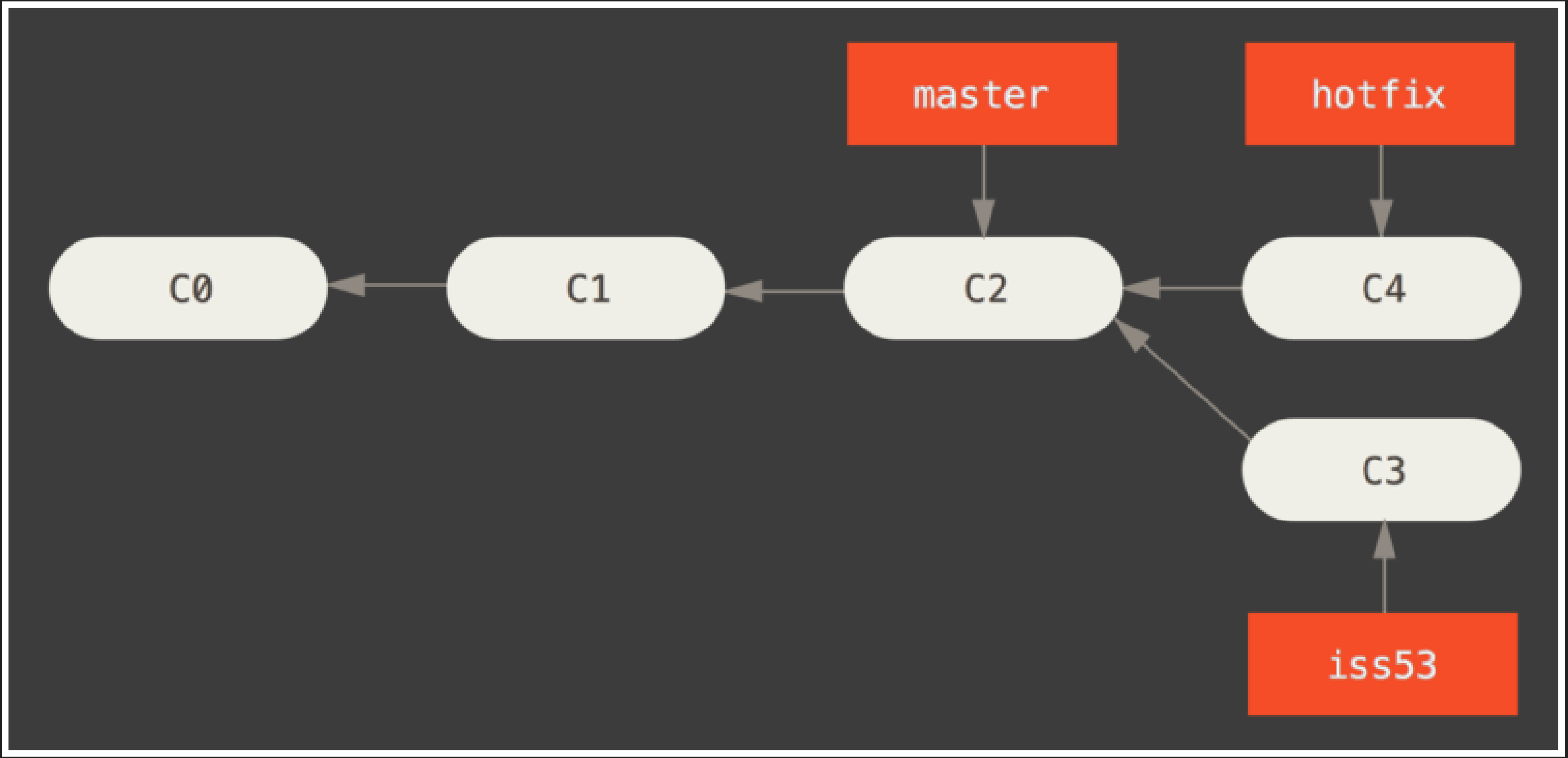
# NEW BRANCH



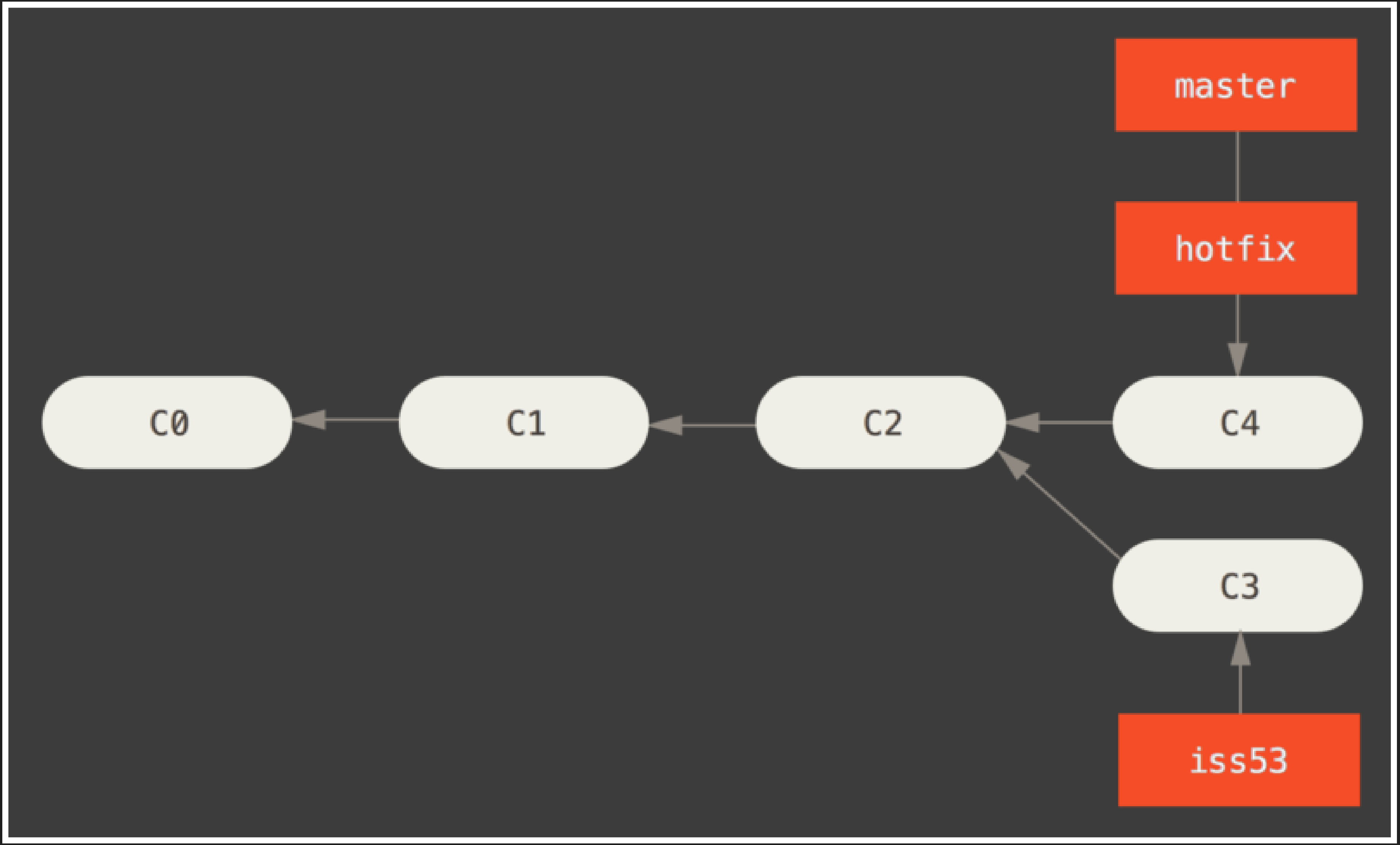
## COMMITING A POST



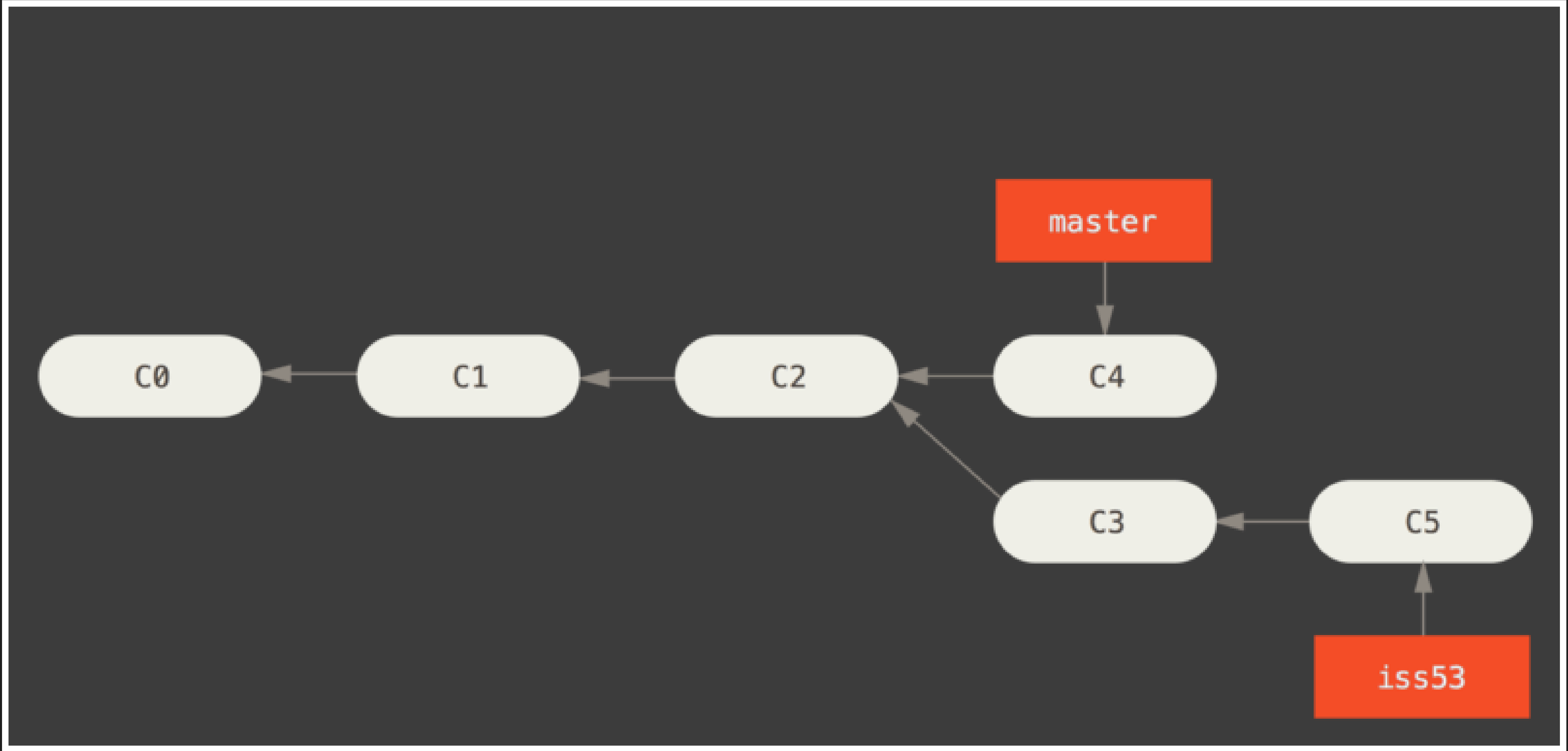
## CREATING THE HOTFIX BRANCH



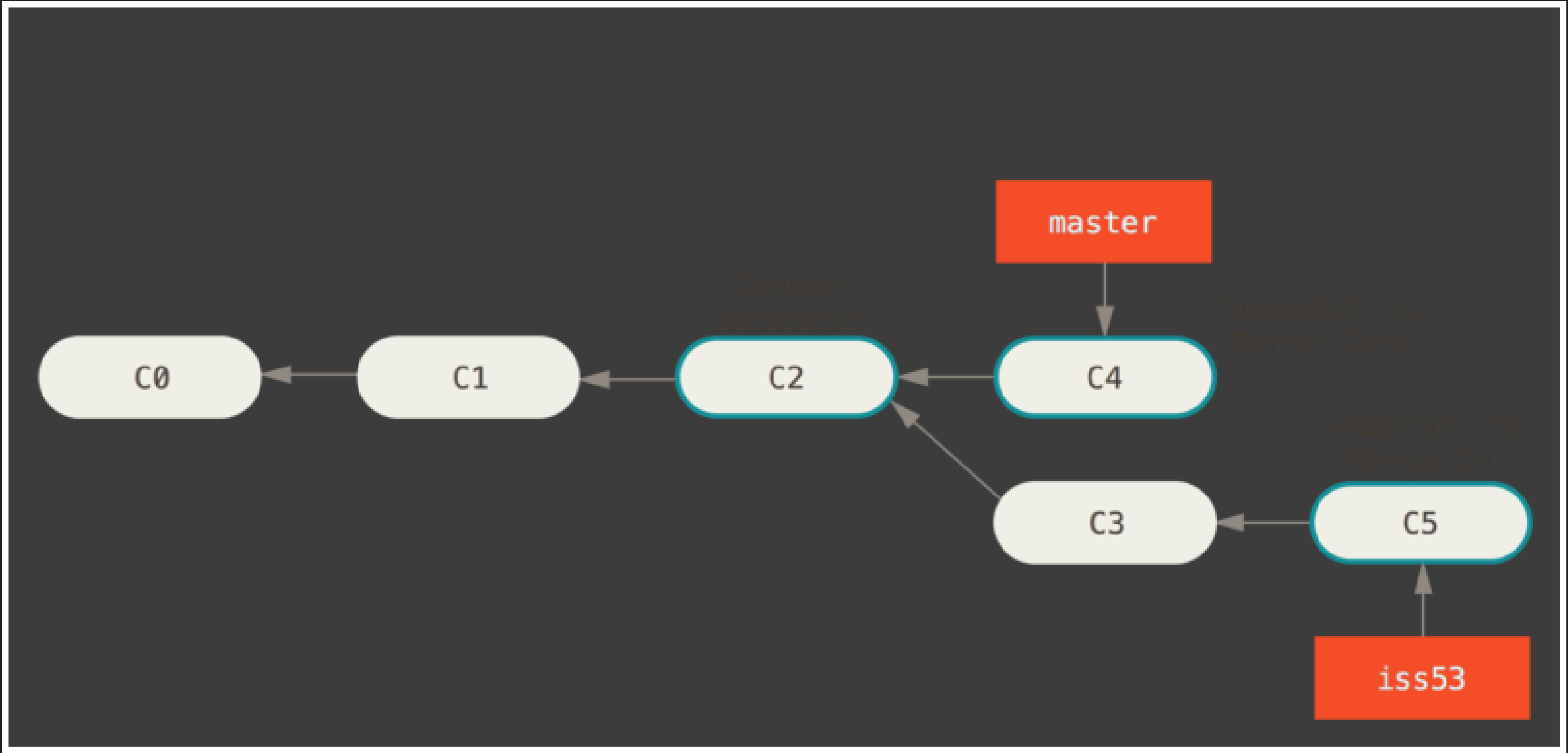
# MERGING HOTFIX BRANCH



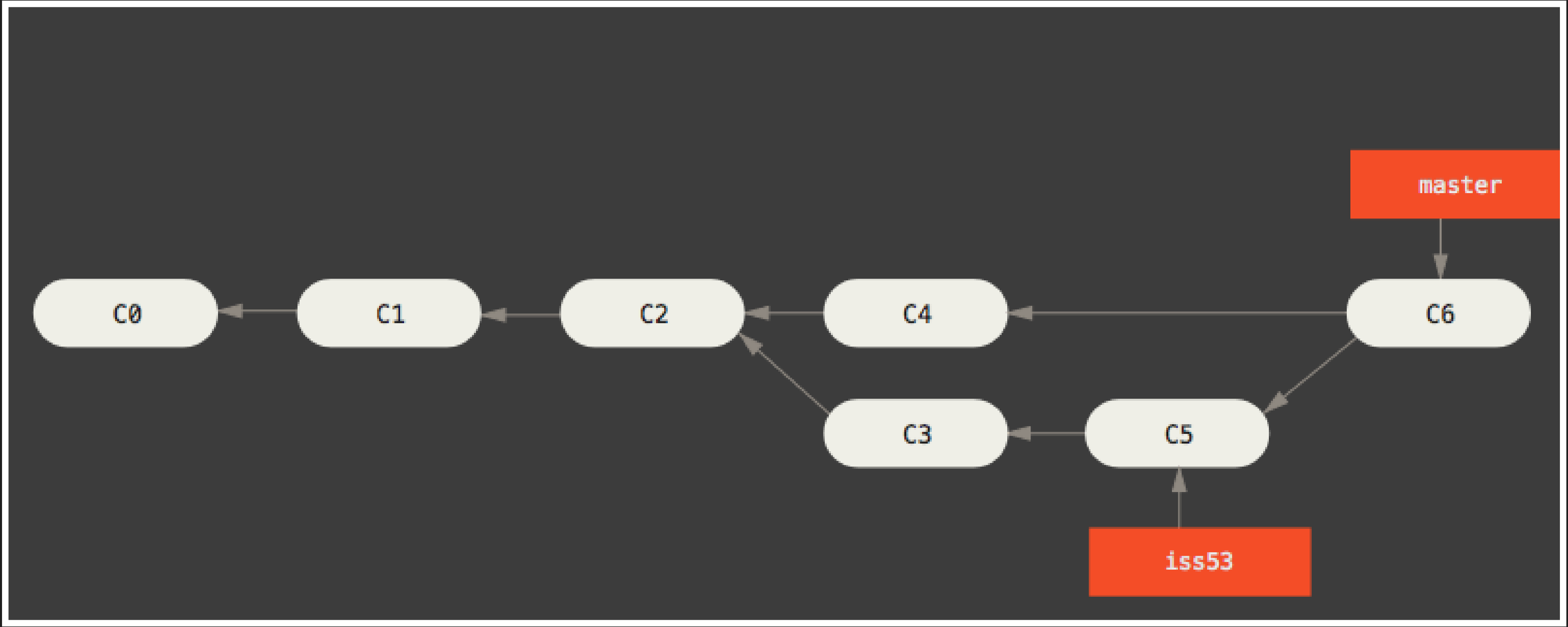
# FINISHING OUR POST



# MERGING OUR CHANGES



## MERGING OUR CHANGES



## CONFLICTS

Conflits happen more often than don't. First rule: don't panic.

EXAMPLE

## CUSTOMIZING GIT

## ALIASES

## NICE HISTORY

```
git config --global alias.glog log --graph --pretty=format:'%C
```

## SHORT STATUS

```
git config --global alias.s status -s
```

## LIST OF FILES THAT CHANGED IN A COMMIT

```
git config --global alias.lshow diff-tree --no-commit-id --nam
```

## LIST OF COMMITS THAT MODIFY A FILE

```
git config --global filelog log -u
```

## FINISHING UP

## QUESTIONS?

## FURTHER READING

1. Git documentation
2. Pro Git
3. Pro Git - Git Tools
4. Pro Git - Git in Other Environments
5. Git videos
6. External links

## THANKS FOR YOUR ATTENTION