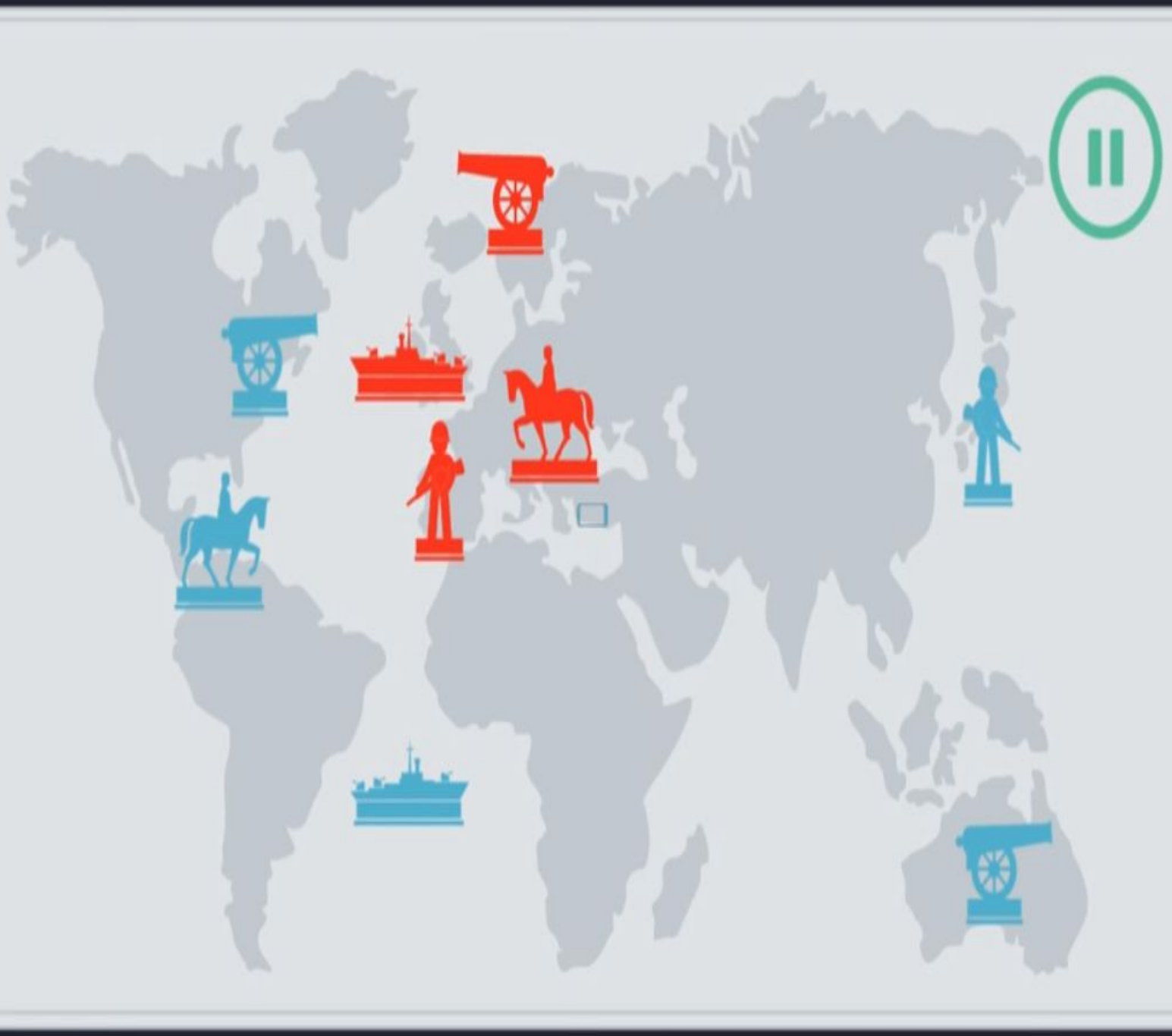
Git

Presentation by Pavlo Konietin



- **Terminology**

  Version Control System / Source Code Manager

  A **version control system** (abbreviated as **VCS**) is a tool that manages different versions of source code. A **source code manager** (abbreviated as **SCM**) is another name for a version control system.
- **Commit**

  Git thinks of its data like a set of snapshots of a mini filesystem. Every time you **commit** (save the state of your project in Git), it basically takes a picture of what all your files look like at that moment and stores a reference to that snapshot. You can think of it as a save point in a game - it saves your project's files and any information about them.
- **Repository / repo**

  A **repository** is a directory which contains your project work, as well as a few files which are used to communicate with Git. Repositories can exist either locally on your computer or as a remote copy on another computer. A repository is made up of commits.
- **Working Directory**

  The **Working Directory** is the files that you see in your computer's file system. When you open your project files up on a code editor, you're working with files in the Working Directory.

  This is in contrast to the files that have been saved (in commits!) in the repository.
- **Checkout**

  A **checkout** is when content in the repository has been copied to the Working Directory.
- **Staging Area / Staging Index / Index**

  A file in the Git directory that stores information about what will go into your next commit. You can think of the **staging area** as a prep table where Git will take the next commit. Files on the Staging Index are poised to be added to the repository.
- **SHA**

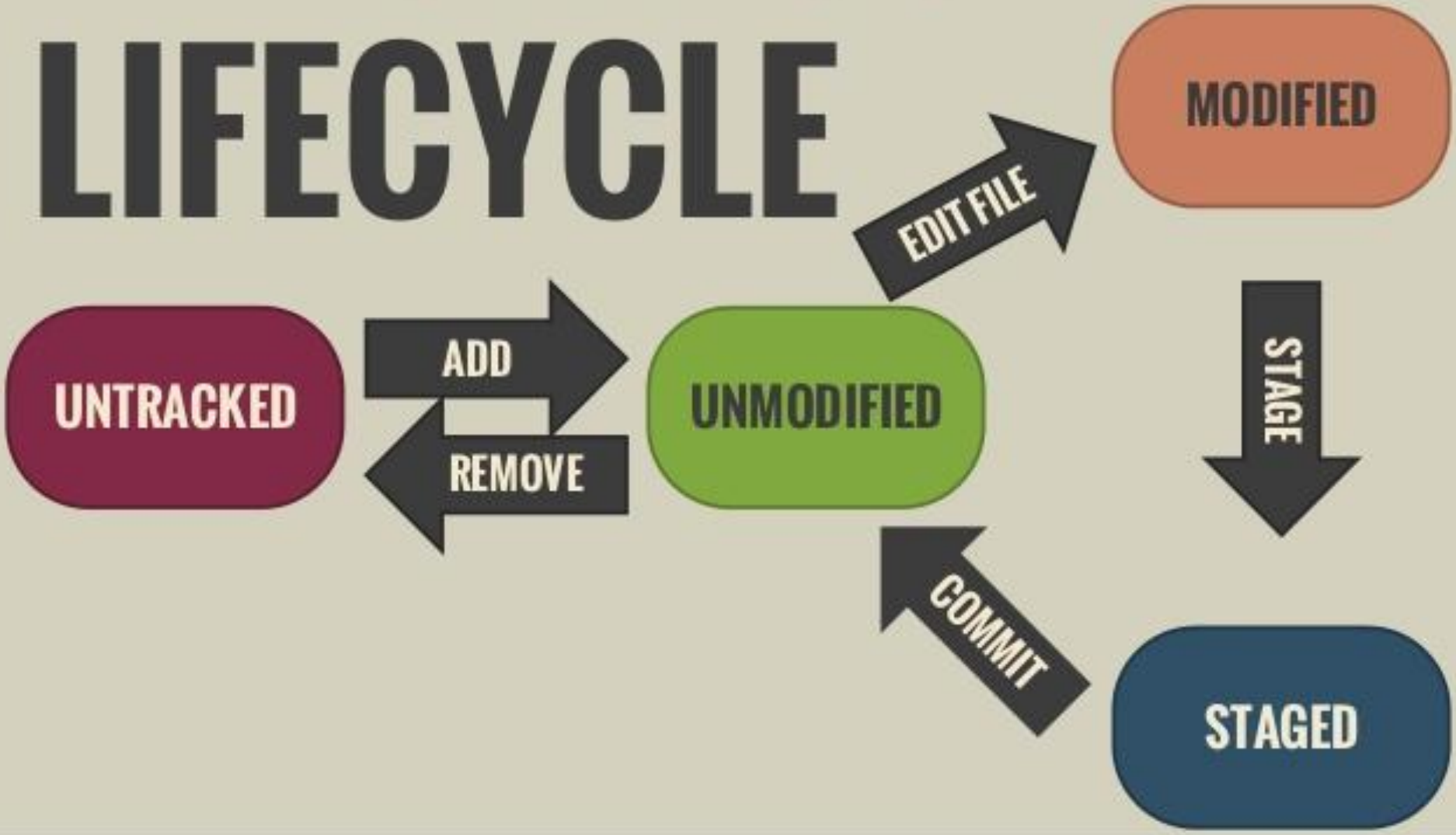
  A **SHA** is basically an ID number for each commit. Here's what a commit's SHA might look like: e2adf8ae3e2e4ed40add75cc44cf9d0a869afeb6.

  It is a 40-character string composed of characters (0–9 and a–f)
- A **branch**

  is when a new line of development is created that diverges from the main line of development. This alternative line of development can continue without altering the main line.

  Going back to the example of save point in a game, you can think of a branch as where you make a save point in your game and then decide to try out a risky move in the game. If the risky move doesn't pan out, then you can just go back to the save point. The key thing that makes branches incredibly powerful is that you can make save points on one branch, and then switch to a different branch and make save points there, too.

FILE STATUS LIFECYCLE




HTML
CSS
JS

Repository

Working Directory

Staging Index

Repository

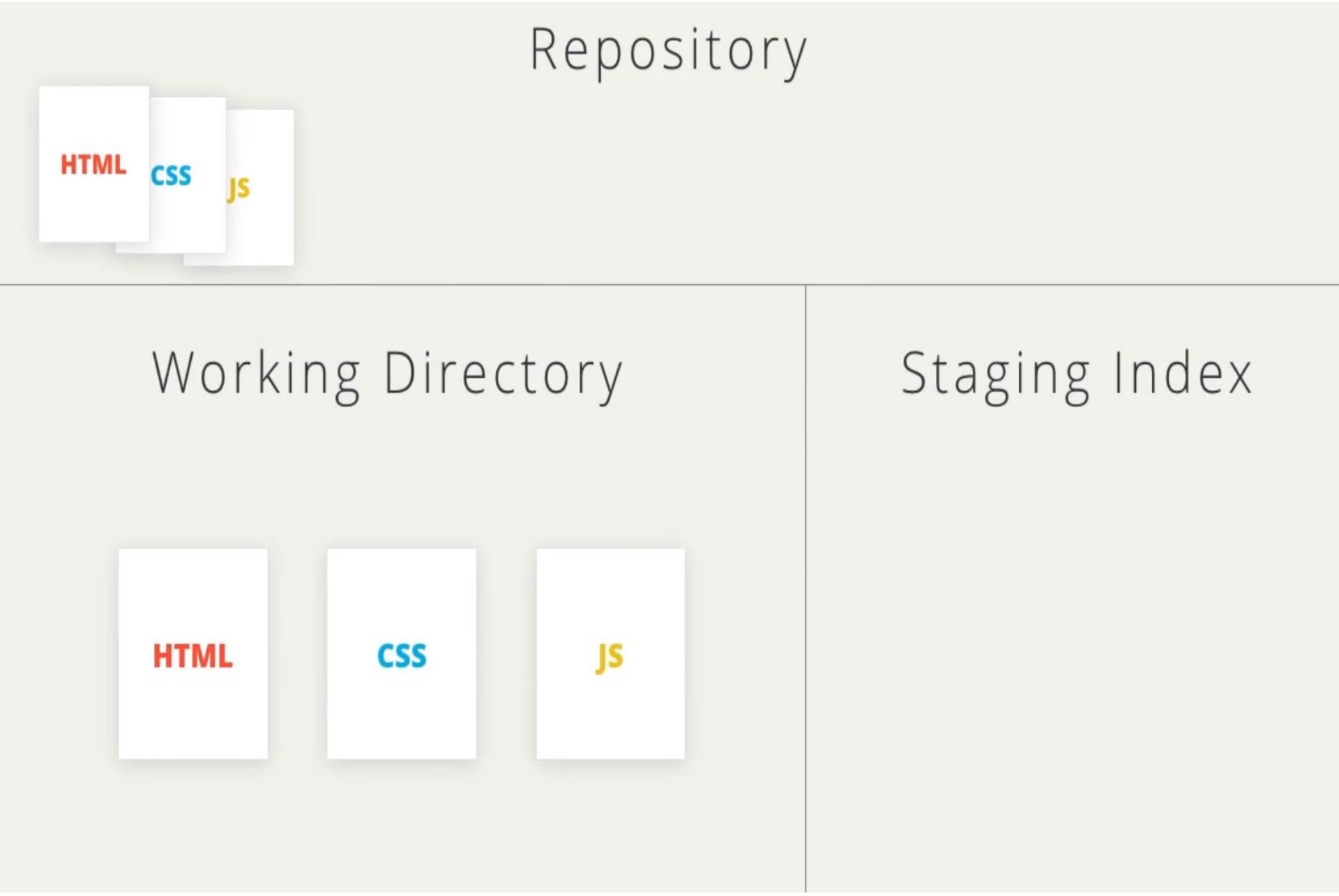
Working Directory

Staging Index

Repository

Working Directory

Staging Index


Repository
HTML
CSS
JS
Working Directory
HTML
CSS
JS
Staging Index

• Git Init's Effect

Running the git init command sets up all of the necessary files and directories that Git will use to keep track of everything. All of these files are stored in a directory called .git (notice the . at the beginning - that means it'll be a hidden directory). This .git directory *is the "repo"*! This is where git records all of the commits and keeps track of everything!

WARNING: Don't directly edit any files inside the .git directory. This is the heart of the repository. If you change file names and/or file content, git will probably lose track of the files that you're keeping in the repo, and you could lose a lot of work! It's okay to look at those files though, but don't edit or delete them.

- **config file** - where all *project specific* configuration settings are stored.
- Git looks for configuration values in the configuration file in the Git directory (.git/config) of whatever repository you're currently using. These values are specific to that single repository.
- For example, let's say you set that the global configuration for Git uses your personal email address. If you want your work email to be used for a specific project rather than your personal email, that change would be added to this file.
- **description file** - this file is only used by the GitWeb program, so we can ignore it
- **hooks directory** - this is where we could place client-side or server-side scripts that we can use to hook into Git's different lifecycle events
- **info directory** - contains the global excludes file
- **objects directory** - this directory will store all of the commits we make
- **refs directory** - this directory holds pointers to commits (basically the "branches" and "tags")

```
$ git status
```

The git status is our key to the mind of Git. It will tell us what Git is thinking and the state of our repository as Git sees it. When you're first starting out, you should be using the git status command *all of the time*! Seriously. You should get into the habit of running it after any other command. This will help you learn how Git works and it'll help you from making (possibly) incorrect assumptions about the state of your files/repository.

```
$ git status
On branch master

Initial commit

nothing to commit (create/copy files and use "git add" to track)
```

Next command to show:

- git log
- git log -- oneline
- git log -- stat
- git log -p
- git show 6453824 (--stat, -p)

Add commits to A Repo

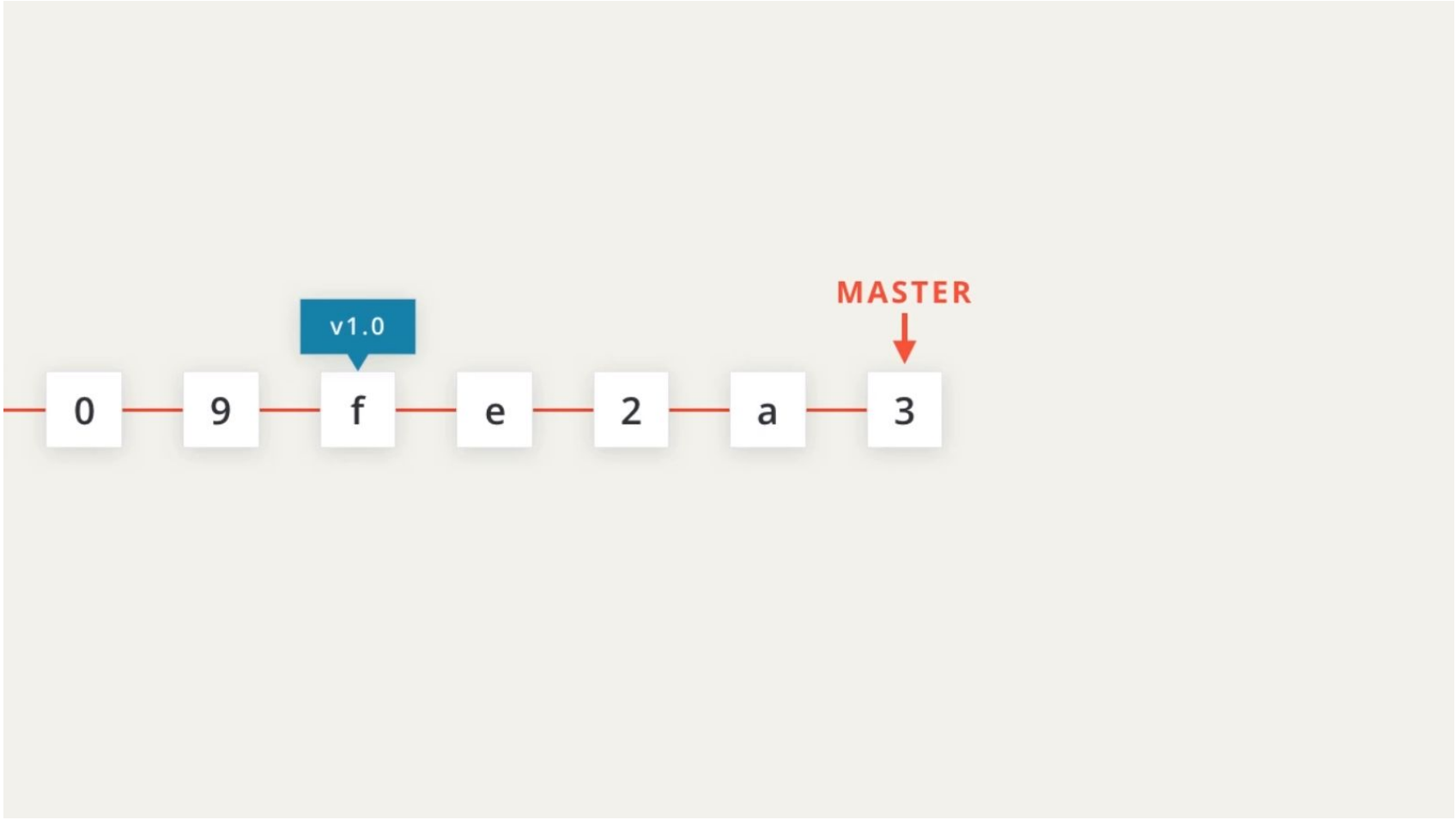
In those section we will look throw some git commands that will help us to add changed files from :

- git add “Working directory” to “Staging index”
- git commit “Staging index” to “Repository”
- git diff displays the difference between to versions of file

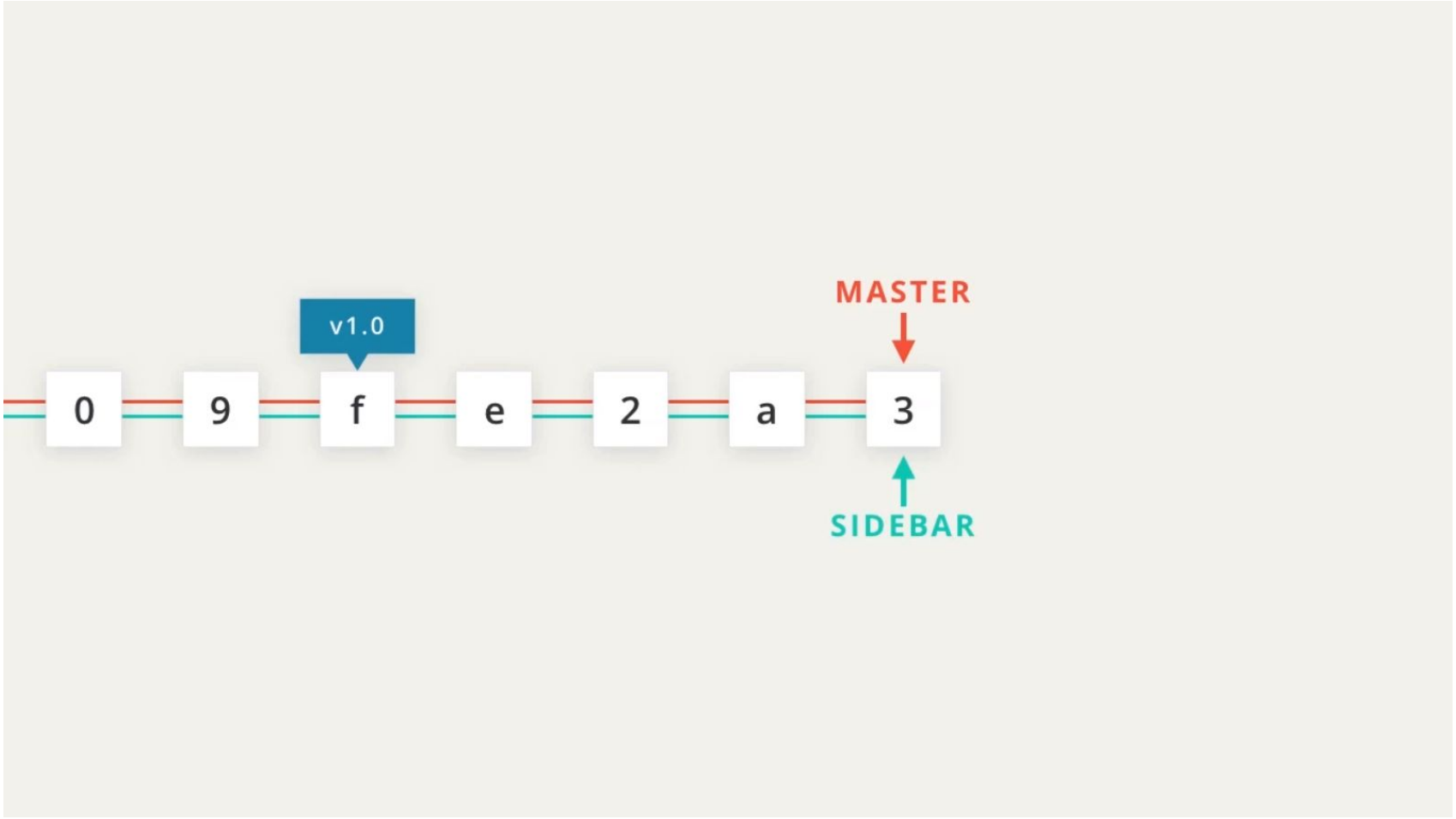
And now Version Control SUPER POWERS

- git branch - allows multiple lines of development
- git checkout - switch between different branches
- git merge - combines changes on different branches

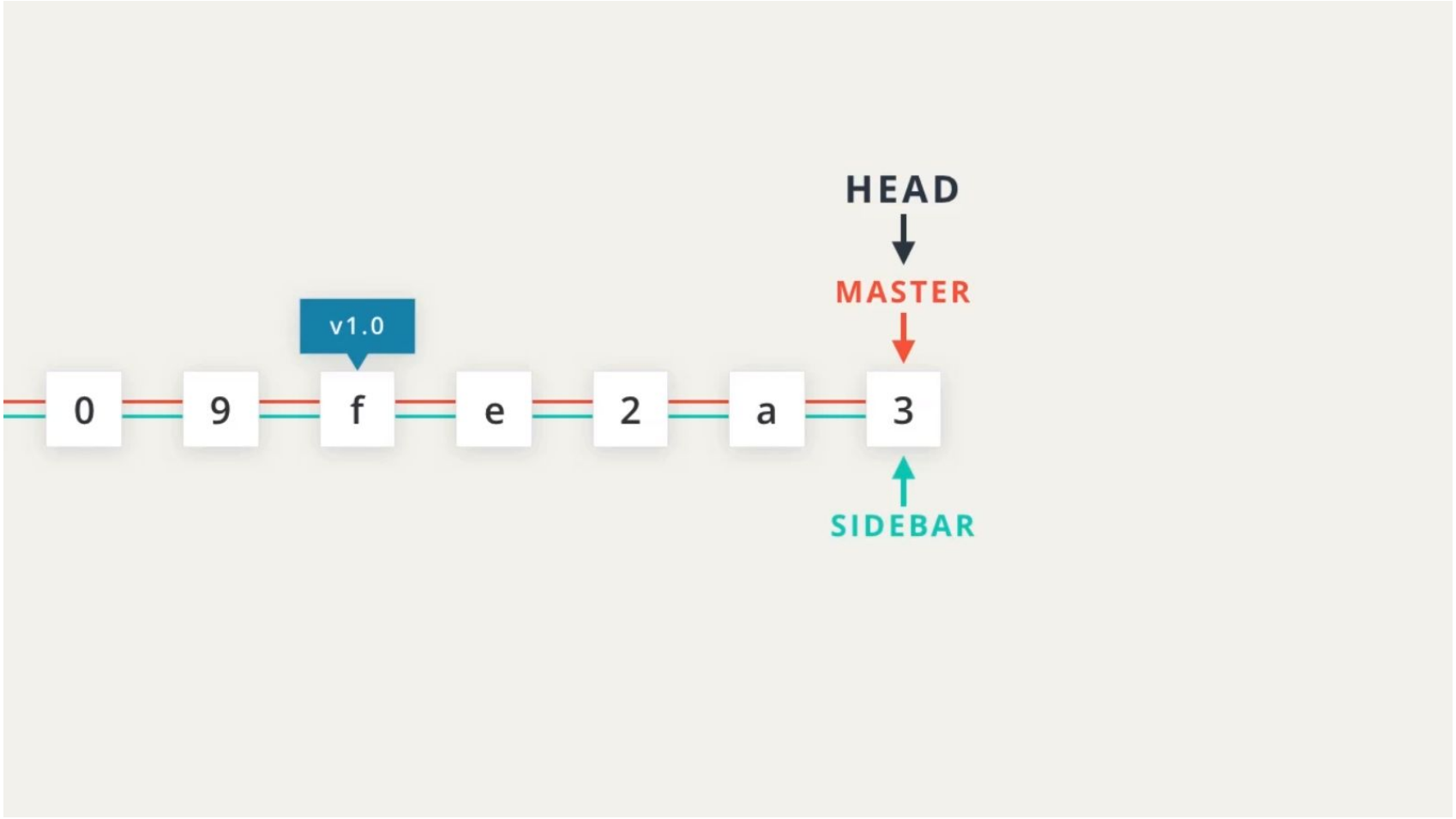
Branching

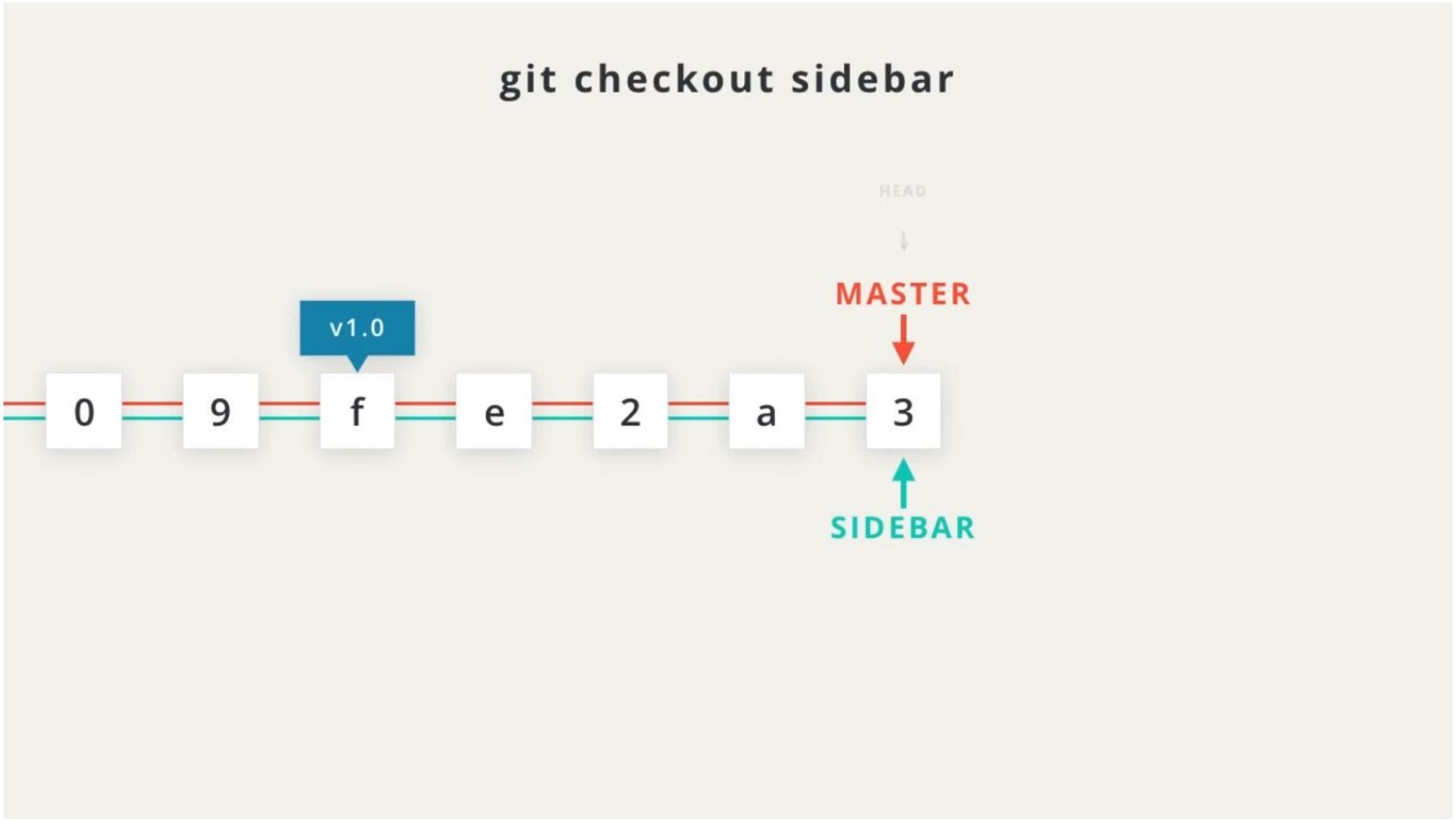


Another branch

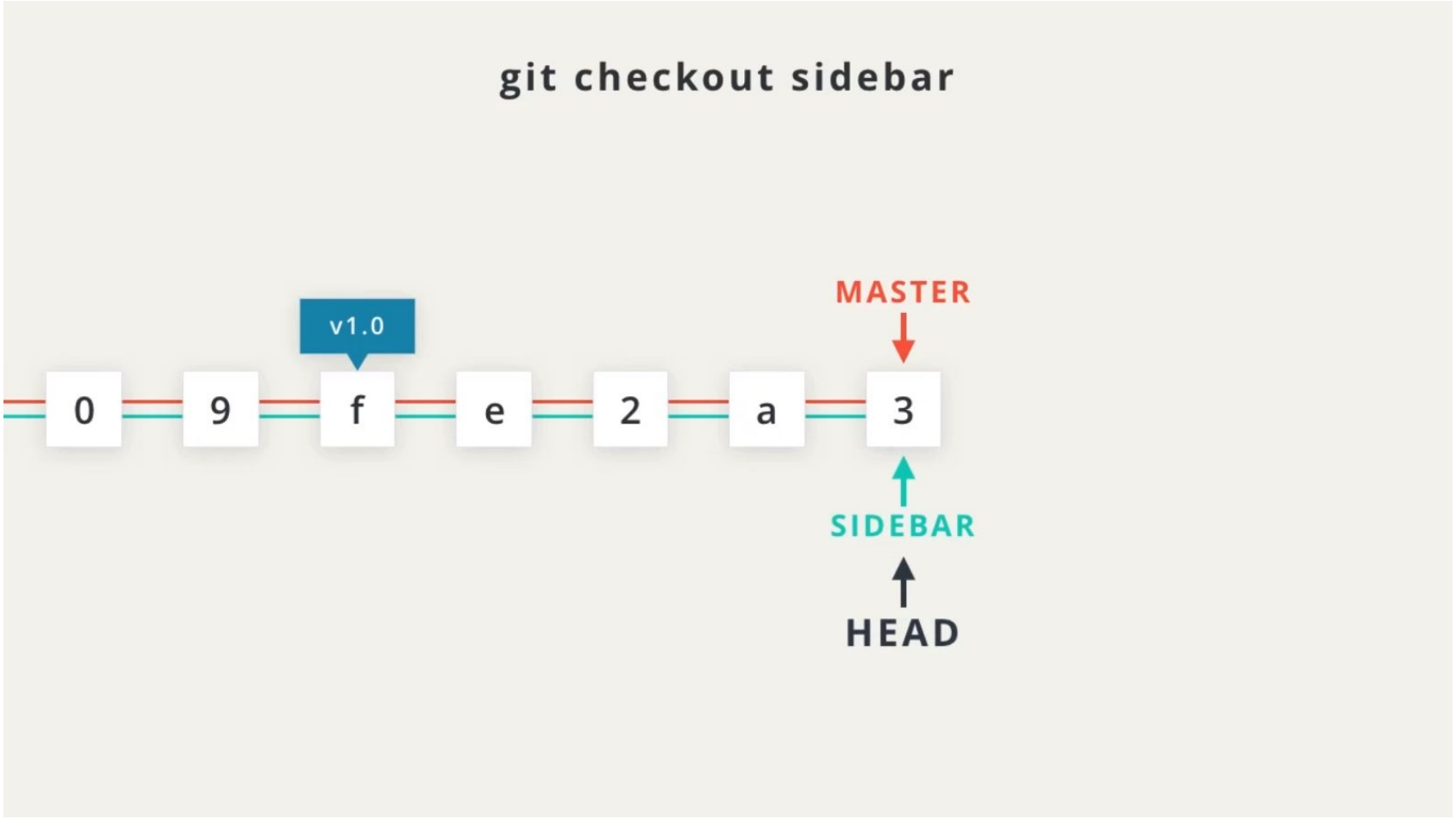


Head points to the branch that is active

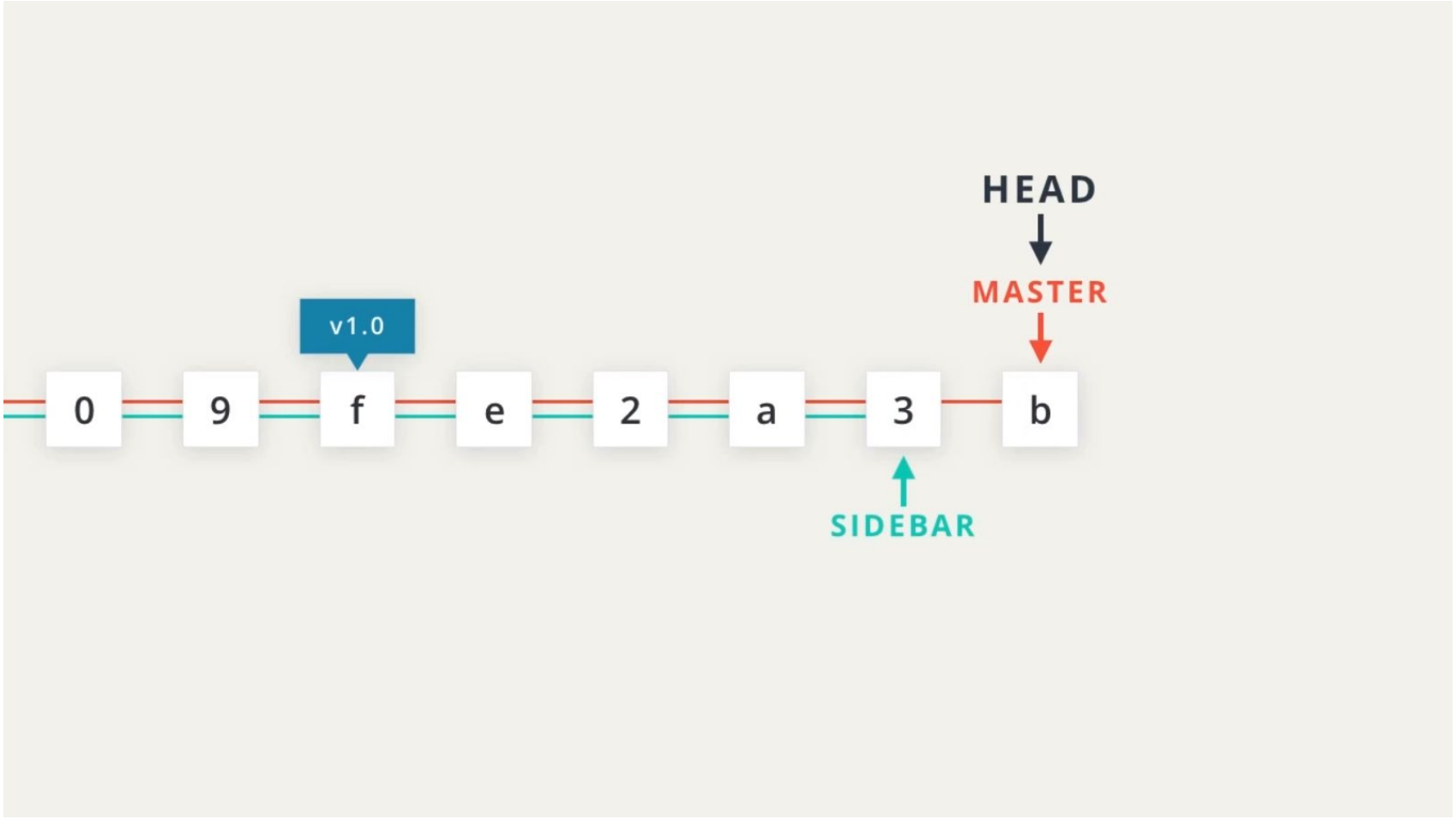



git checkout sidebar
HEAD
MASTER
v1.0
0
9
f
e
2
a
3
SIDEBAR

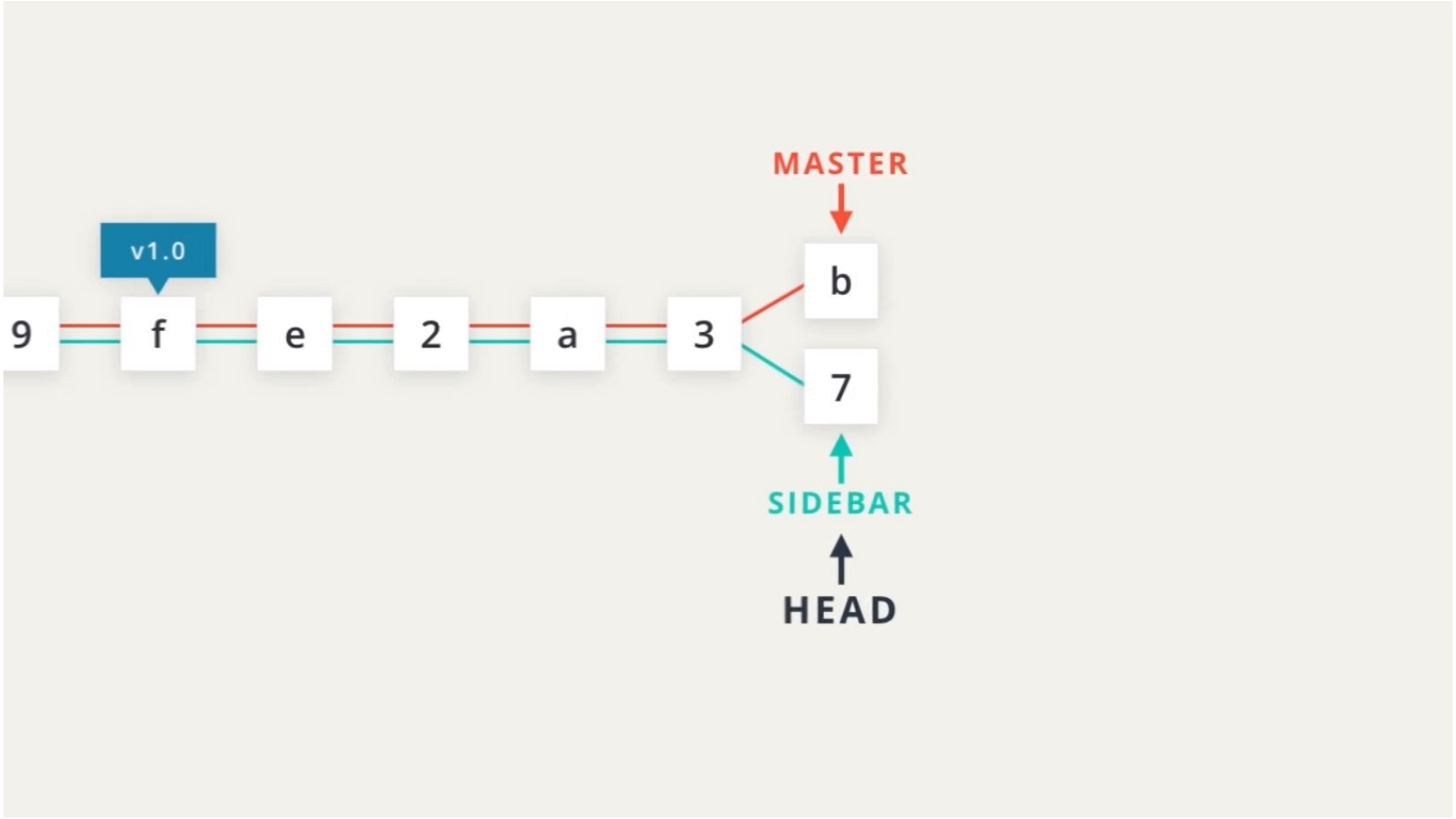
Lets switch to the another branch



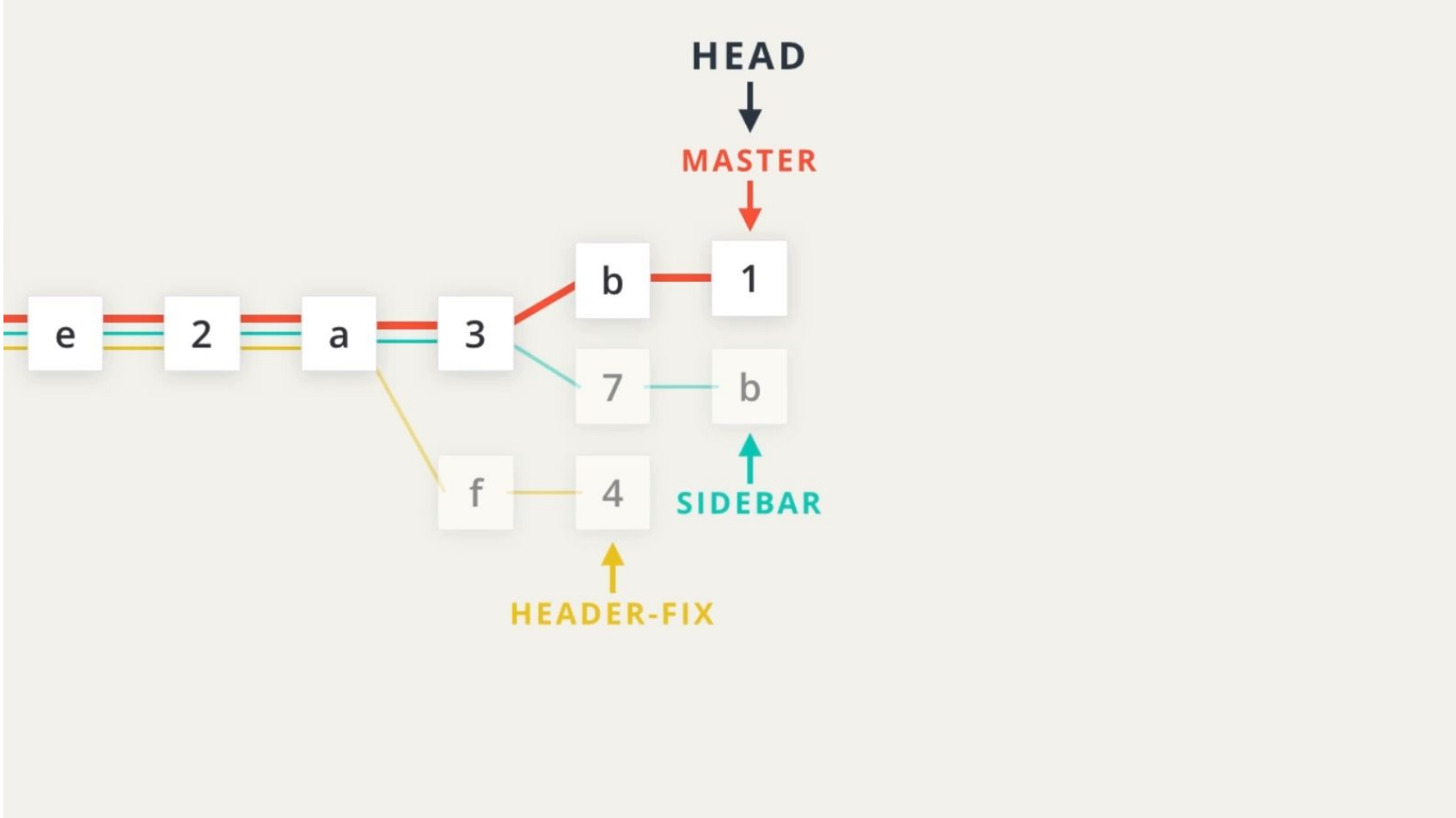
We commit into master



Switched to another branch and commit there



Some more commits. And new branch with some fix



More detailed about git branch command

```
$ git branch
```

The git branch command is used to interact with Git's branches:

- list all branch names in the repository
- create new branches
- delete branches

1) If we type out just git branch it will list out the branches in a repository

2) To create a branch, all you have to do is use git branch and provide it the name of the branch you want it to create. So if you want a branch called "sidebar", you'd run this command: git branch sidebar

Don`t forget to CHECK

Remember that when a commit is made that it will be added to the current branch. So even though we created the new sidebar, no new commits will be added to it since we haven't *switched to it*, yet. If we made a commit right now, that commit would be added to the master branch, *not* the sidebar branch. We've already seen this in the demo, but to switch between branches, we need to use Git's checkout command.

```
$ git checkout sidebar
```

will:

- remove all files and directories from the Working Directory that Git is tracking
  - (files that Git tracks are stored in the repository, so nothing is lost)
- go into the repository and pull out all of the files and directories of the commit that the branch points to