



little languages

lecture 15:

fundamentals of git

Respond to git survey:
<http://bit.ly/590-19s-git>
Start-up the VM!
No lecture directory today.

Respond to **git** survey:
<http://bit.ly/590-19s-git>
Start-up the VM!
No lecture directory today.

Following Along (in home directory on VM)

```
git clone https://github.com/KrisJordan/git-demo.git
```

```
cd git-demo
```

What is a Version Control System (VCS)?

- Non-trivial software projects involve *lots* of files
- Not only that, but there are lots of *dependent* files
- Suppose you have a class defined in one file and change the name of one of its fields or methods... *every dependent file needs to be updated too*
- When you are working on a project, you will want to play with various changes and refactorings

Why **git** over another VCS?

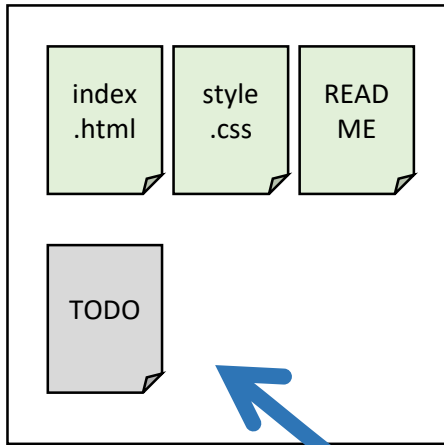
- Initially developed in 2005 by Linus Torvalds, creator of Linux, to be the version control system *for* the Linux operating system's code.
- In the last decade, **git** won out as the de facto VCS of engineers.
 - Previously: SVN (Subversion 2000) and CVS (Concurrent Versions System 1990)
 - Contemporary: Mercurial (2005)
- Why did **git** win?
 - It's fast... remarkably performant compared to prior VCS systems.
 - It's distributed... everyone has a project's *complete* history, no internet needed.
 - It's immutable by default... you have to try *really* hard to mutate existing commits.
 - It's append-only... you have to try *really* hard to accidentally delete old work.
 - It's robust... it ensures integrity of all data to avoid corruptions.

What is GitHub versus `git`?

- `git` is Version Control System software you install and use locally
- GitHub is a social web site for sharing and collaborating on projects whose source code is maintained with the `git` VCS
- You can use `git` without using GitHub, but not vice-versa.

Suppose you're working on a web site...

Project Working Directory



This is your project's **working directory**.

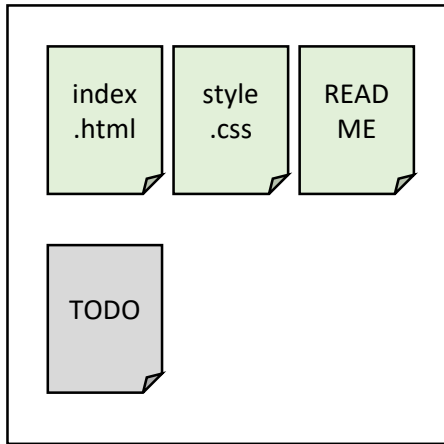
Its README file describes the project.

Its index.html and style.css files are the important files you're working on.

The TODO file is a personal text file you're keeping to yourself, independent of team.

... and that its history is maintained by a **git repository**.

Project Working Directory



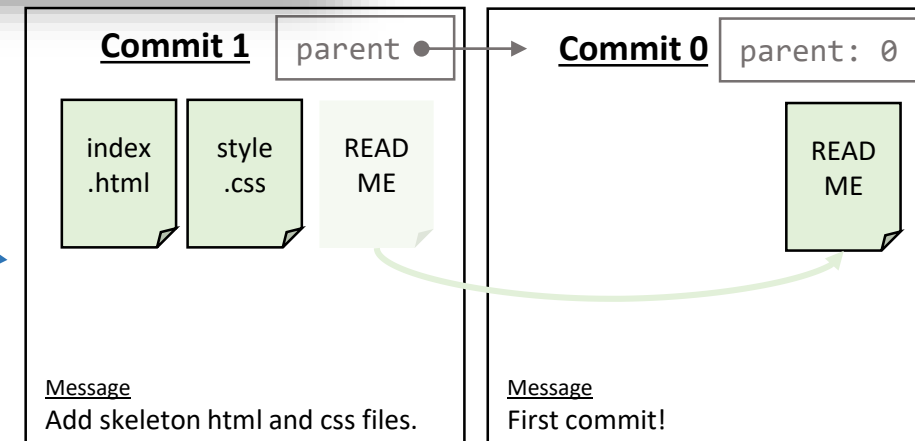
The .git Repository

This is your project's **commit history**!

What is a commit? It's a snapshot, or backup, of your project's files at a specific moment in your project's history.

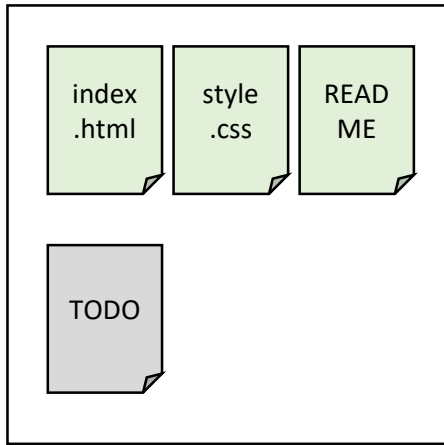
How did these commits get here? For now, assume your teammate added them. Soon, you'll make a commit!

Where is this data stored? In the `.git/` folder in your project.



Commits are linked to their parents or "previous version".

Project Working Directory

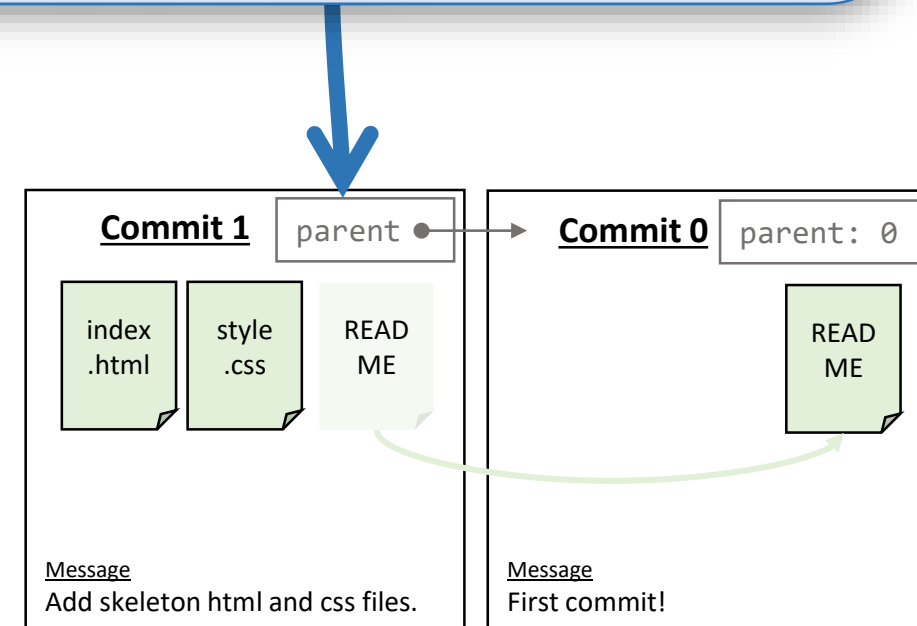


The .git Repository

Notice the second commit holds a reference to its **parent**, or "previous" commit.

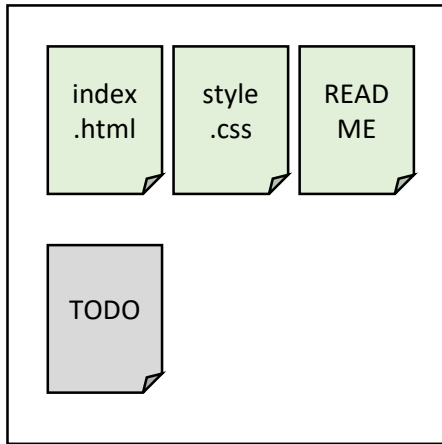
You can imagine the history of commits in a very simple project like a linked list*.

* This reference is called parent, though, because once a repository makes use of branches it is more like a tree than a list.



A commit contains only important* file changes.

Project Working Directory

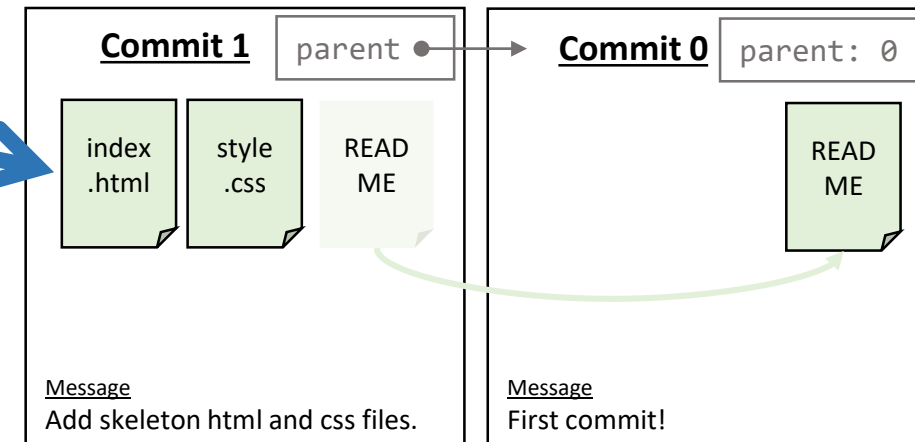


The .git Repository

Suppose index.html and style.css were added to the project for this commit. No changes to README.

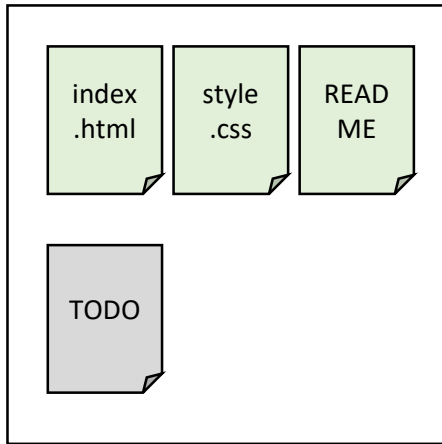
We're illustrating those two files were added/changed in this commit, but README was unchanged and the previous version still holds.

What is an **important** file change? **Any file the person making the commit decides to add to the commit.** Notice, TODO was not included.



Each commit has a message and other metadata.

Project Working Directory

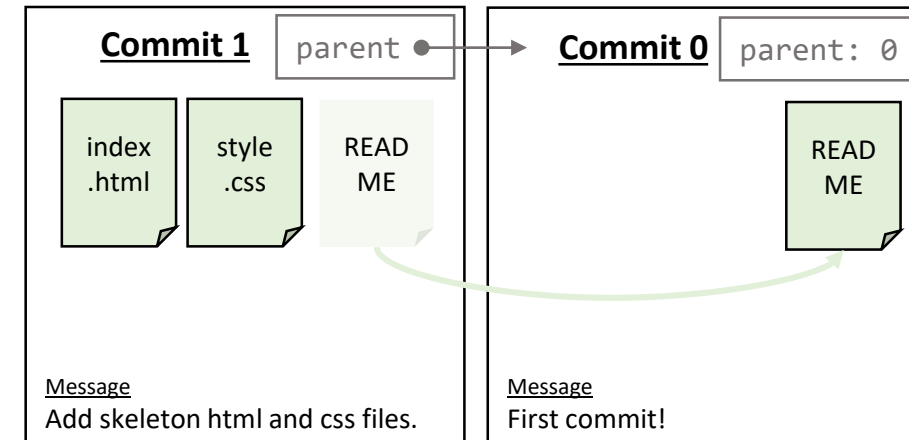


The .git Repository

Each commit has a **message** describing what is important about this specific commit in the project's history.

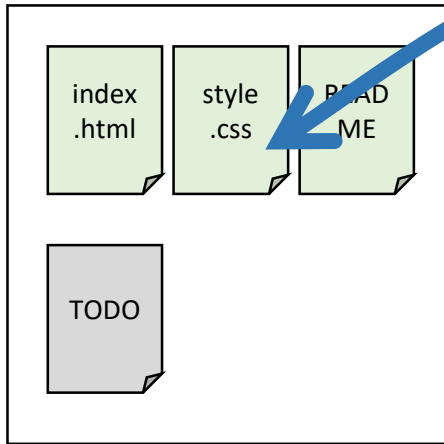
The person who makes a commit must write this message. **When you make a commit you should write an informative description!**

Each commit also has a timestamp and author name/e-mail.



What's the big deal?

Project Working Directory

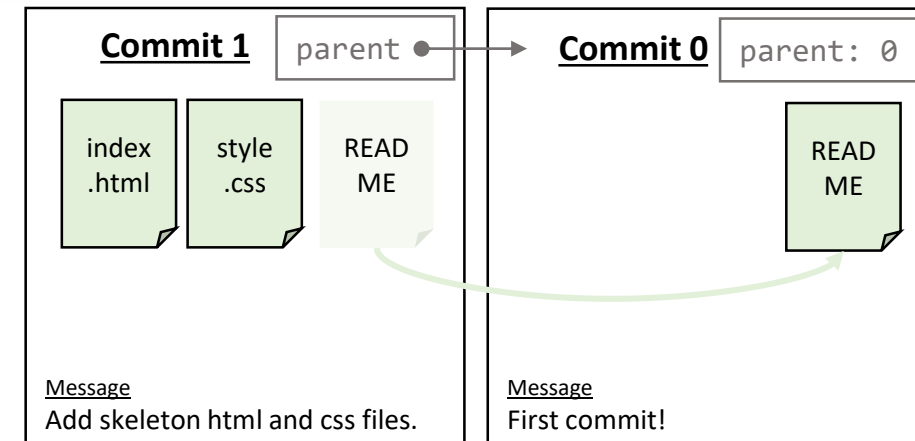


The .git Repository

Suppose you deleted a lot of work in style.css, saved, and then realized you needed something deleted...

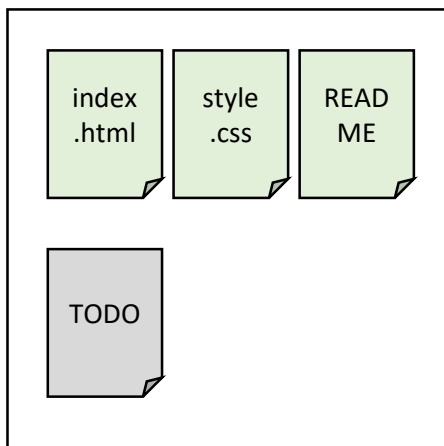
...with your history in git, it's easy to checkout a committed version of this file without fear of loss.

You can also restore all files in a project back to a specific commit in its history.



How do you get *your* changes into a git repository?

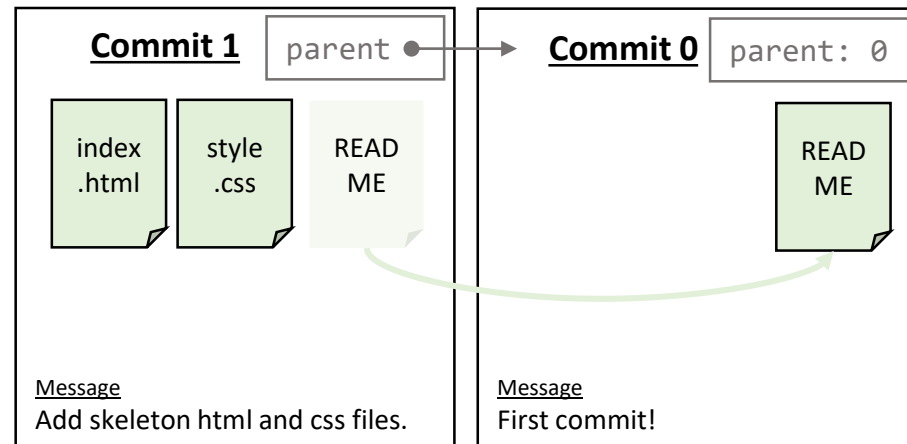
Project Working Directory



The .git Repository

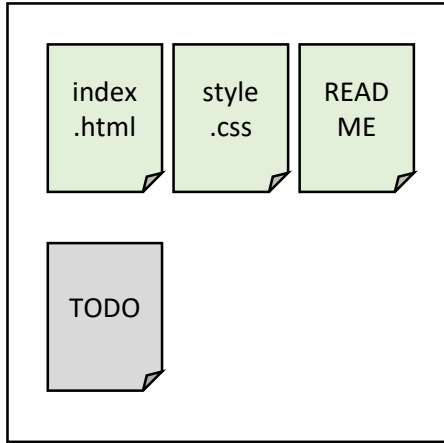
Imagine you've made changes to style.css and are ready to commit those changes to the history of the project.

How do you make a commit?

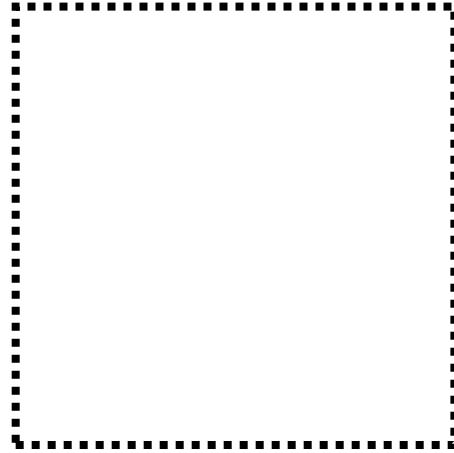


First, you have to tell git which file changes are important.

Project Working Directory



Staging Area



The .git Repository

The first step in making a commit is staging your important file changes to the **staging area**.

The data in your staging area is stored behind the scenes in your .git repository.

Message

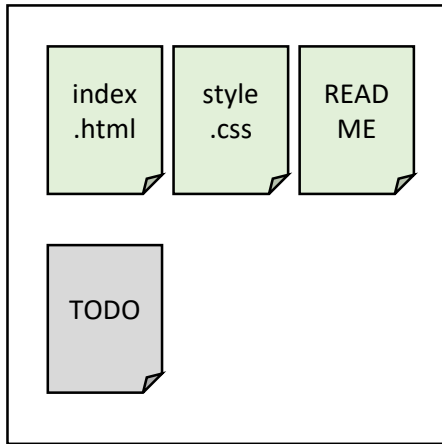
Add skeleton html and css files.

Message

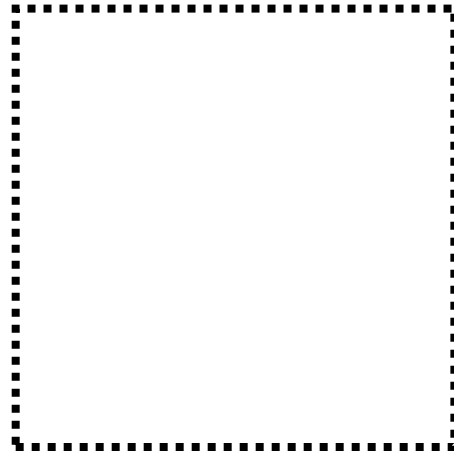
First commit!

Checking the status of your staging area

Project Working Directory



Staging Area



The .git Repository

git status

When `git status` responds with:

nothing to commit, working tree clean

It means there is nothing staged for a commit.

Message

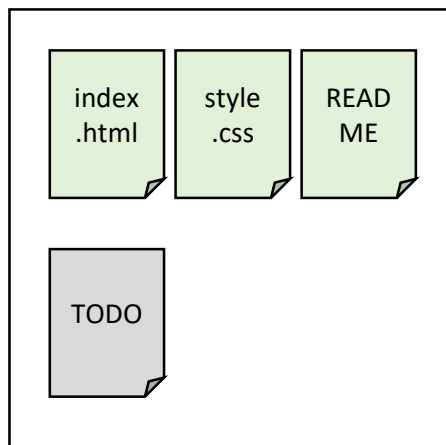
Add skeleton html and css files.

Message

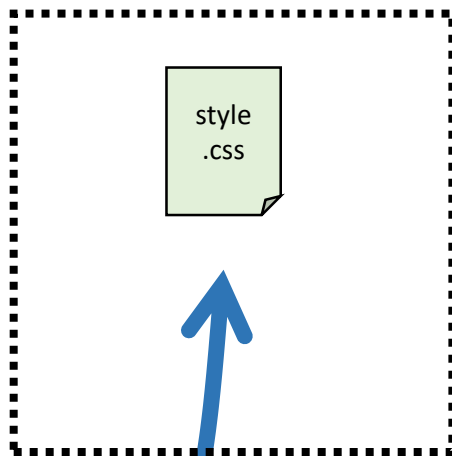
First commit!

Add a file to your staging area.

Project Working Directory



Staging Area



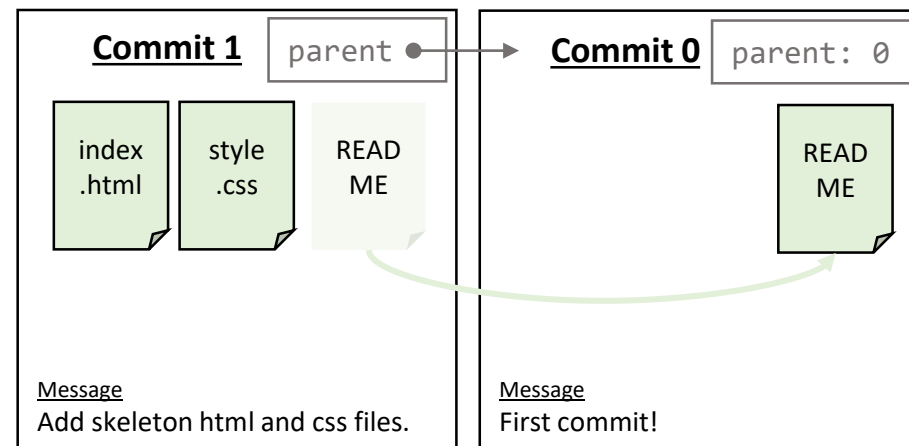
The .git Repository

```
git add style.css
```

```
git status
```

The only important file we want to stage is style.css, so **we use the git add command (above) to add it to the staging area.**

We specifically don't want our personal TODO file in this commit and you could imagine small, tinkering changes to index.html which aren't important to the project either.



Make a commit.

Project Working Directory

The .git Repository

Staging Area

```
git commit -m 'Improve styling.'
```

The **git commit** command takes your staging area and transforms it into a commit.

Notice, git takes care of establishing the parent link and links to previous versions of files unchanged in this commit.

Commit 2

parent •

index .html
style .css
READ ME

Message
Improve styling.

Commit 1

parent •

index .html
style .css
READ ME

Message
Add skeleton html and css files.

Commit 0

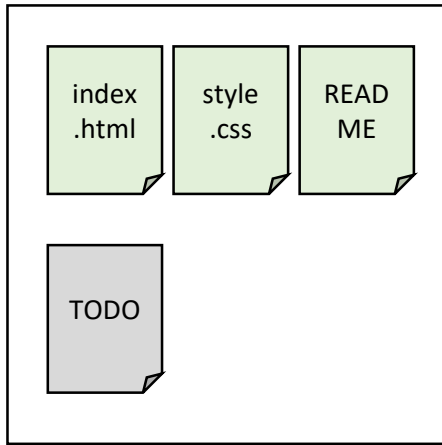
parent: 0

READ ME

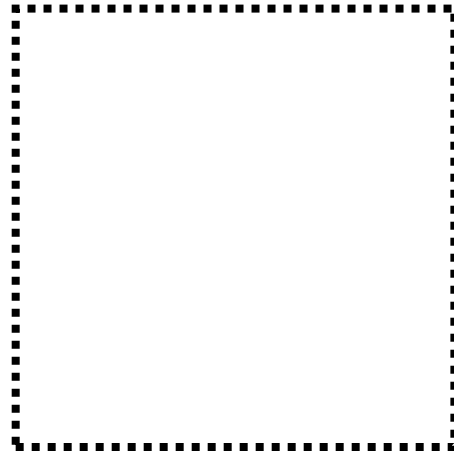
Message
First commit!

After a commit, your staging area is cleared.

Project Working Directory

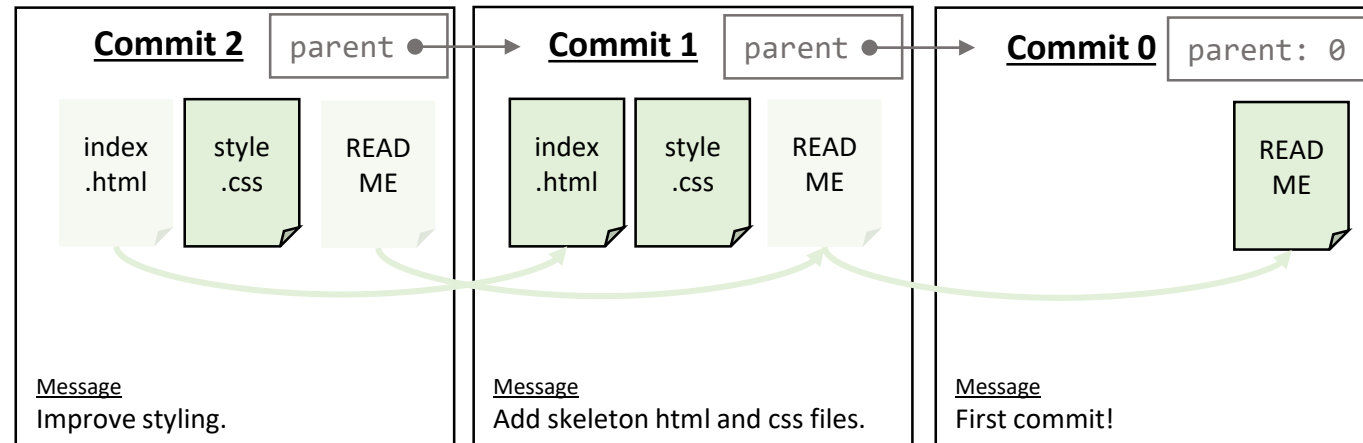


Staging Area



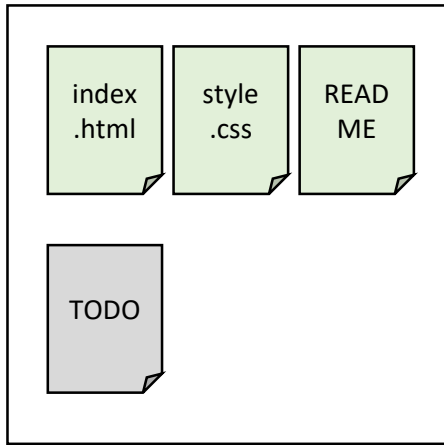
The .git Repository

git status

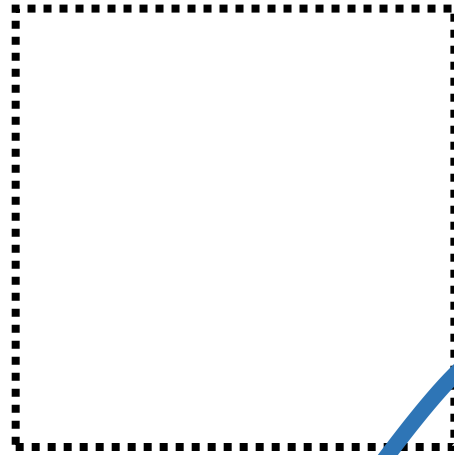


git maintains a special kind of reference called a **branch**

Project Working Directory



Staging Area

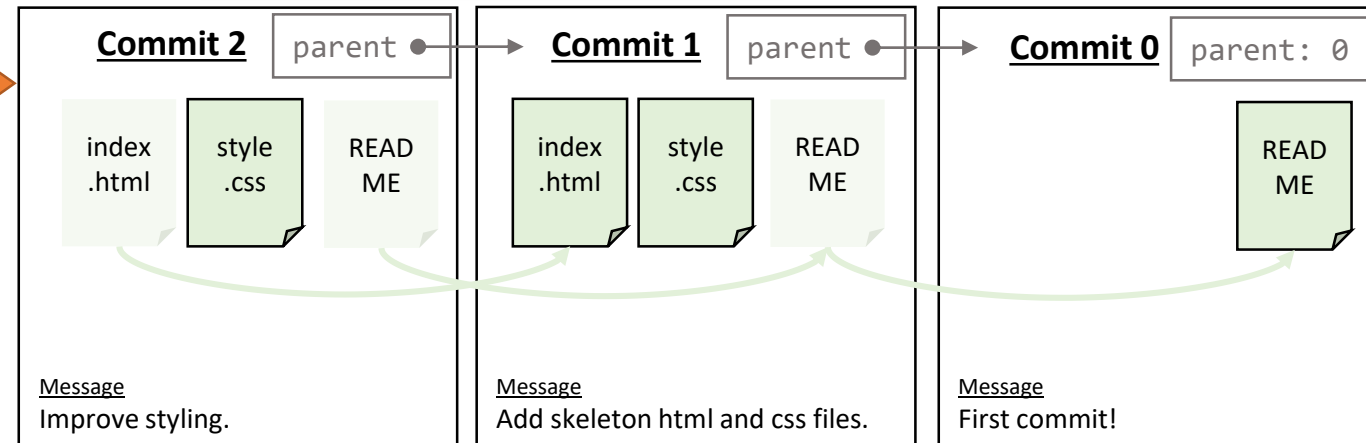


The .git Repository

Your default branch in git is called the **master** branch.

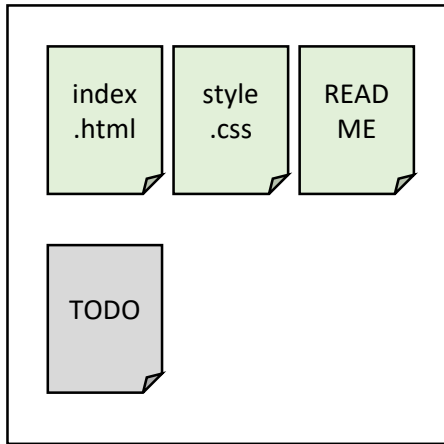
Really, it's just a reference to a commit. By default, it refers to the last commit made.

master →

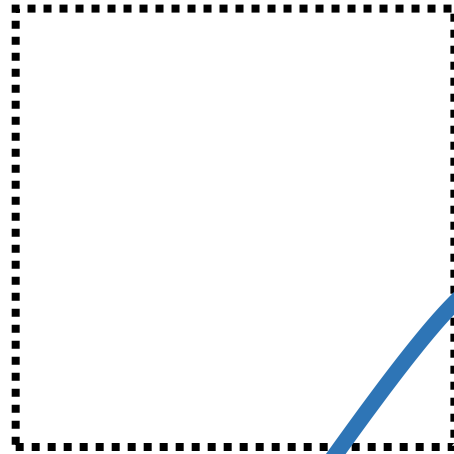


HEAD keeps track of the **branch** you are working on

Project Working Directory



Staging Area



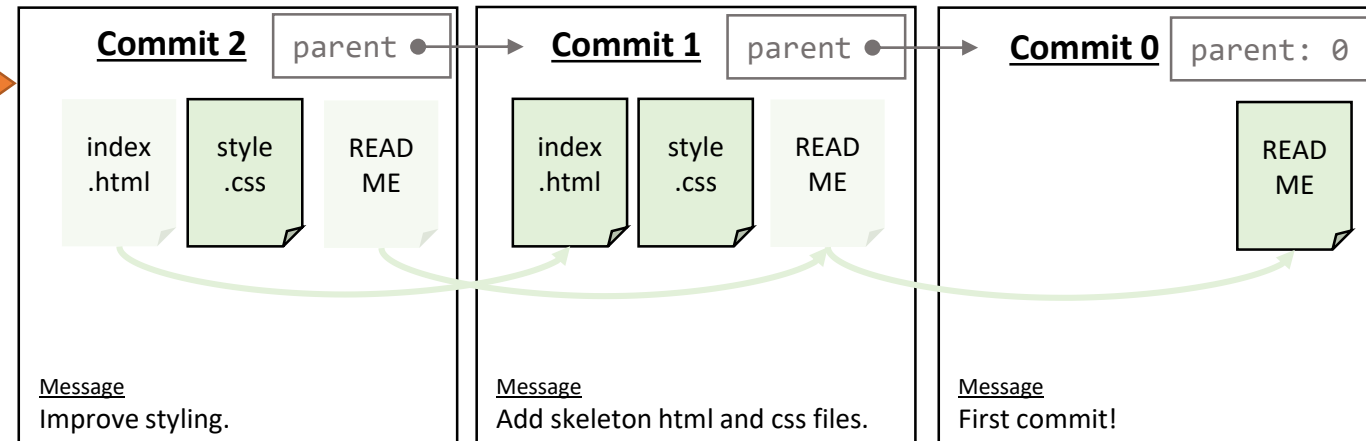
Your repository's **HEAD** is a reference to the branch you're working on.

When you make a commit, the parent of the commit is set to be the commit HEAD refers to.

HEAD and your current branch then update to refer to your new commit.
Just like a linked list.

HEAD

master



Want to try out an idea? Checkout a new branch!

Pr

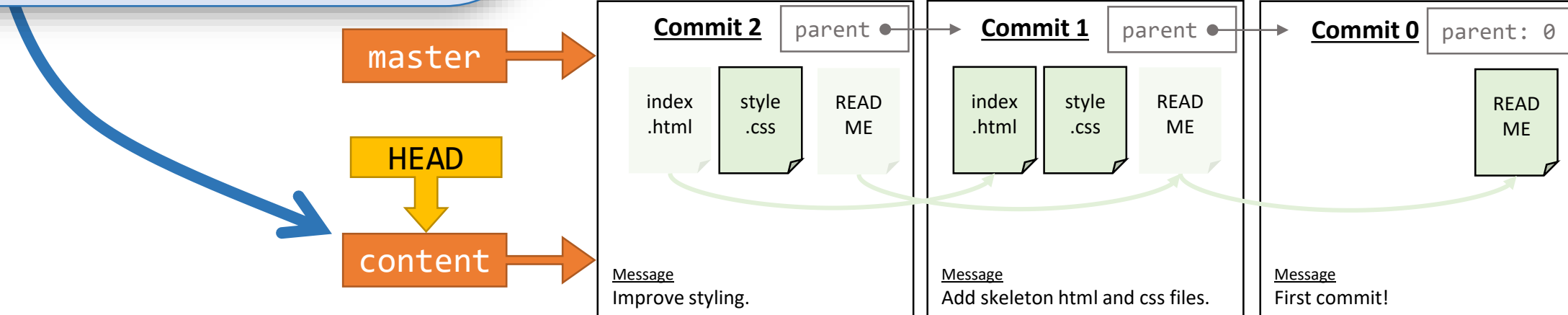
Working in a branch is recommended for trying out ideas. Like a tree branch, its commit history will grow in its own direction separate from the master branch.

Once your idea is fully formed, you can merge your branch into the master branch.

When you run the command to the right, a new branch is established and your HEAD is changed to it.

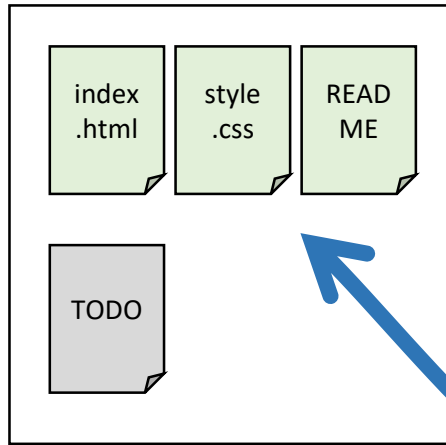
The .git Repository

```
git checkout -b content
```

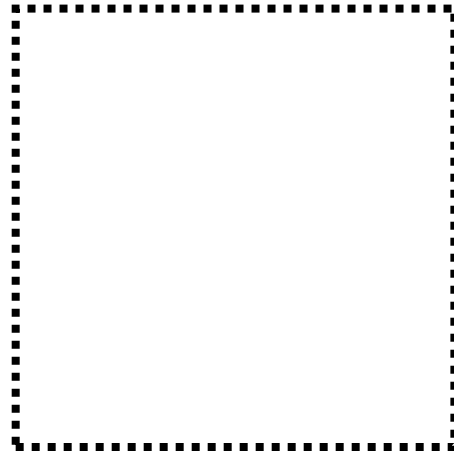


Want to try out an idea? Checkout a new branch!

Project Working Directory

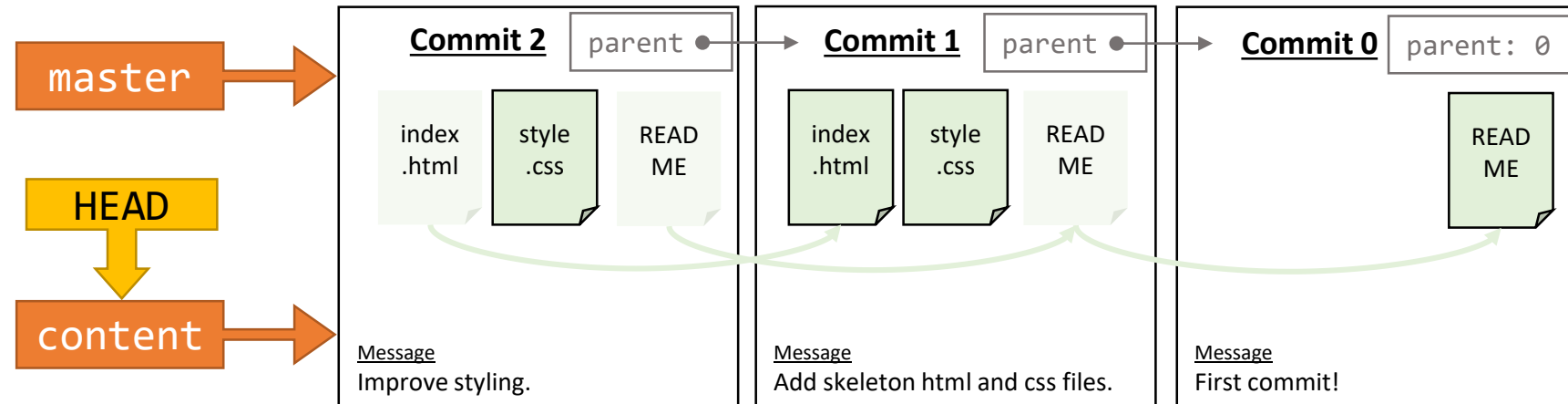


Staging Area



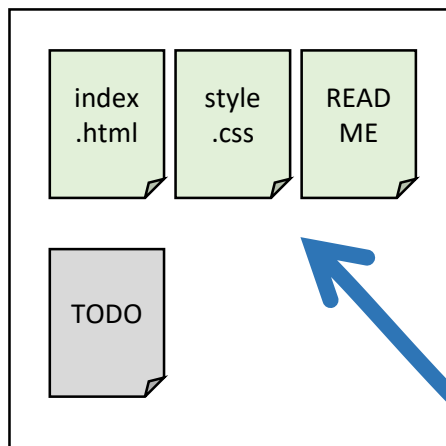
The .git Repository

Suppose you added some content to your index.html file and you're ready to commit it. Do you remember how?

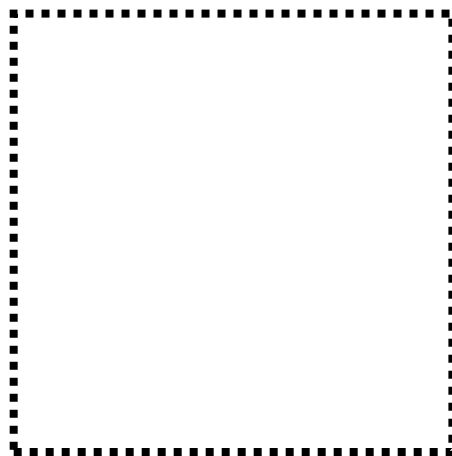


Making another commit...

Project Working Directory

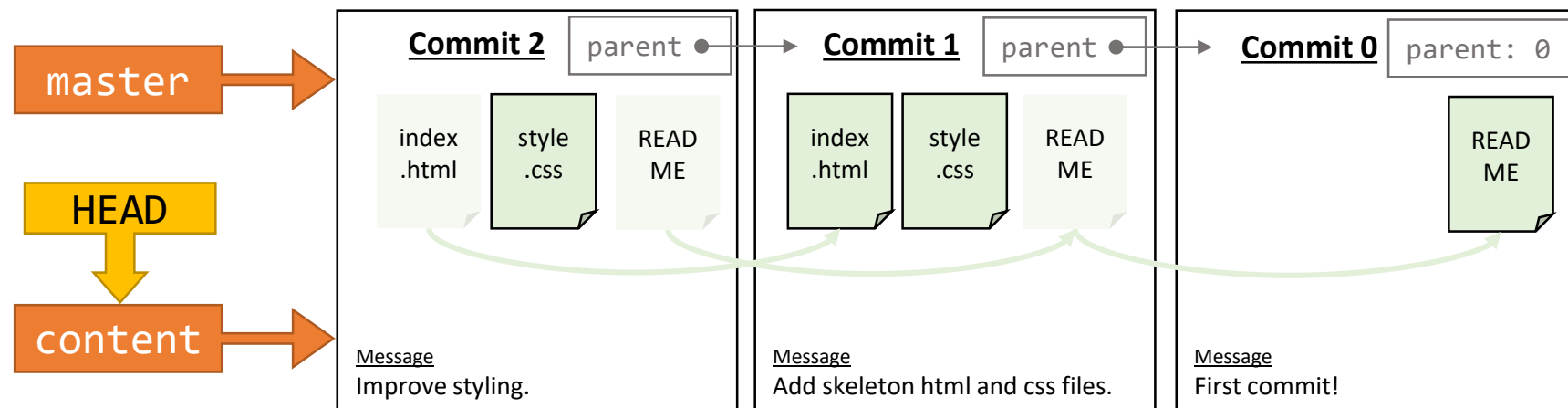


Staging Area



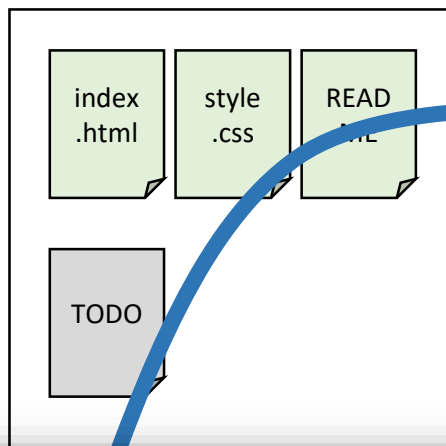
The .git Repository

Suppose you added some content to your index.html file and you're ready to commit it. Do you remember how?

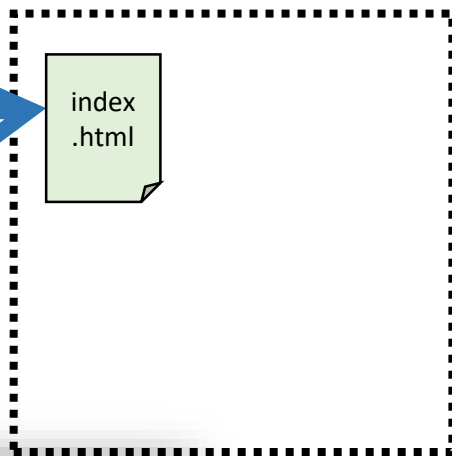


Making another commit...

Project Working Directory



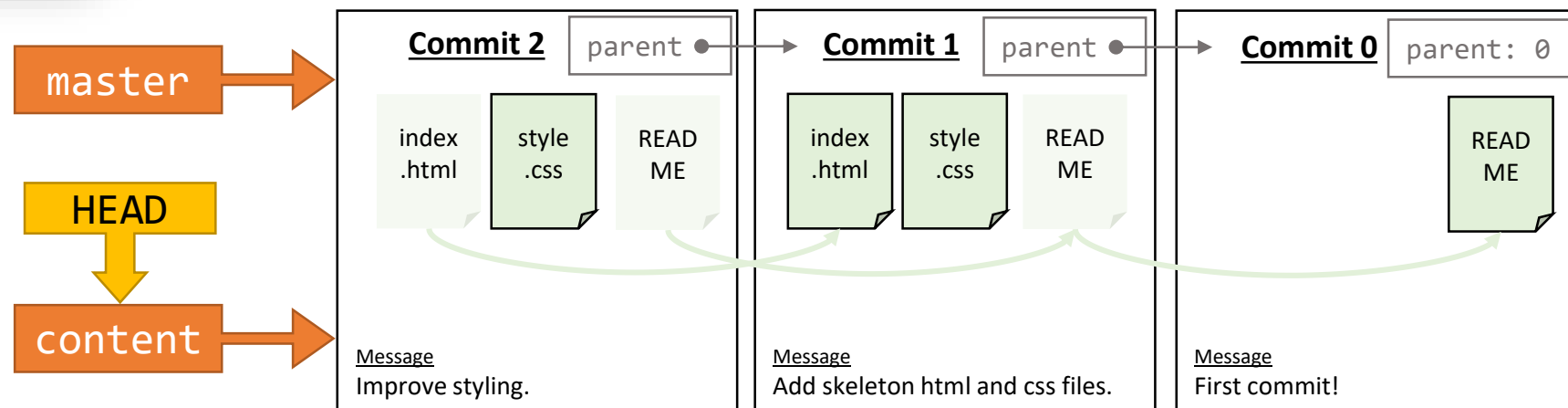
Staging Area



The .git Repository

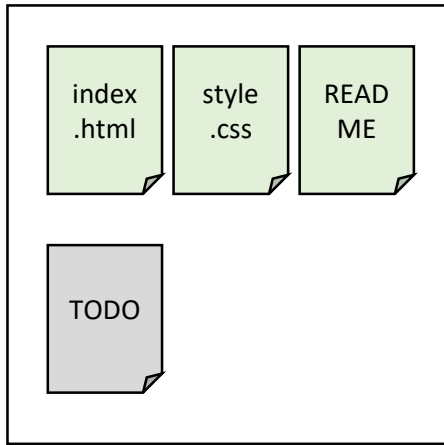
```
git add index.html
```

Step 1) **git add** to staging area.

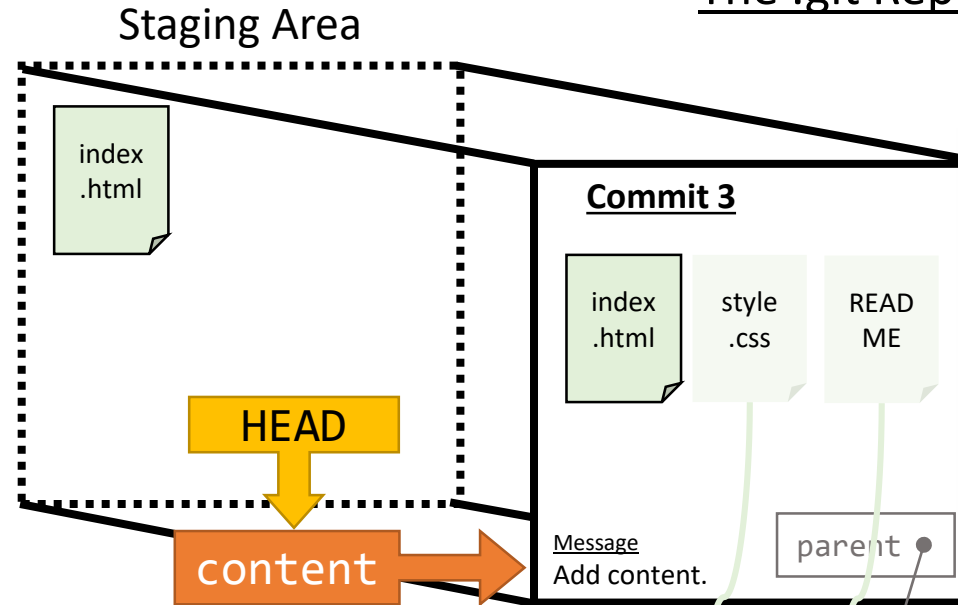


Making a commit on a branch

Project Working Directory



The .git Repository

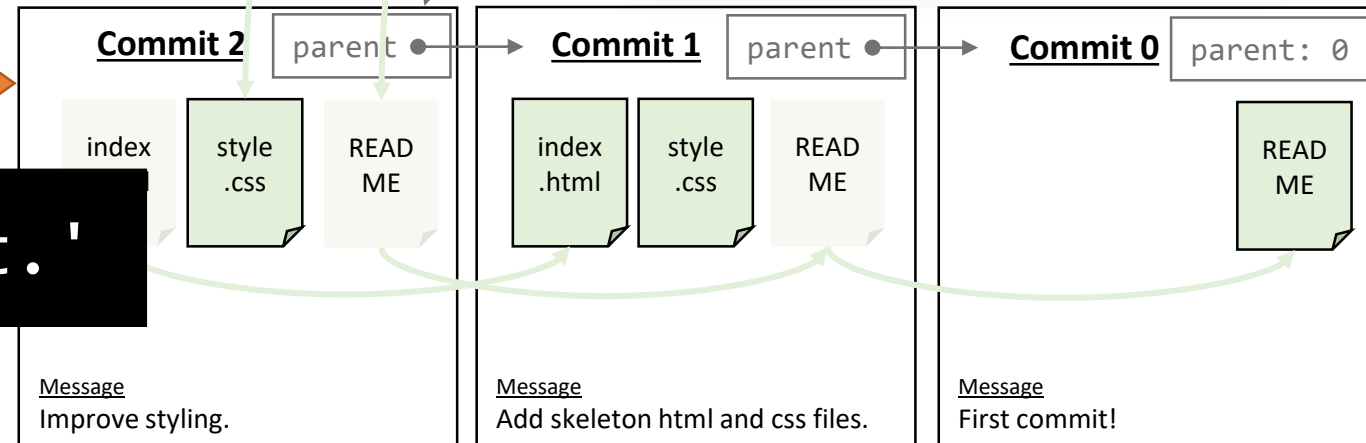


Whoa! Notice the content branch and **HEAD** were updated to refer to the new commit.

The **master** branch remained in place.

We're illustrating this **branch** as moving in a separate direction from master by convention.

master



```
git commit -m 'Add content.'
```


What's the big idea of branches?

You could imagine continuing to make progress and additional commits on this "feature branch".

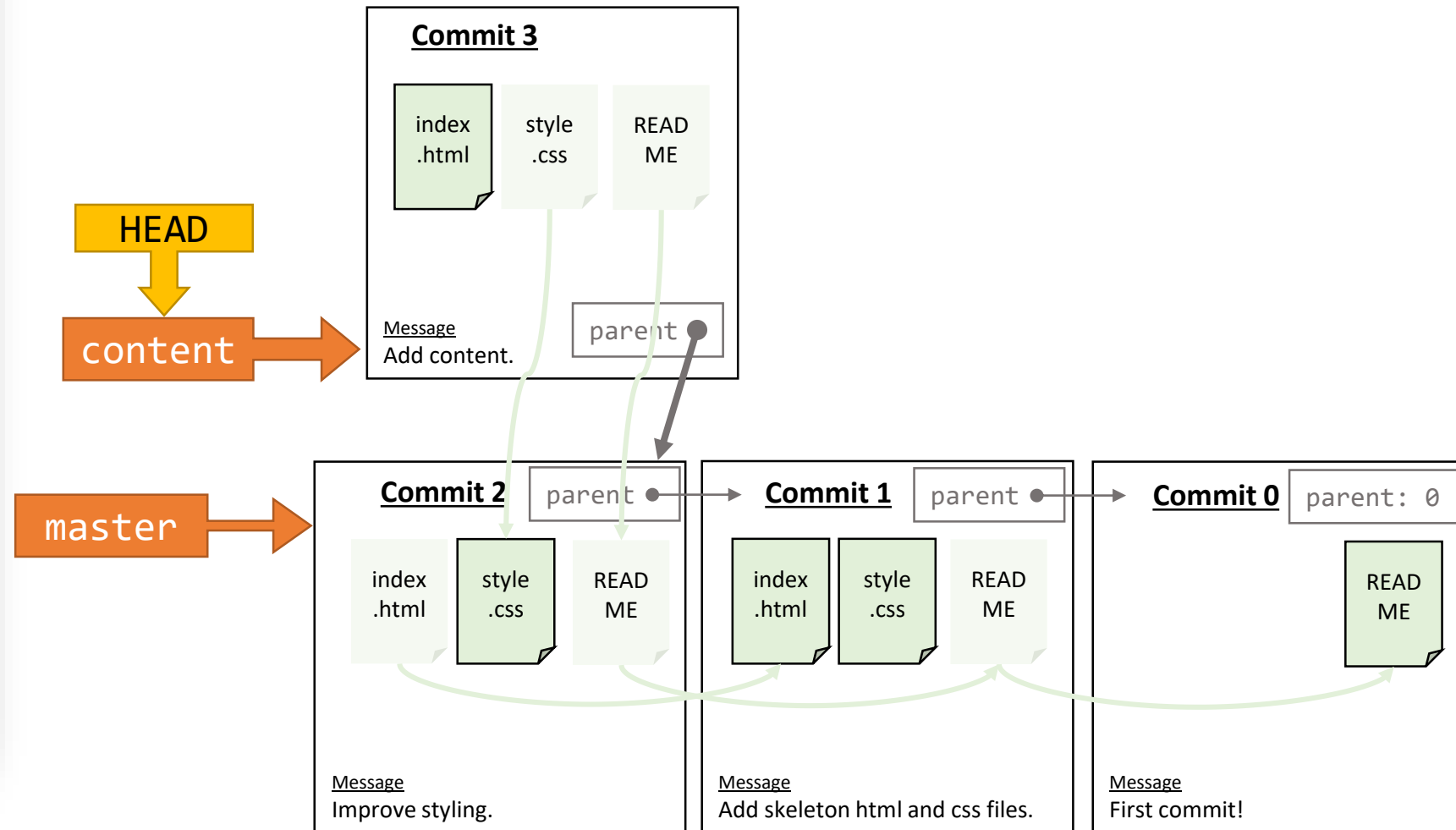
You can *easily* revert back to your master branch, though.

You could also easily start a *separate branch* to explore other feature ideas.

Good idea? Merge the feature branch back into the master branch.

Bad idea? Checkout the master branch and delete the feature branch.

The .git Repository



Pushing a branch to GitHub

```
git push origin content
```

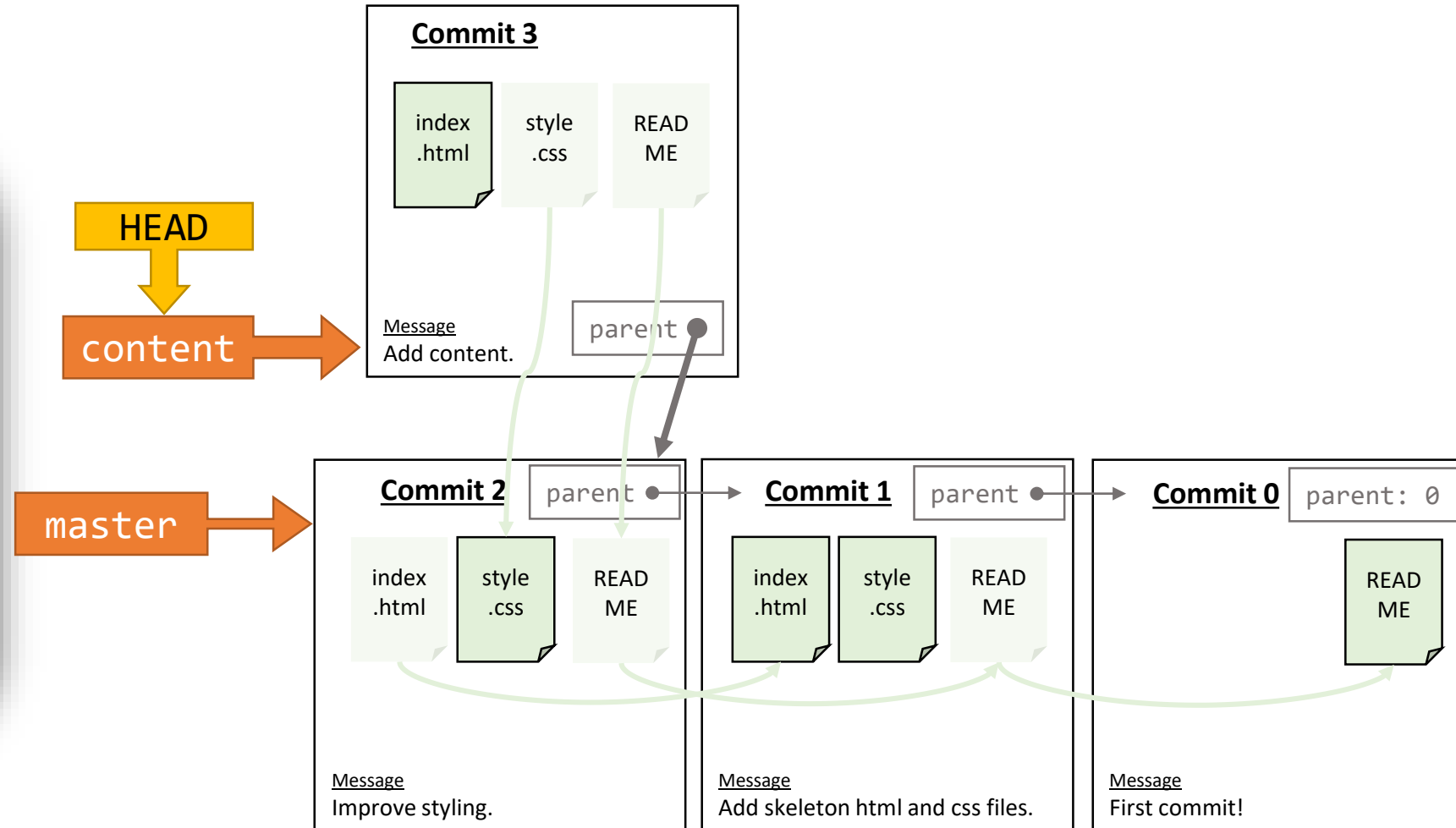
The .git Repository

Want to share your branch with a collaborator to check-out?

You can push a branch to a remote repository using:

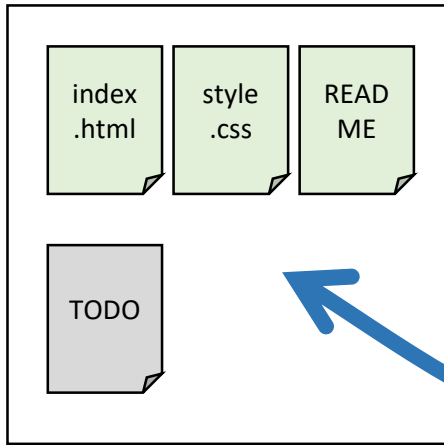
```
git push <remote> <branch>
```

Remote is usually "origin" and branch is the name of the branch you're pushing.



Checking out another branch

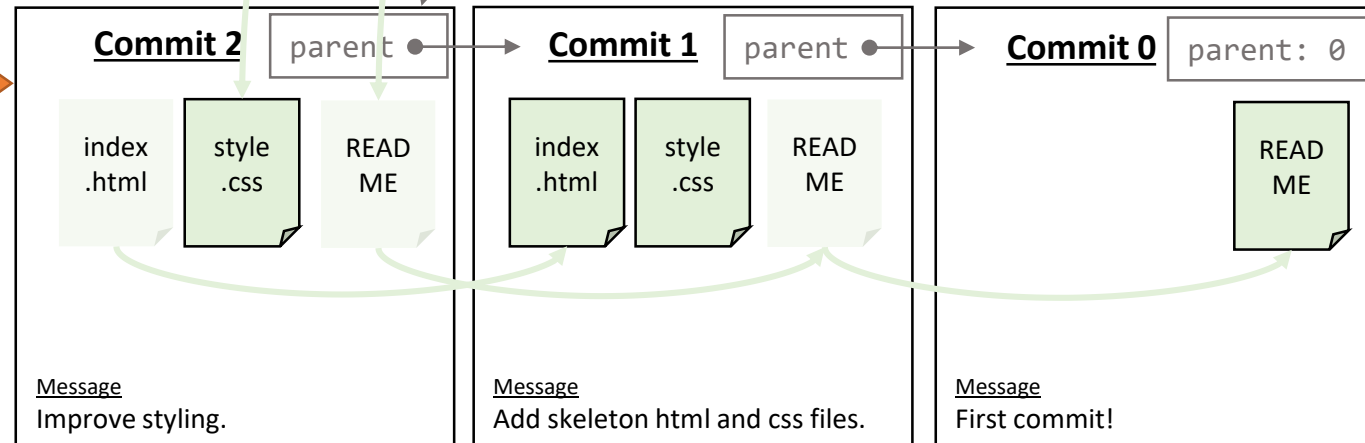
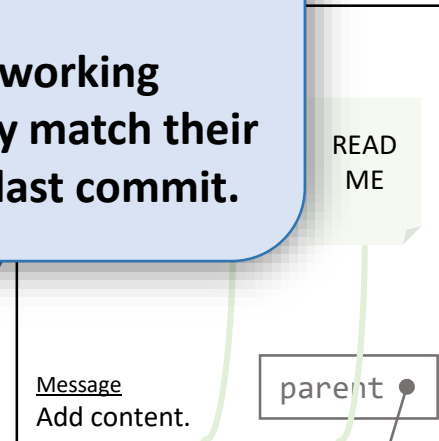
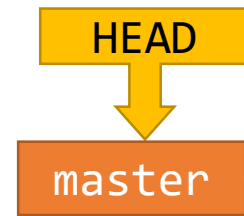
Project Working Directory



When you checkout a branch, your HEAD switches to the branch, AND

The files in your project's working directory are set to exactly match their contents at that branch's last commit.

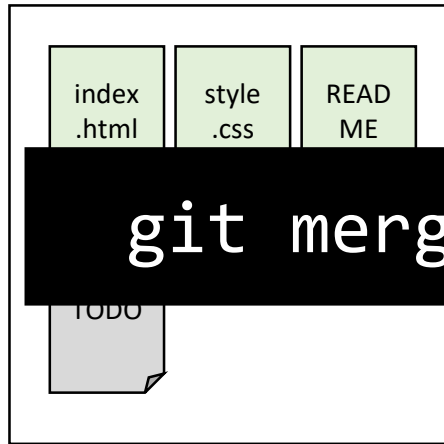
The Git Repository



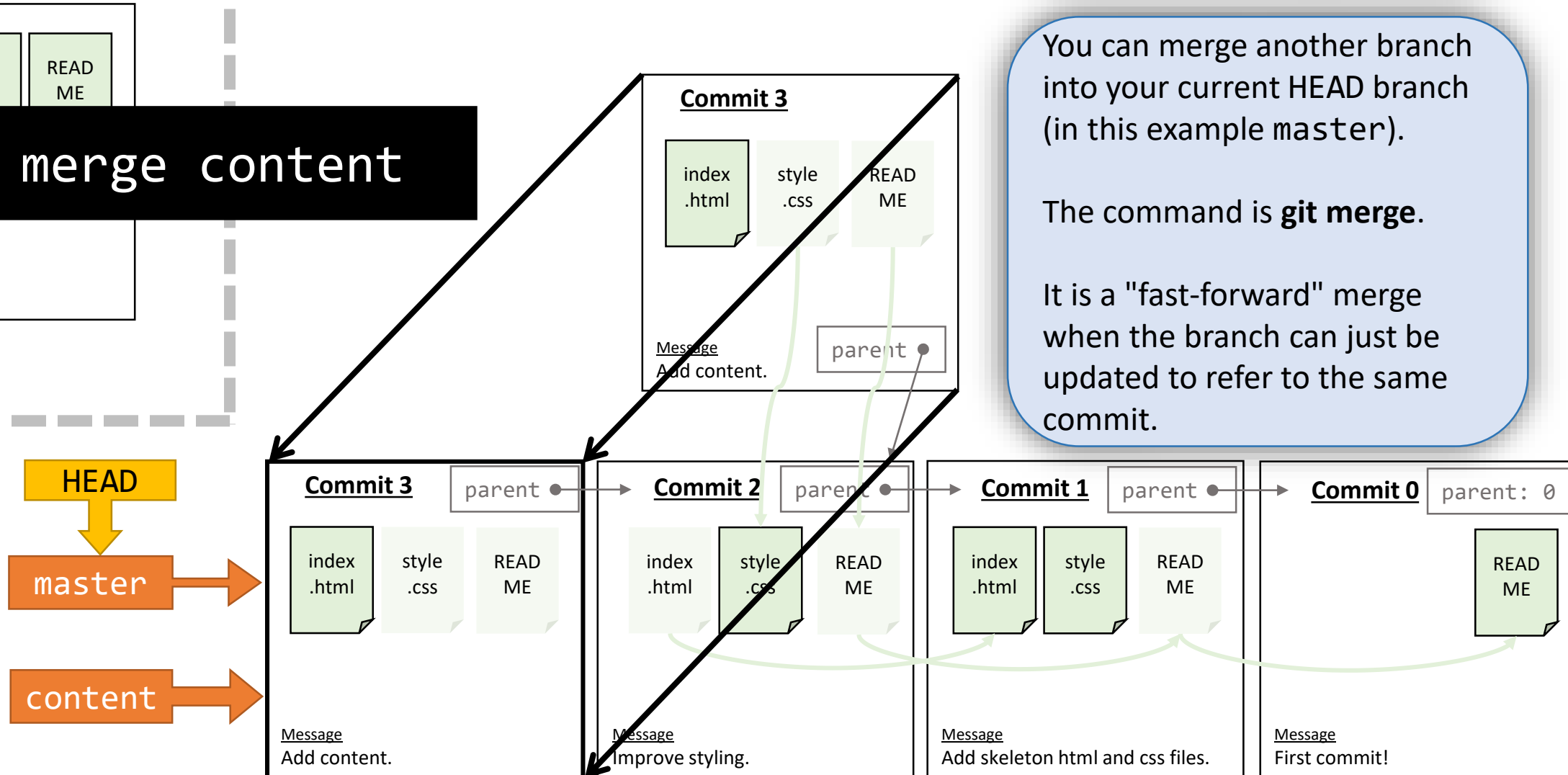
`git checkout master`

Merging a feature branch in to master (fast-forward)

Project Working Directory



The .git Repository



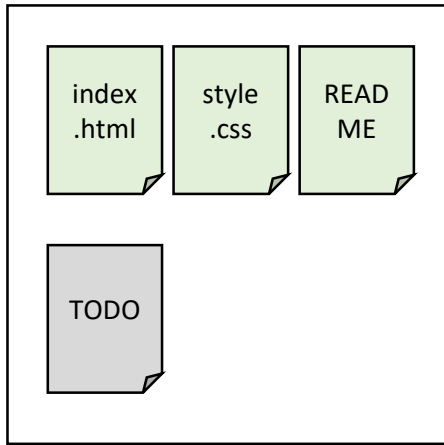
You can merge another branch into your current HEAD branch (in this example master).

The command is **git merge**.

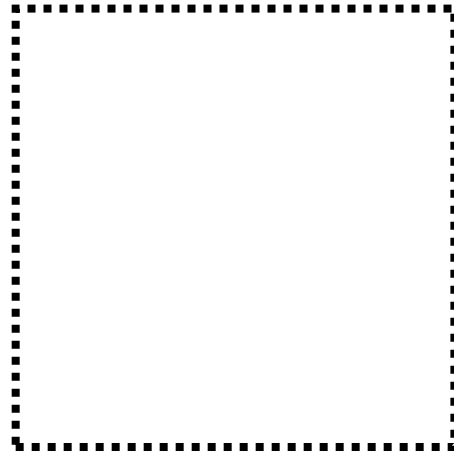
It is a "fast-forward" merge when the branch can just be updated to refer to the same commit.

Deleting a branch when you're done with it.

Project Working Directory

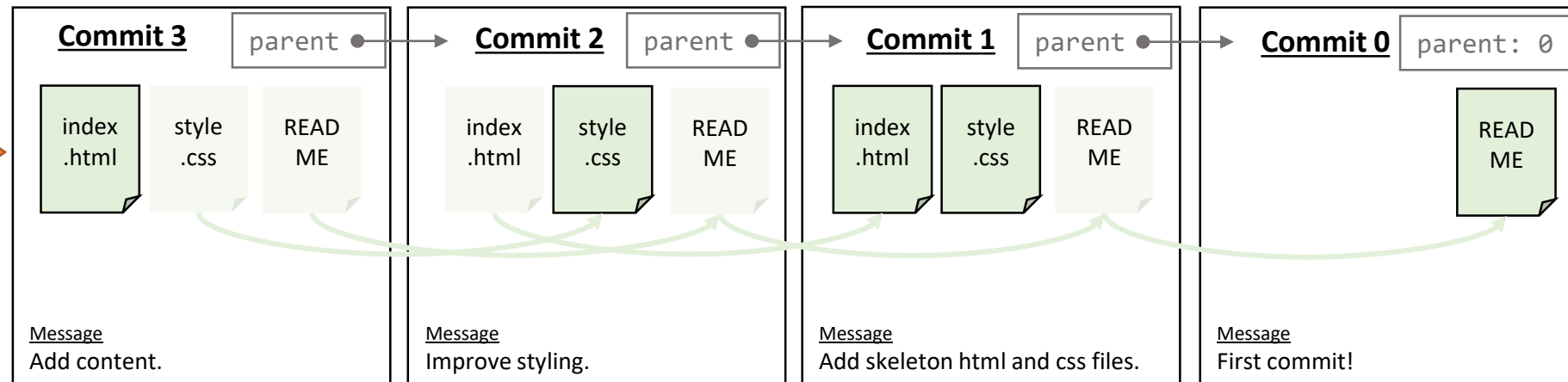
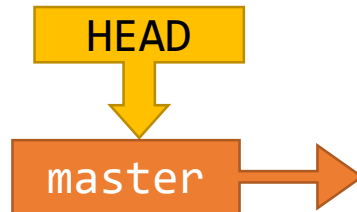


Staging Area



The .git Repository

`git branch -d content`



Pushing your master branch to GitHub

```
git pull origin master  
git push origin master
```

When pushing to a branch others are collaborating on, be sure to pull before pushing.

When you pull, git is trying to merge changes made to the branch by collaborators first.

Then, when you push your master branch, your commits are uploaded to the remote repository.

The .git Repository

