Fundamentals of Git

Missouri Satellite Team

Presented by: Illya Starikov

Getting Started

There'll be some setting up before you can start using git. It'll depend on your operating system — you can refer to the installation guide here. Below are the more popular methods of installation.

macOS `brew install git`[1] or installing command lines tools.

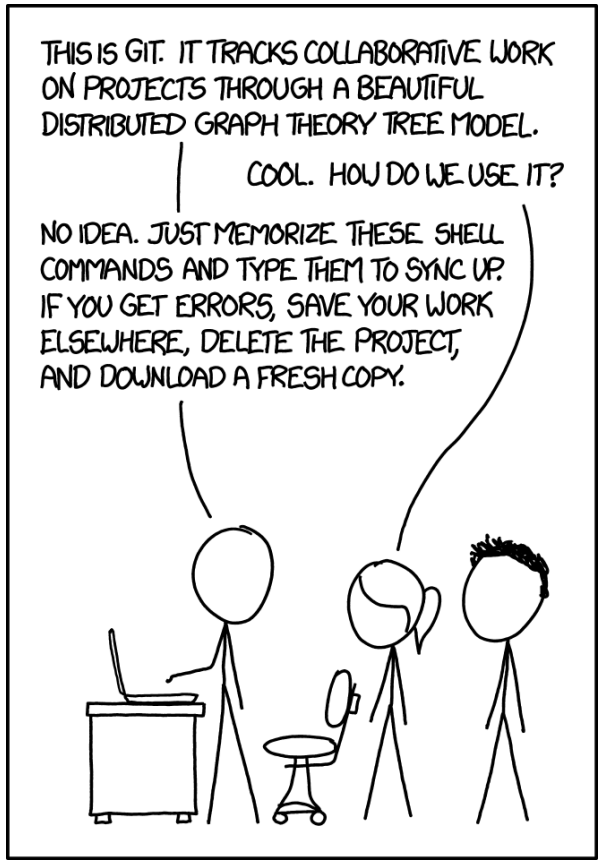
Linux Depends on your distro. If Ubuntu use `sudo apt-get install git-all`, if Arch Linux then `pacman -S git`, if others refer here.

Windows Download a `.exe` from here.

[1] Or whatever hip package manager you use.

What is Git?

- Git is nothing more than Directed Acyclic Graph of objects compressed and identified by an SHA-1 hash.
- Git works in snapshots, not differences.
- Git is local.
- Git has data integrity.
- Git is parallelizable.



The Five Stages of Git

1. Working Directory
2. Staging Area
3. Git Directory
4. ...
5. Profit

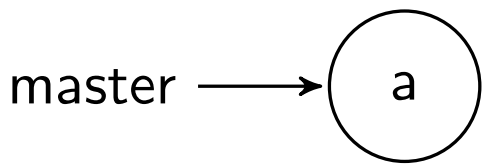
`Git`ting Good

If the "stages" didn't make sense, that's alright. It's better to go through a workflow apposed to the formalities.

1. Start a new repository with `git init`
2. Work on project in bite sized chunks, and add files that were changed with `git add file(s)`
3. Commit your changes with `git commit`
4. Optionally, `git push` to save changes to the remote branch
5. Of course, profit.

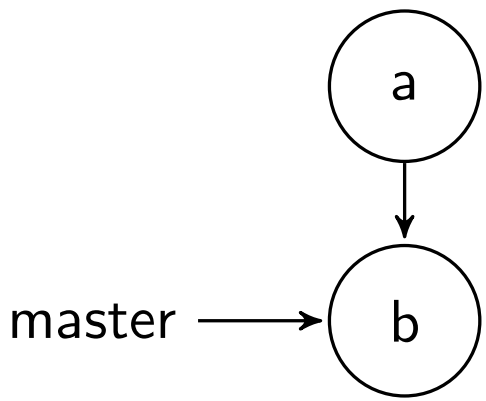
Committing I

```
git commit -m ‘‘a’’
```



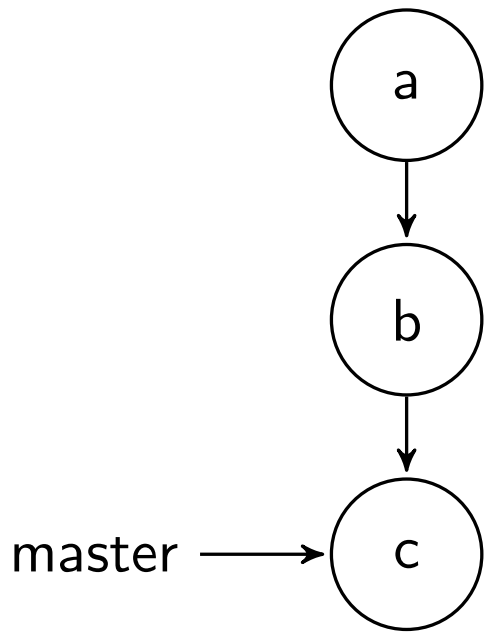
Committing II

```
git commit -m ‘‘b’’
```



Committing III

```
git commit -m ‘‘c’’
```



The Commit Message

This is the easiest part of git — and also the easiest to mess up. Here are seven rules of a great commit message.

1. Separate subject from body with a blank line.
2. Limit the subject line to 50 characters.
3. Capitalize the subject line.
4. Do not end the subject line with a period.
5. Use the imperative mood in the subject line.
6. Wrap the body at 72 characters.
7. Use the body to explain *what* and *why* vs. *how*.

The Commit Message (Example)

```
Summarize changes in around 50 characters or less

More detailed explanatory text, if necessary. Wrap it to about 72
characters or so. In some contexts, the first line is treated as the
subject of the commit and the rest of the text as the body. The
blank line separating the summary from the body is critical (unless
you omit the body entirely); various tools like `log', `shortlog'
and `rebase' can get confused if you run the two together.

Explain the problem that this commit is solving. Focus on why you
are making this change as opposed to how (the code explains that).
Are there side effects or other unintuitive consequences of this
change? Here's the place to explain them.

Further paragraphs come after blank lines.

 - Bullet points are okay, too

 - Typically a hyphen or asterisk is used for the bullet, preceded
   by a single space, with blank lines in between, but conventions
   vary here
```

Demo

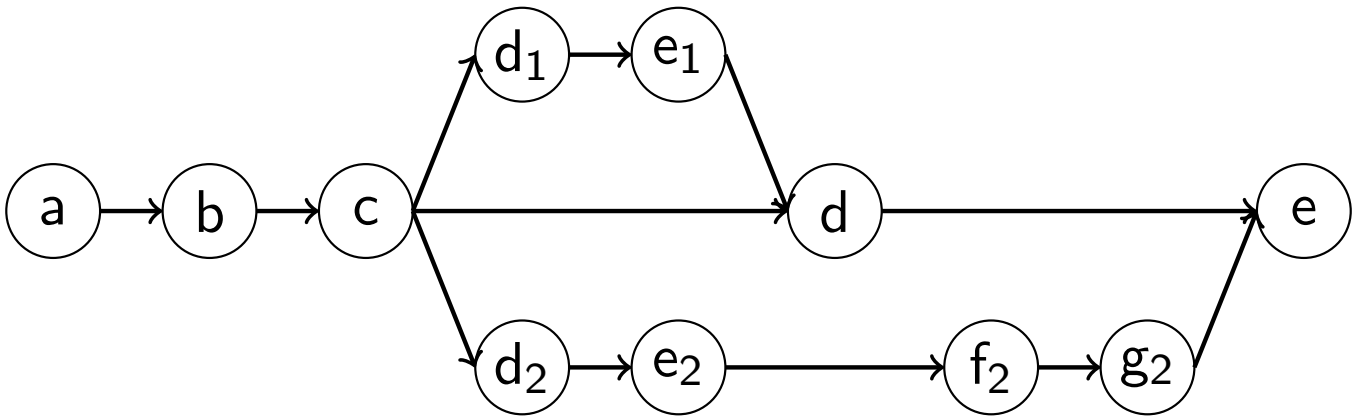
`Git`ting Better

Some more advanced commands to make your job easier.

- The wildcard `*` expands to whatever can fit a certain pattern.
  - `git add *.cpp` stages all files with a `cpp` extension.
  - `git add damon.*` add all document types with the name of `damon`, whether it be `cpp`, `txt`, or (unfortunately) `jpg`.
- `git add -A` stages new, modified, and removed files.
- `git commit -m ‘‘<msg>’’` commits with the commit message `<msg>`. **Only use if you absolutely know what you're doing.**

Branching I

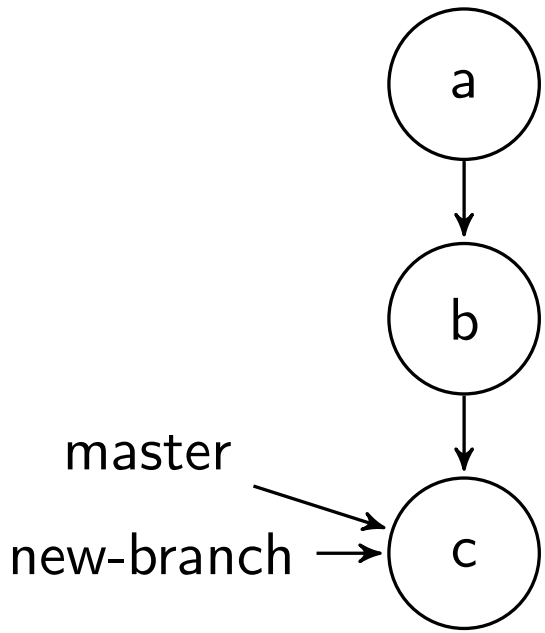
Branching is what allows for multiple people to work on multiple parts of the project.



So $D_1 \cdots E_1$ could be a bug fixing branch while $D_2 \cdots G_2$ could be a feature branch.

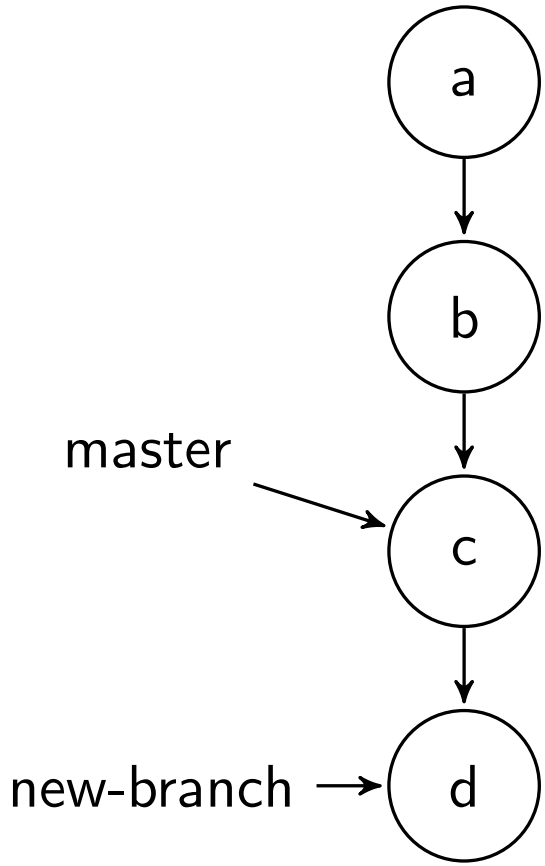
Branching II

```
git checkout -b new-branch
```

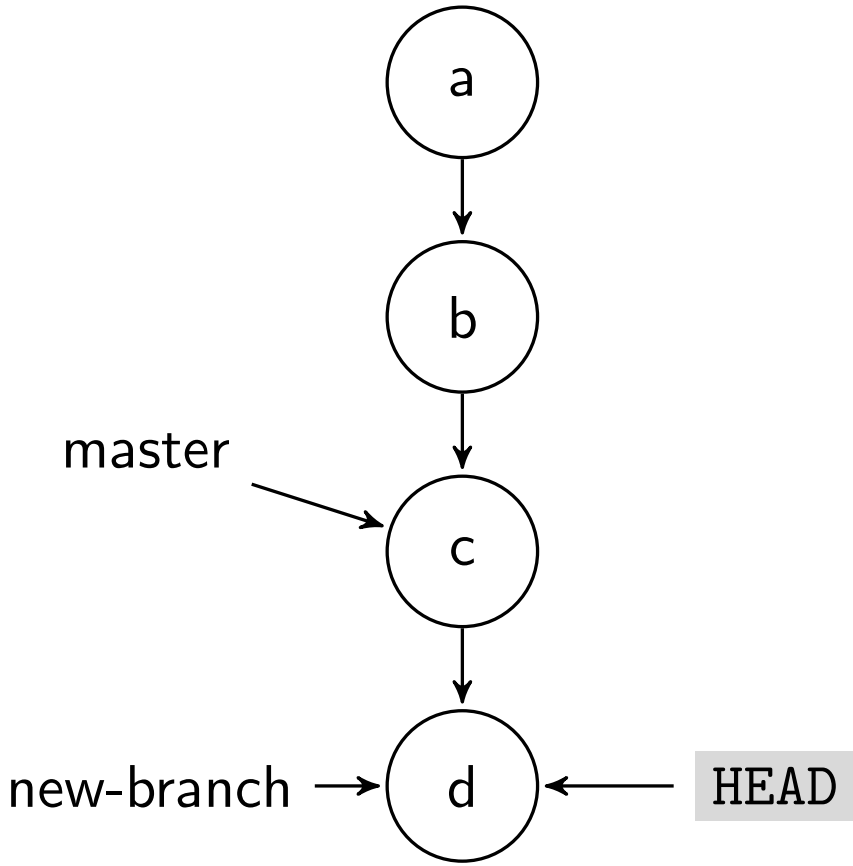


Branching III

```
git commit -m ‘‘d’’
```

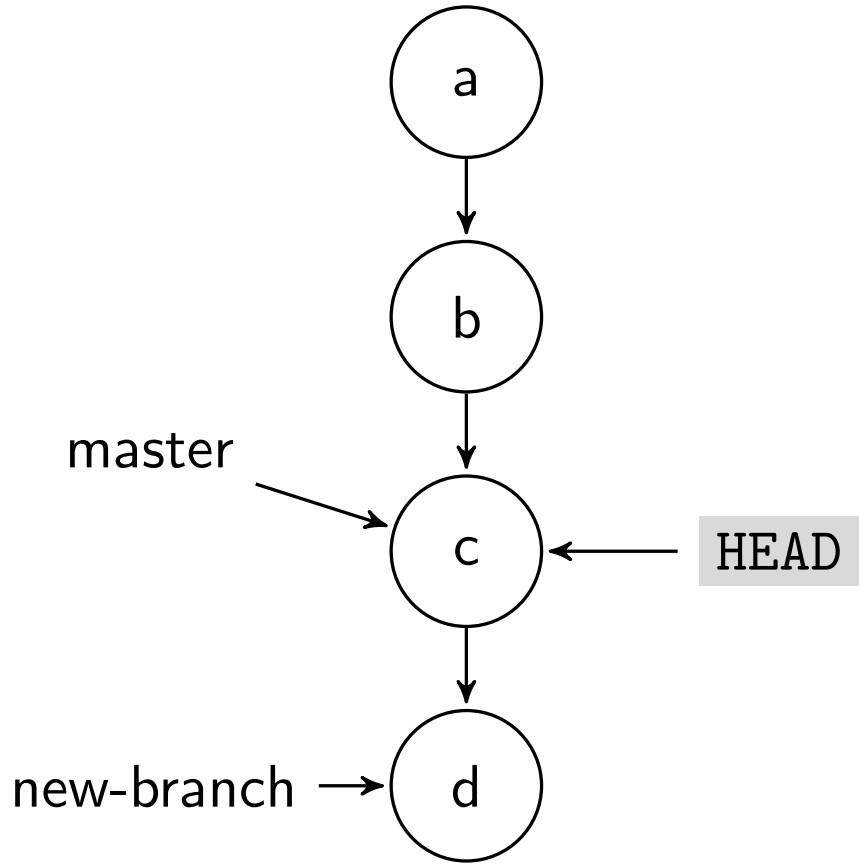


Branching IV



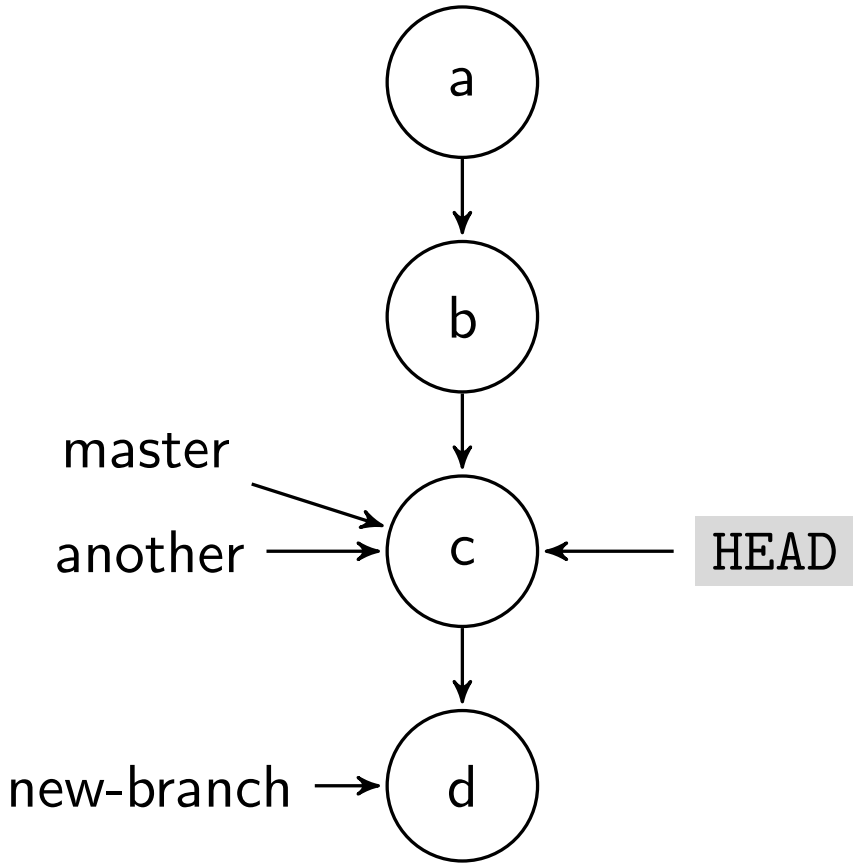
Branching V

```
git checkout master
```



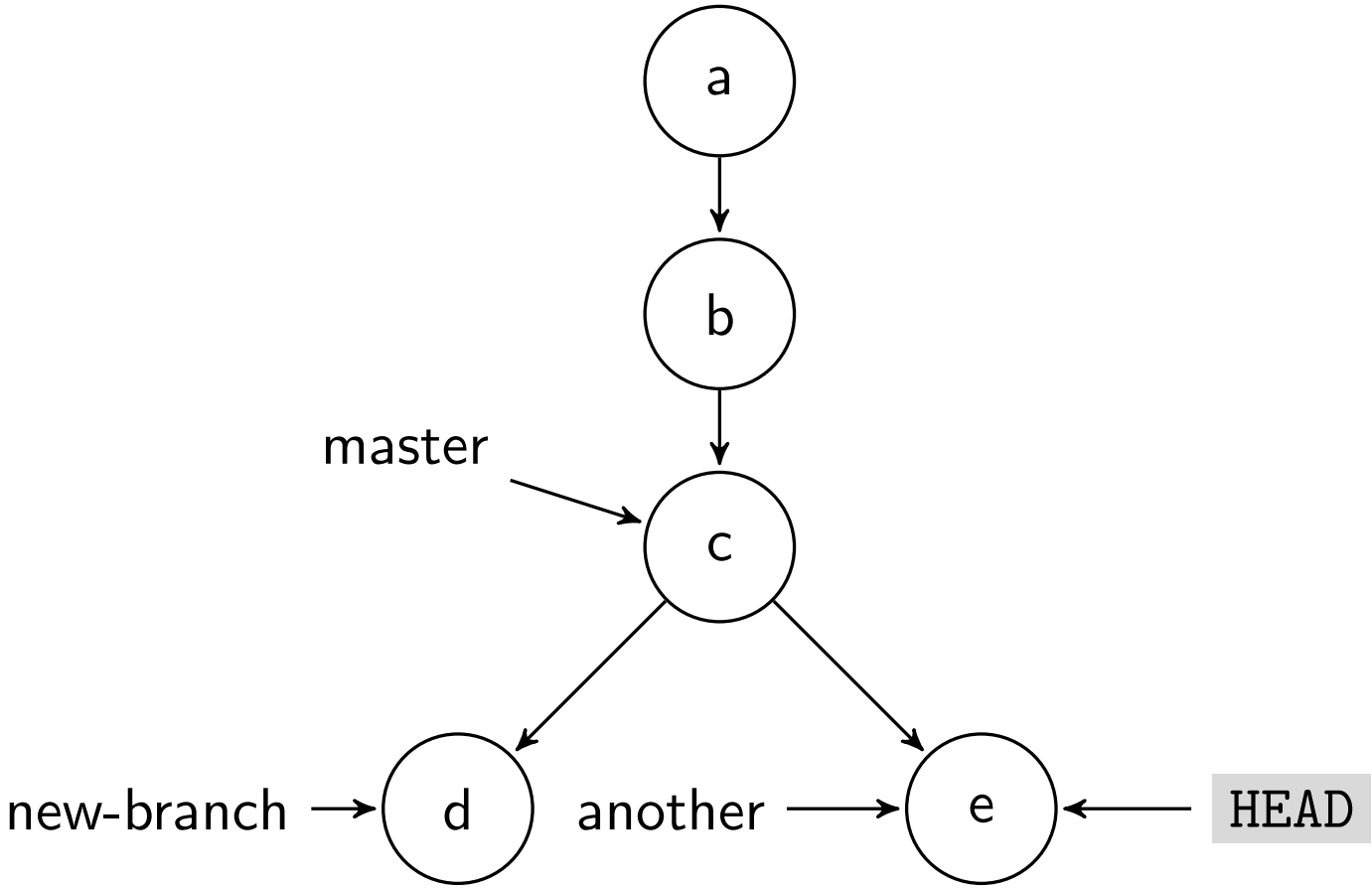
Branching VI

```
git checkout -b another
```



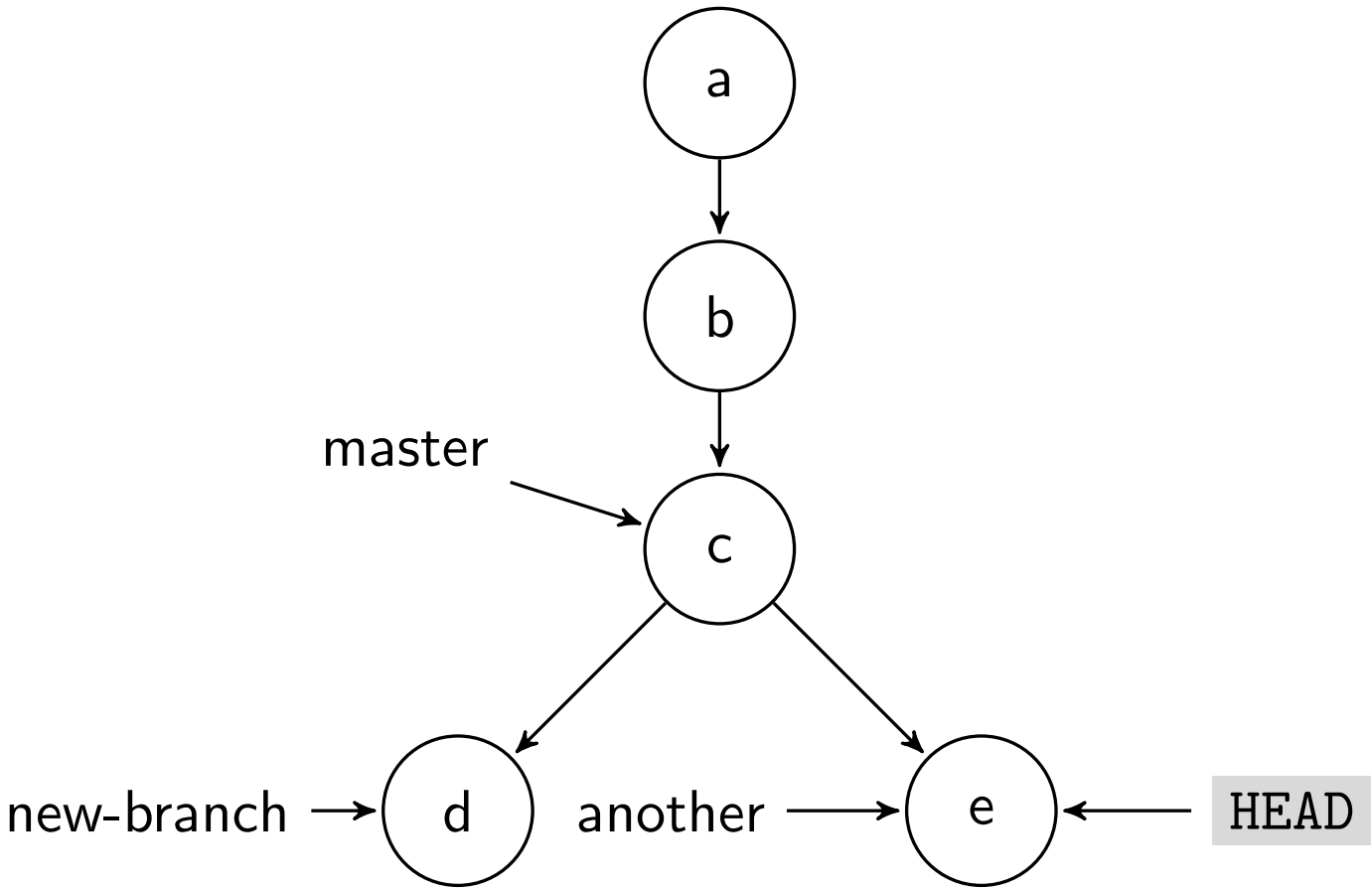
Branching VII

```
git commit -m ‘‘e’’
```



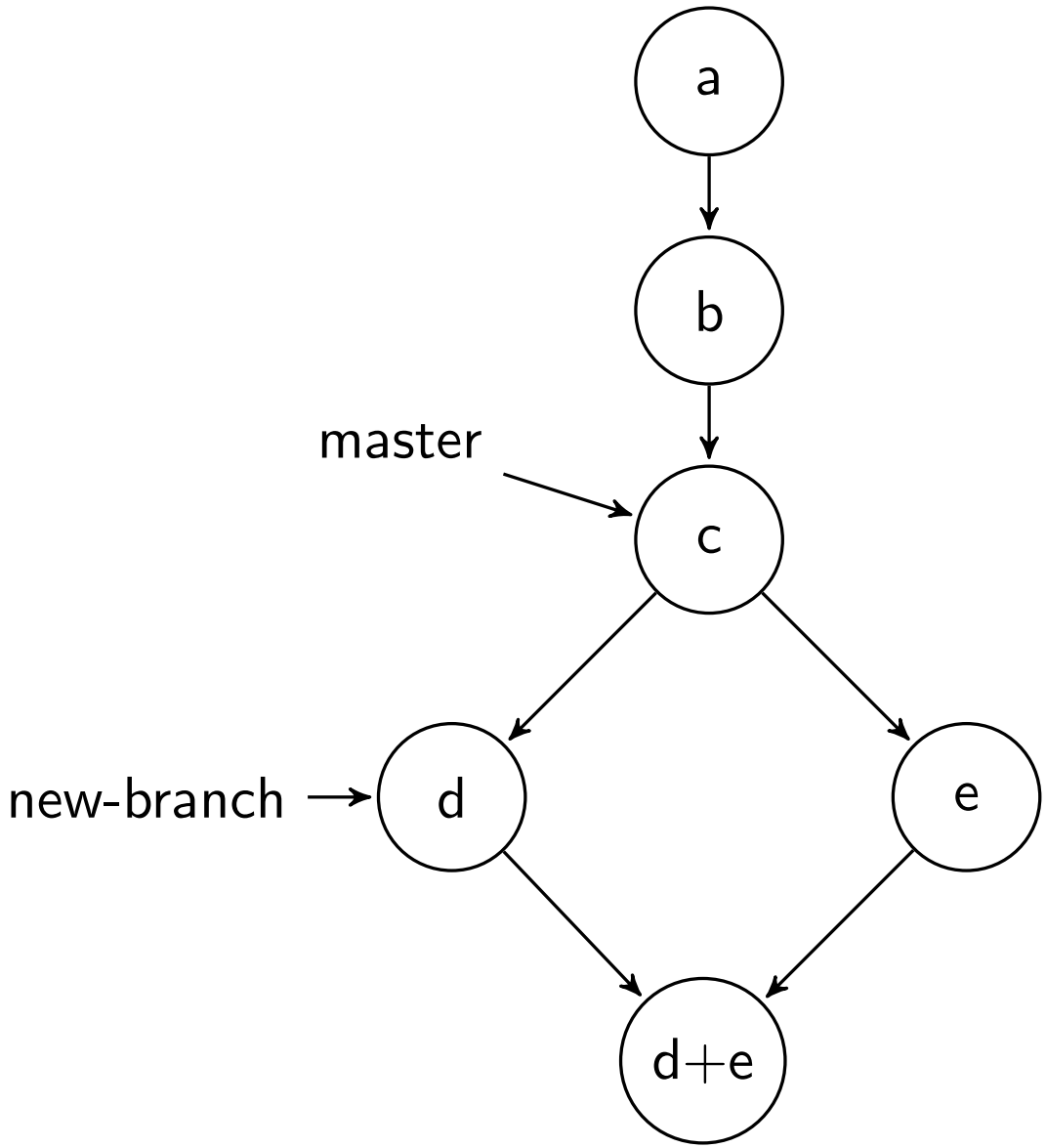
Merging I

```
git merge new-branch
```



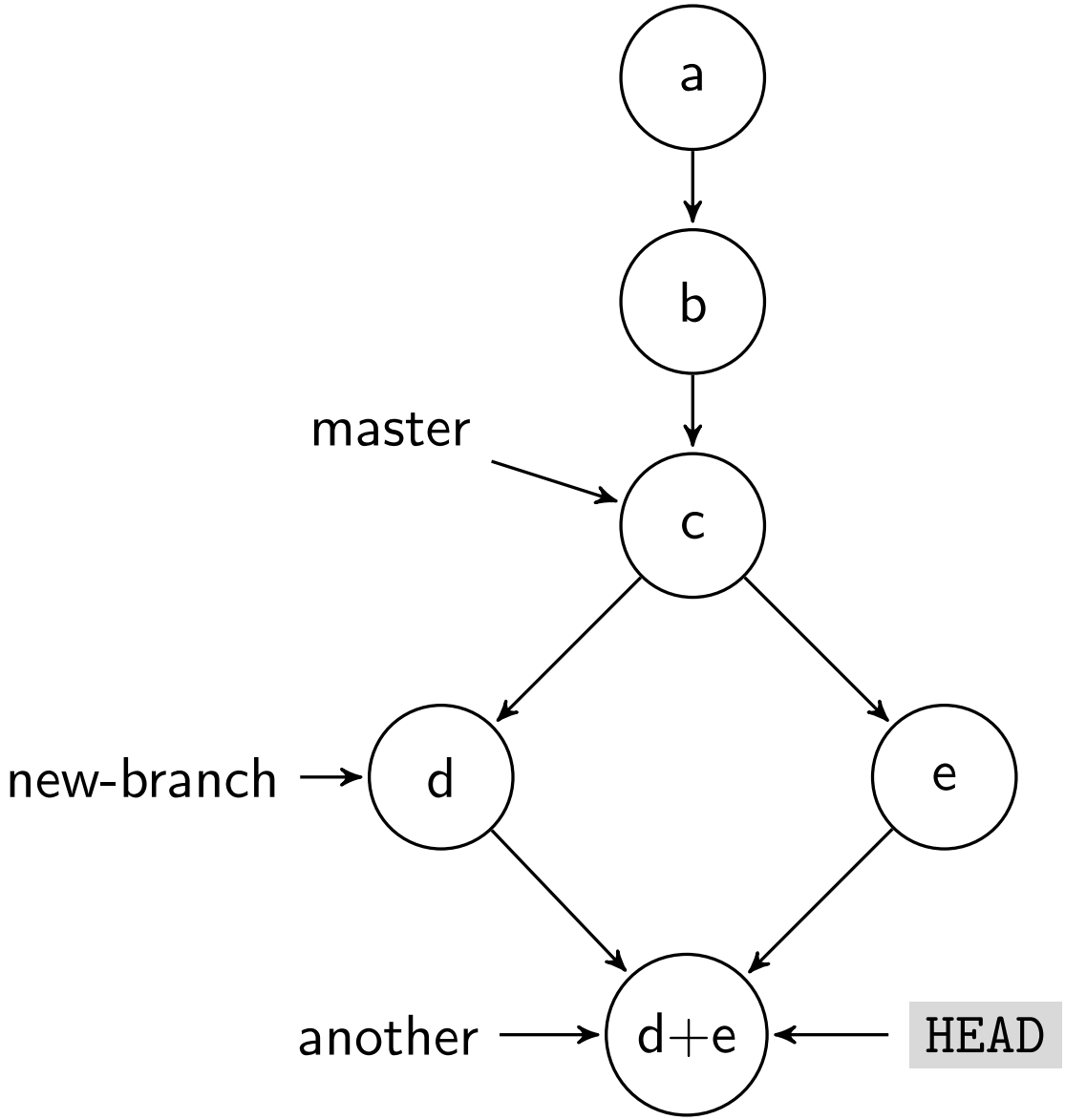
Merging II

```
git merge new-branch
```



Merging III

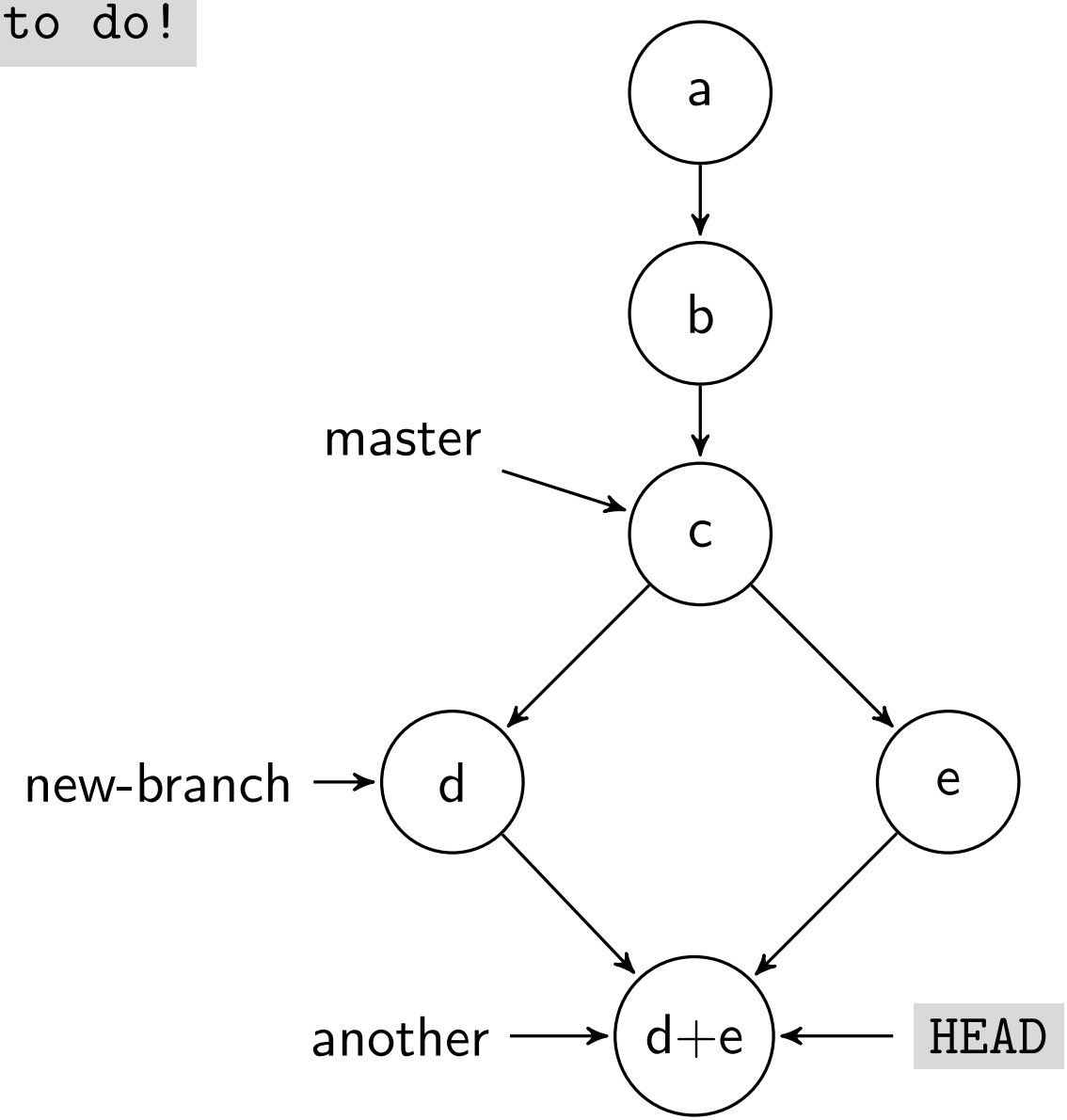
```
git merge new-branch
```



Merging IV

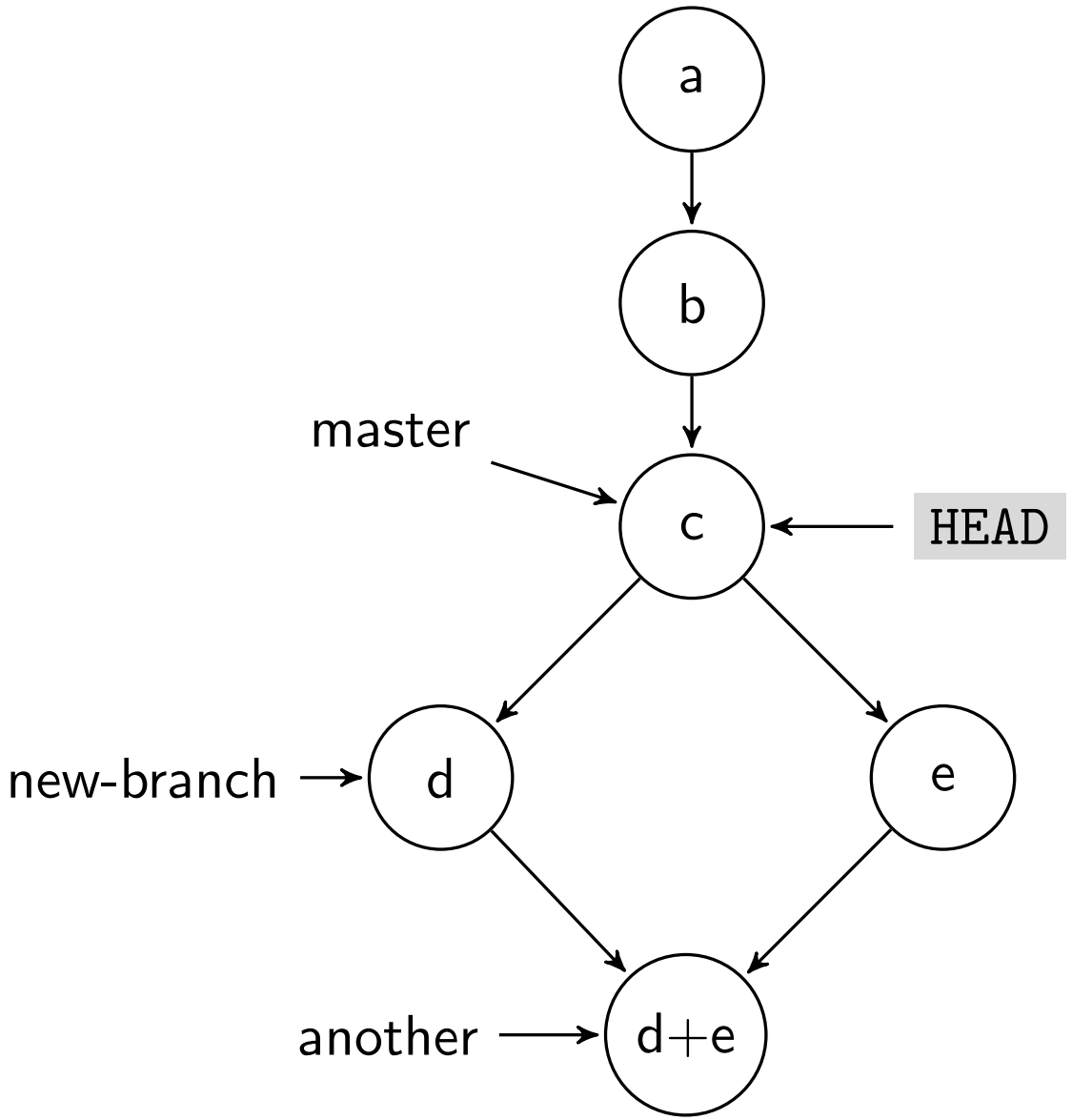
```
git merge master
```

```
Nothing to do!
```



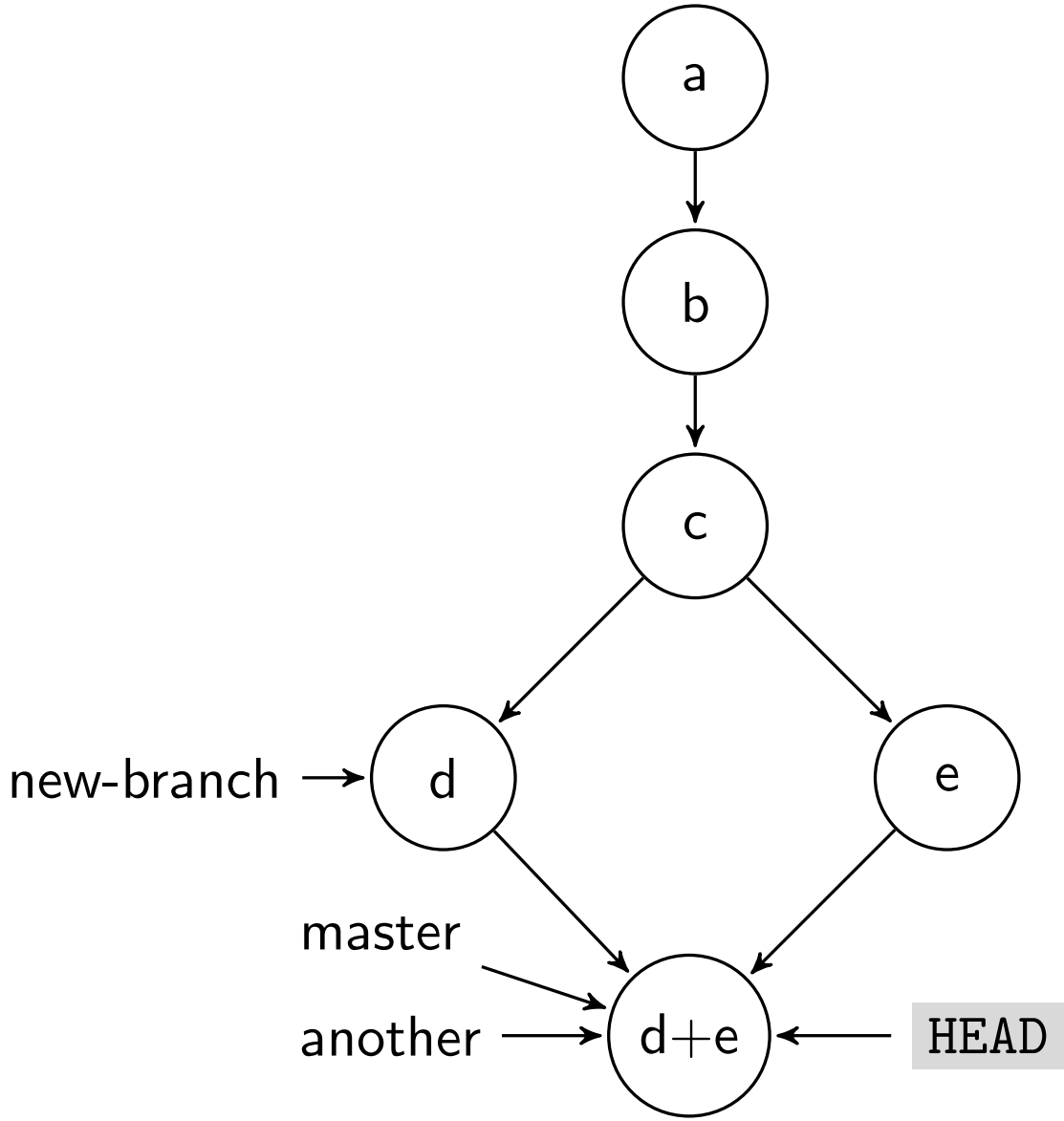
Merging V

```
git checkout master
```



Merging VI

```
git merge another
```



Merge Conflicts I

On occasion, you might run into a merge conflict. These arise when you modify two parts of a shared code-base. For instance, an error message could appear like so:

```
Auto-merging the-files(s).txt
CONFLICT (content): Merge conflict in the-file(s).txt
Automatic merge failed; fix conflicts and then commit the result.
```

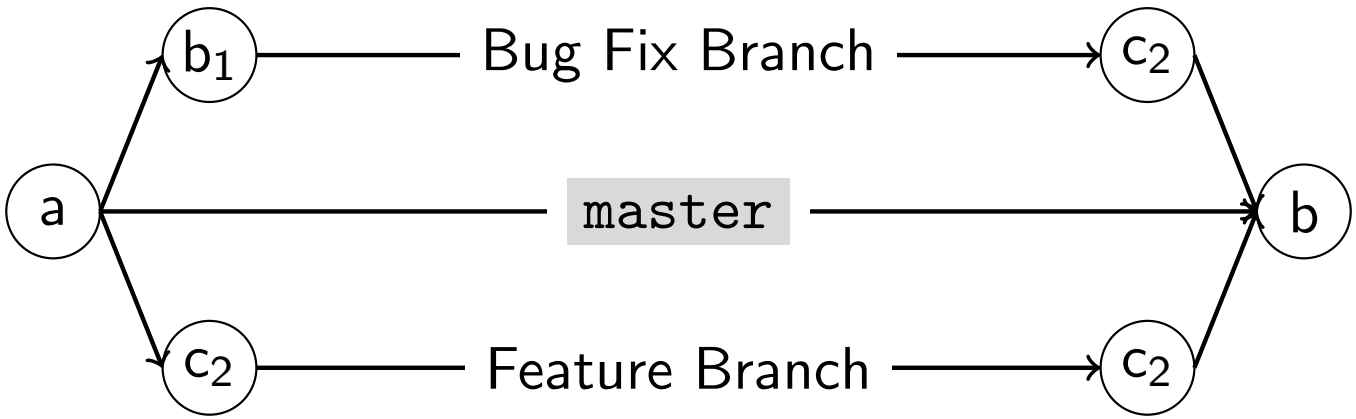
Upon inspection, you should find something along the lines of:

```
<<<<<<< HEAD
The current branch's contents
=======
The branch you're merging's contents
>>>>>>> other-branch
```

Merge Conflicts II

There are many great ways to deal with merge conflicts!

1. Merge early, merge often.



Demo

You Broke It — Finding The Mistake

- `git log` shows all previous commits. It has many useful parameters.
  - `--stat` Shows statistics (deletions, insertions, files changed) for your files.
  - `--pretty=oneline` A one line list of all your changes.
  - `--graph` Shows a graph.
  - `-p` Shows what are the changes to your code base.
- `git diff` show the difference between the working directory and the last commit.

You Broke It — Fixing The Mistake

- `git checkout COMMIT` Just as branches, looks at a previous commit.
- `git revert COMMIT` Generate a new commit that undoes all of the changes introduced in `COMMIT`, then apply it to the current branch.
- `git reset COMMIT` Move `head` to `commit` and reset the staging area to match. *Leaves the working directory alone*.
- `git reset --hard COMMIT` Move `head` to `commit` and reset the staging area to match. *Destroys working direcory*. **Only use if you absolutely know what you're doing.**

Working With Remotes

- `git clone LOCATION` Makes a copy of a repository in the current directory — you can set up SSH keys for Git/Gitlab or just use the url.
- `git push ORIGIN BRANCH` Pushes changes from your current branch to the remote branch it tracks.
- `git pull ORIGIN BRANCH` Pulls changes from the remote branch and merges them into your current branch.

In Closing

Special thanks to Nathan Jarus for "lending" me his LaTeX tikz code for the branching, committing and merging section.