2. Commits

Git training Duco

Nero Vanbiervliet

September 9, 2018

Understanding commits

Commits are snapshots

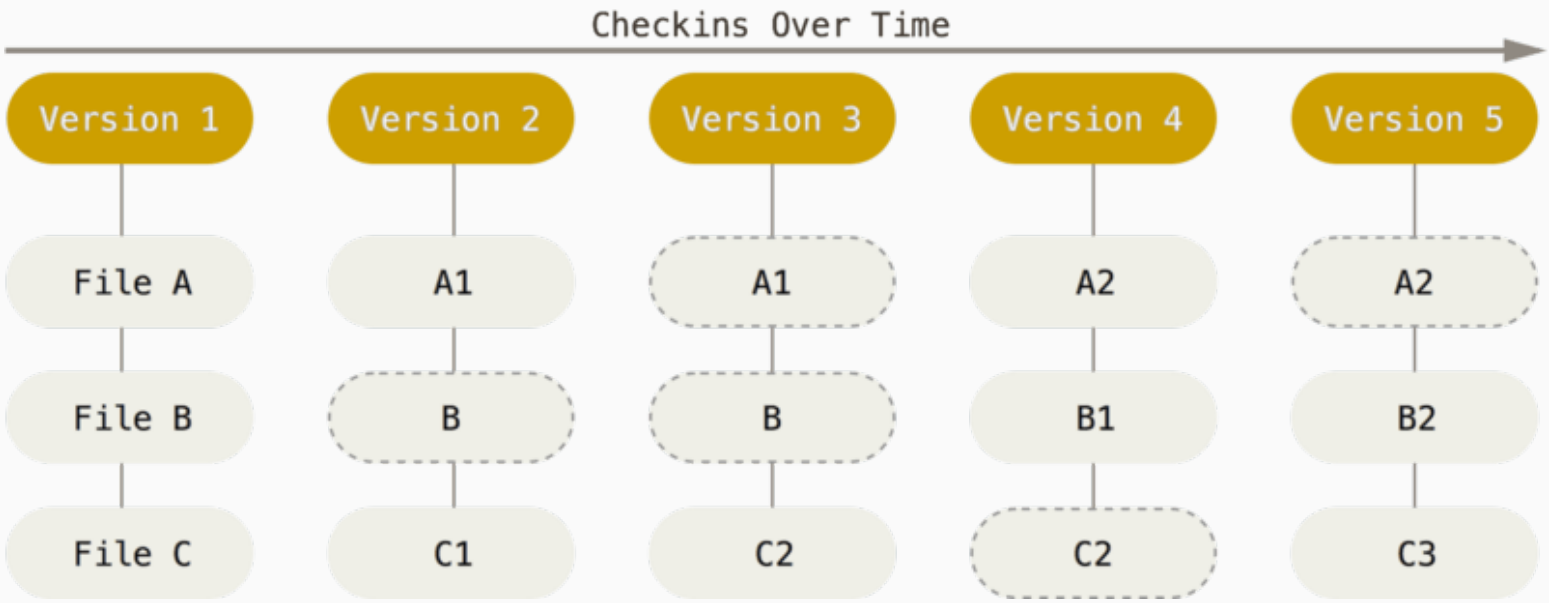

Figure 1: Over time, files change. *Source: Git Pro book*

With Git, we can take snapshots at different times of our software

We call this *commits*

Commit hash

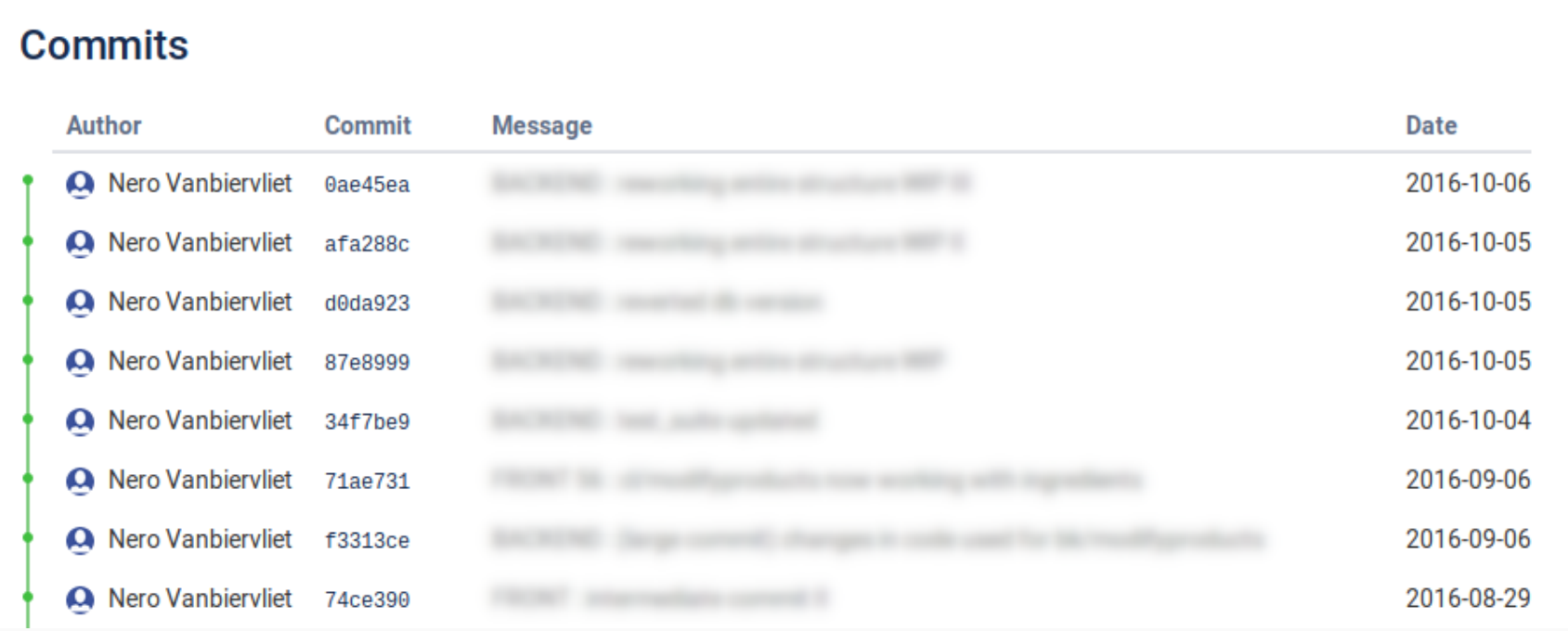

Figure 2: The commit hash is a unique identifier.

E.g. `cc9da8579b24ddf13f9e306ce437064bf58072ea`

E.g. `cc9da85` (short)

Commit message

Commits have a message, indicating what was changed since the last commit

Tips for making good commit messages

- Keep it short: maximum 50 characters
- Use imperative, e.g. ”Add documentation for duco box eco”

Commit Guidelines

Relation between commits

Figure 3: Each commit has an ancestor. *Source: Git Pro book*

Branching

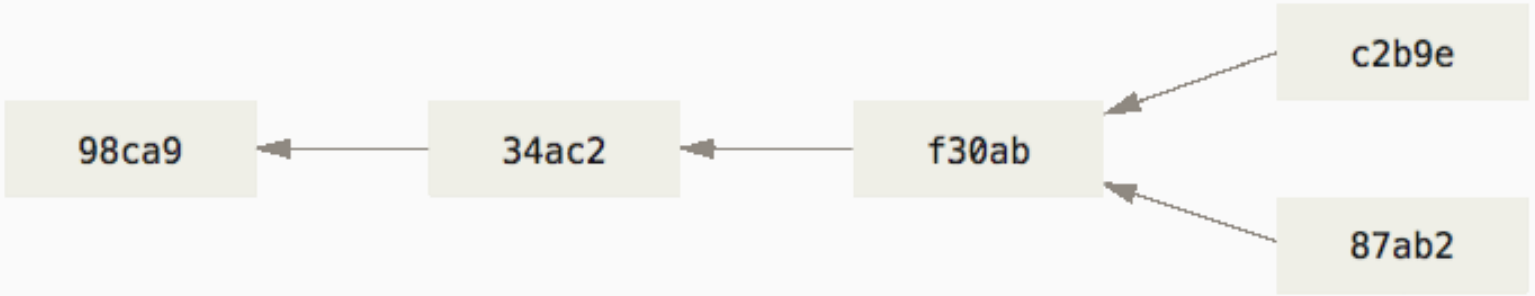

Figure 4: Commits can have the same ancestor. *Source: Git Pro book*

Branching will be discussed in detail as the next topic

Branches in a Nutshell

Navigating between commits

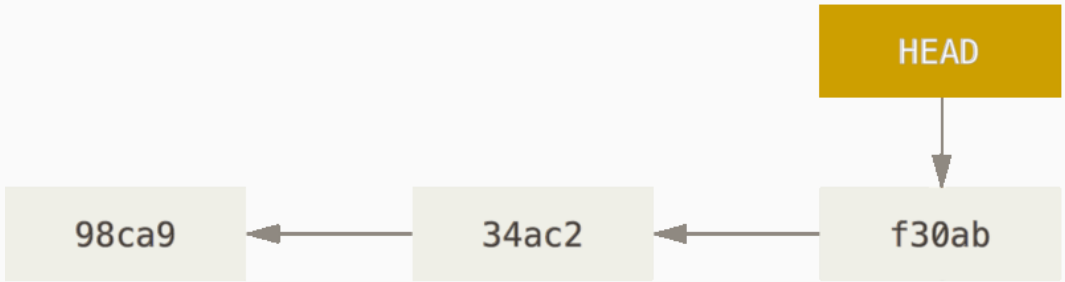

Figure 5: HEAD points to a commit. *Source: Git Pro book*

HEAD normally points to the last commit

Your repository can only be at one commit at a given time

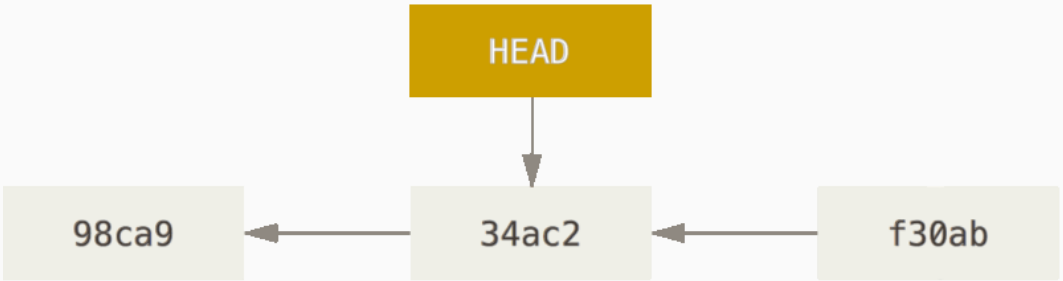

Figure 6: The HEAD can change location. *Source: Git Pro book*

You can *checkout* an other commit
The HEAD pointer will move accordingly

Making commits

Git is a watchdog

Git is a watchdog
All file changes are monitored

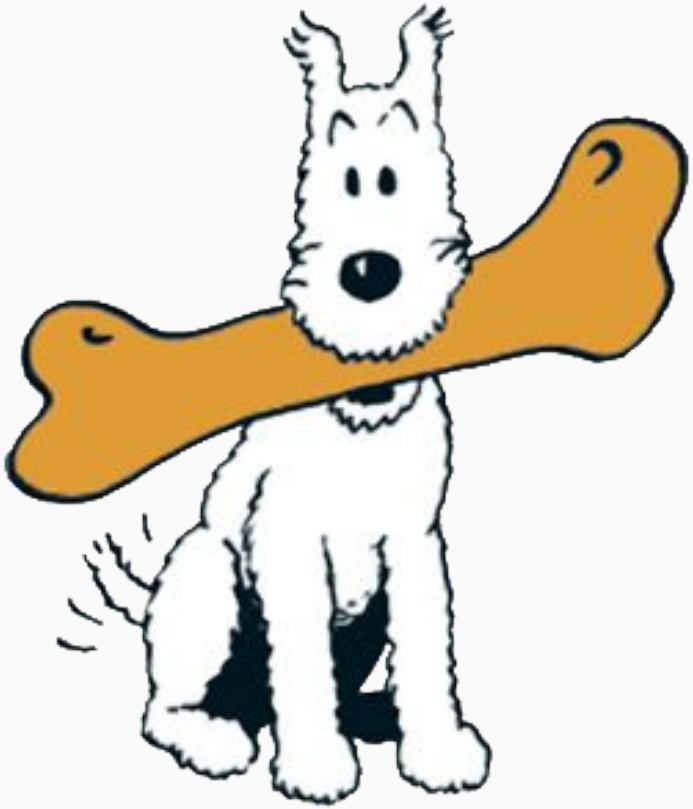

Figure 7: *Source: Hergé*

Four file states

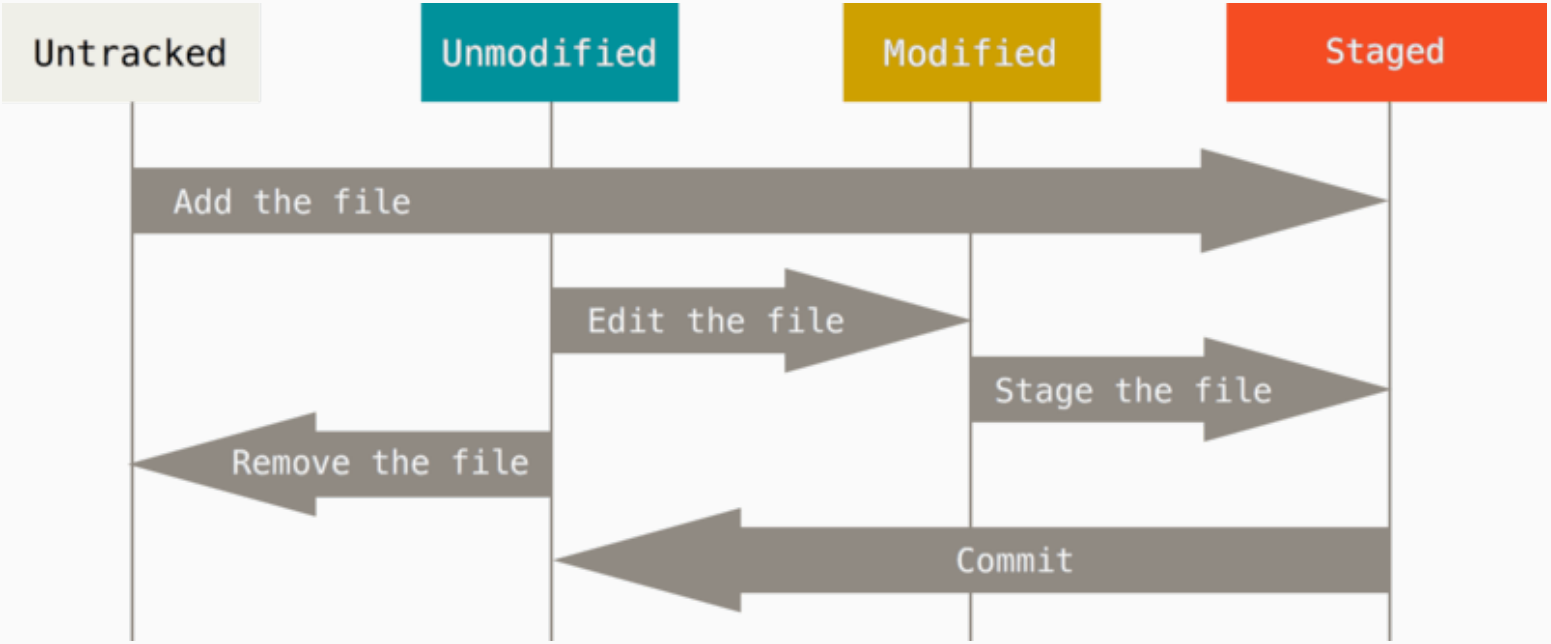

Figure 8: Every file is in one of four possible states. Different actions alter the state of a file. *Source: Git Pro book*

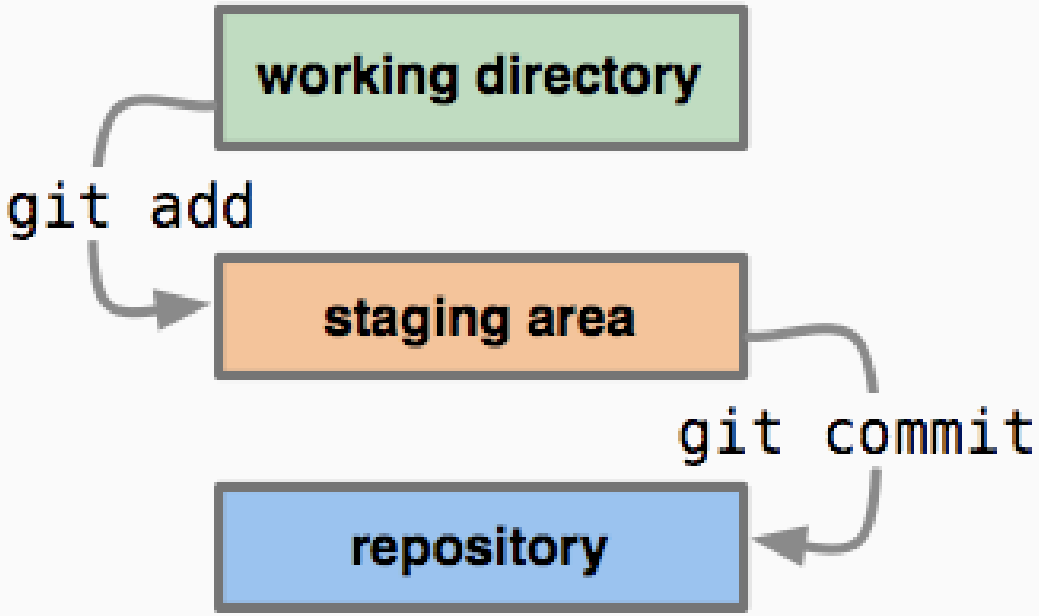

Figure 9: The Git add and commit commands move file changes from the working directory to the repository. *Source: Scott Chacon*

Making commits in Sourcetree

Sourcetree interface

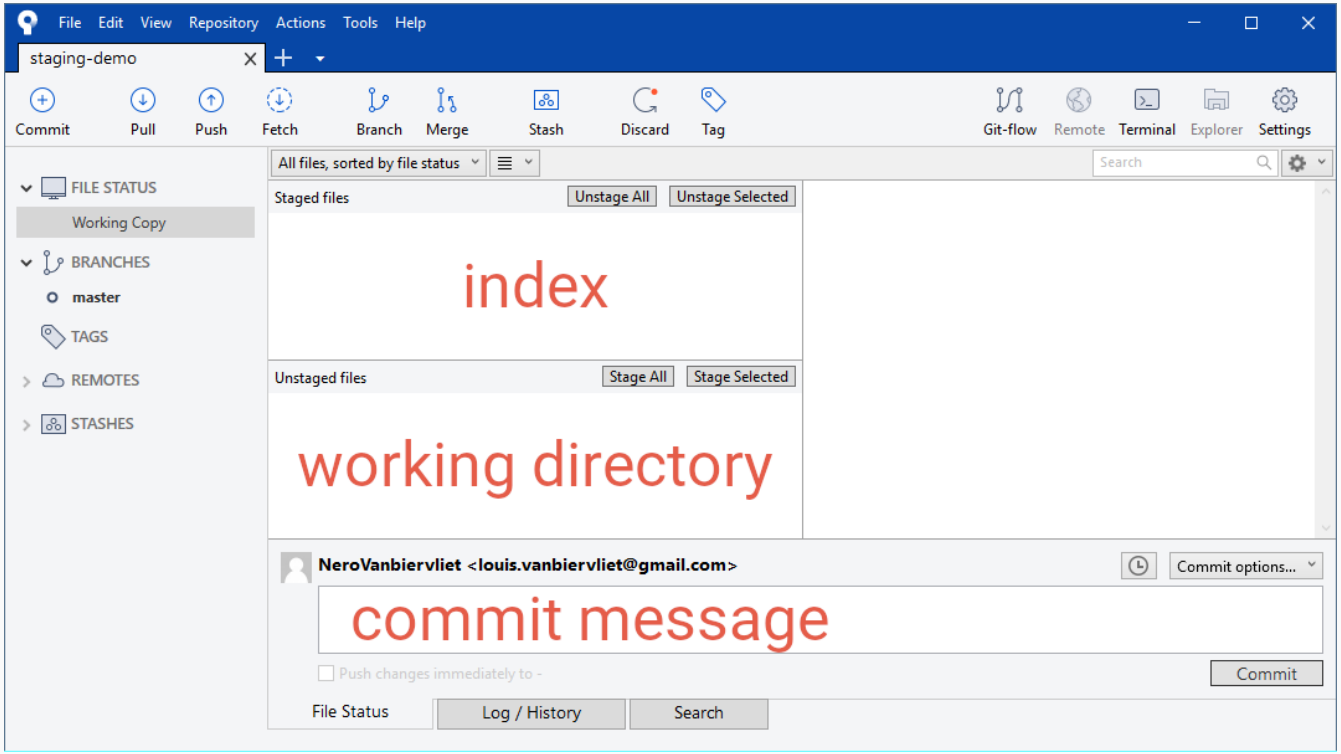

Figure 10: The Sourcetree interface divides the index (staging area) from the working directory. Commit messages can be entered in the textbox.

Icons

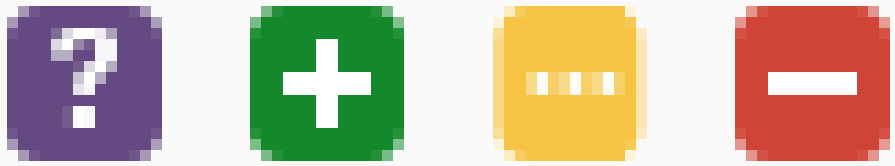

Figure 11: The Sourcetree icons for file states.

From left to right

- untracked
- added
- modified
- removed

Ignoring files

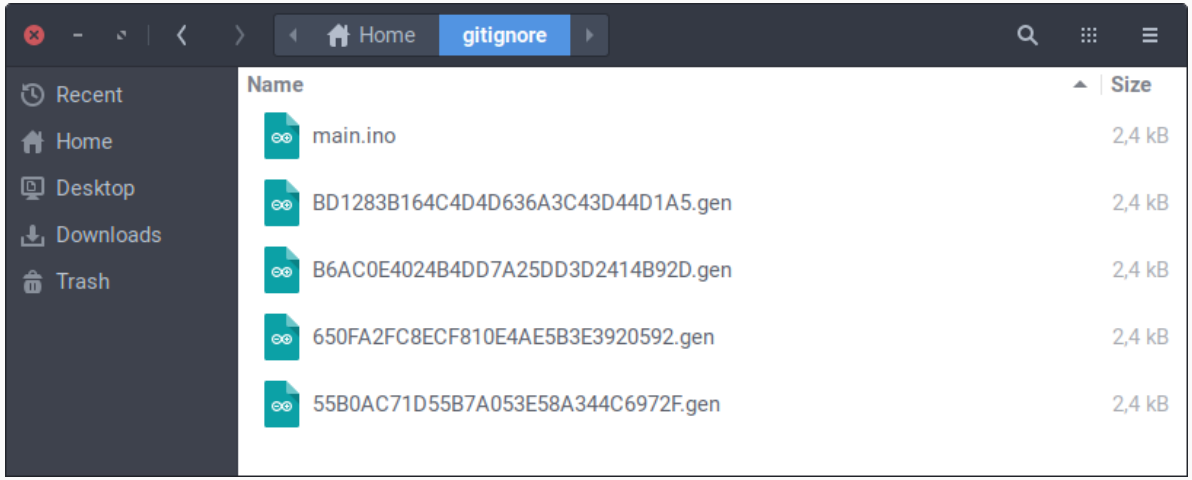

Figure 12: Automatic generated files are best ignored.

You can tell Git to ignore files

Useful for automatically generated files

Configuration is done using a .gitignore file

- Recording Changes to the Repository
- Gitignore documentation

Exercises

Exercise 2.1

Continue working on the `adam-and-eve` repo

1. Take a look at this repository in your file system and make a mental note of the files that are in it
2. Checkout the commit with message ”create first woman”
3. **Question:** what happened in your file system?
4. At some point, Adam found a secret paper message (a file was created). It was later destroyed (the file was deleted). Find the commit where the file still exists and checkout this commit
5. Verify that you can now read the lost file

Exercise 2.2

Continue working on the `adam-and-eve` repo

1. Checkout the commit with message ”remove pluto”
2. Add your name to `people.txt`
3. Stage and commit your change
4. Verify that a new commit has been created. It should be displayed above the commit ”remove pluto”

Exercise 2.3

Clone the repository setting-the-stage-zip. Extract the directory from the `zip` file and place it in the same directory as your other repositories. Add this directory as a repository in Sourcetree with the *Add* button

1. If you did well, this repository should contain some modified files in the working directory that are not yet committed. Verify this
2. Commit these changes, but not in one commit. Often, it is better to split your work in coherent small pieces. Split the changes in **three commits**. *Hint: you can stage a part of a file*

Exercise 2.4 (optional)

Create a new repository for this exercise

1. In the repository, make a .gitignore file
2. Add a rule to ignore .txt files
3. Commit your changes
4. Add a new text file in your repository
5. Verify that this file does not show up in Sourcetree because it is ignored

Vocabulary

1
repository
clone

working directory
staging area/index
gitignore

2
commit
commit hash
checkout
HEAD
untracked
unmodified
modified
staged

3
branch
remote
ahead/behind
merging
master
develop

4
remote
origin
push
fetch
pull

5
fast forward
three-way
rebase
merge conflict
ours & theirs

6
feature
release
tag
hotfix
versioning

7
amend
revert
stash
reset