‹ (../01-local/)

Introduction to Git and GitHub (../)

› (../03-branching/)

Looking at history and differences

Overview

Teaching: 20 min
Exercises: 0 min
Questions

- How does Git store information?

Objectives

- Be able to view history of changes to a repository
- Be able to view differences between commits

Looking at differences

We forgot to reference a second paper in the introduction section. Correct it, save the file but do not commit it yet. We can review the changes that we made using:

```
$ nano journal.md              # Add second reference to introduction
$ git diff journal.md          # View changes to file
```

This shows the difference between the latest copy in the repository and the unstaged changes we have made.

- `-` means a line was deleted.
- `+` means a line was added.
- Note that a line that has been edited is shown as a removal of the old line and an addition of the updated line.

Looking at differences between commits is one of the most common activities. The `git diff` command itself has a number of useful options (http://git-scm.com/docs/git-diff.html).

Now commit the change we made by adding the second reference:

```
$ git add journal.md
$ git commit                   # "Reference second paper in introduction"
```

Looking at our history

To see the history of changes that we made to our repository (the most recent changes will be displayed at the top):

```
$ git log
```

```
commit 4dd7f5c948fdc11814041927e2c419283f5fe84c
Author: Your Name <your.name@manchester.ac.uk>
Date:   Mon Jun 26 10:21:48 2017 +0100

    Write introduction

commit c38d2243df9ad41eec57678841d462af93a2d4a5
Author: Your Name <your.name@manchester.ac.uk>
Date:   Mon Jun 26 10:14:30 2017 +0100

    Add author and title
```

The output shows (on separate lines):

- the commit identifier (also called revision number) which uniquely identifies the changes made in this commit
- author
- date
- your commit message

Git automatically assigns an identifier (e.g. 4dd7f5) to each commit made to the repository — we refer to this as *COMMITID* in the code blocks below. In order to see the changes made between any earlier commit and our current version, we can use `git diff` followed by the commit identifier of the earlier commit:

```
$ git diff COMMITID            # View differences between current version and COMMITID
```

And, to see changes between two commits:

```
$ git diff OLDER_COMMITID NEWER_COMMITID
```

Using our commit identifiers we can set our working directory to contain the state of the repository as it was at any commit. So, let's go back to the very first commit we made,

```
$ git log
$ git checkout INITIAL_COMMITID
```

We will get something like this:

```
Note: checking out '21cfbdec'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b new_branch_name

HEAD is now at 21cfbde... Add title and authors
```

If we look at `journal.md` we'll see it's our very first version. And if we look at our directory,

```
$ ls
```

```
journal.md
```

then we see that our `common` directory is gone. But, rest easy, while it's gone from our working directory, it's still in our repository. We can jump back to the latest commit by doing:

```
$ git checkout master
```

And `common` will be there once more,

```
$ ls
```

```
common journal.md
```

So we can get any version of our files from any point in time. In other words, we can set up our working directory back to any stage it was when we made a commit!

If we want to make a commit now, we should create a new branch to retain these commits. If we created a new commit without first creating a new branch, these commits would not overwrite any of our existing work, but they would not belong to any branch. In order to save this work, we would need to checkout a new branch. To discard any changes we make, we can just checkout master again.

📌 Where to create a Git repository?

Avoid creating a Git repository within another Git repository. Nesting repositories in this way causes the 'outer' repository to track the contents of the 'inner' repository - things will get confusing!

Key Points

- `git log` shows the commit history
- `git diff` displays differences between commits
- `git checkout` recovers old versions of files

‹ (../01-local/)

› (../03-branching/)

Edit on GitHub (https://github.com/emdupre/git-course/edit/gh-pages/_episodes/02-history.md) / Contributing (https://github.com/emdupre/git-course/blob/gh-pages/CONTRIBUTING.md) / Source (https://github.com/emdupre/git-course/) / Cite (https://github.com/emdupre/git-course/blob/gh-pages/CITATION) / Contact (mailto:elizabeth.dupre@mail.mcgill.ca?subject=Git course)