DataCamp_Notes_Introduction-to-Git

Chapter 1: Introduction to Git

Video 1.1: Introduction to Version Control

version control ::: processes and systems to manage changes to files, programs, and directories
What should be version controlled? :: version control is useful for anything that *changes over time* or *needs to be shared.*
What can version control do? (4) :: (1) track files in different states, (2) combine different versions of files, (3) identify a particular version, (4) revert changes.
Why is version control important? :: A project without version control is like cooking without a recipe -- it'll be difficult to remember how to produce the same results again.

Imagine we work for an e-commerce company and release a new feature on our website that recommends products to customers. However, there is a bug in our code, and the website stops working! With version control, we can easily revert our website to a previous working version, then work separately to identify the issue before safely re-releasing the new feature.

So, how do we perform version control? :: One popular program for version control is called Git. Git is *open source* and *scalable* to easily track everything from small solo projects to complex collaborative efforts with large teams!

4 benefits of Git :: (1) git stores everything, so nothing is lost. (2) we can compare files at different times. (3) see what changes were made, by who,

and when. (4) if something goes wrong, we can revert to previous versions of files!

Using Git:

- Git commands are run on the shell, also known as the terminal.
- The shell :: is a program for executing commands and can be used to easily preview or inspect files and directories
- Directory = :: folder

Useful terminal commands

`pwd` ::: terminal command to print current working directory, e.g. `home/repl/Documents` ls `::: terminal command to print the contents of the current directory` cd `::: terminal command to change to a different directory, e.g.` cd archive git --version` ::: terminal command to print which version of Git we have installed.

Video 1.2: Creating repos

What is a Git repo?

Git repo ::: directory containing files and subdirectories, and Git storage. ***DO NOT EDIT*** `.git`
4 benefits of repos :: (1) systematically track versions, (2) revert to previous versions, (3) compare versions at different points in time, (4) collaborate with colleagues.

create a new repo
??

```
git init <project name>
git init mental-health-workspace
```

to change into the new repo we just created, we use
??

```
cd mental-health-workspace
```

to check that the repo was initialized correctly, use
??

```
git status

returns:
On branch main

No commits yet

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

to convert an existing project into a repo, use
??

```
git init
--run this command from within the project directory
returns:
Initialized empty Git repository in /home/repl/mental-health-
workspace/.git/
```

What is being tracked?

`git status` shows that there are modified files not being tracked.

Nested repositories

- Do not create a Git repo inside another Git repo
    - Known as nested repos
- There will be two `.git` directories
- Which `.git` directory should be updated?

Video 1.3: Staging and committing files

Git workflow

- Edit and save files on our commputer
- Add the file(s) to the Git staging area
    - Tracks what has been modified
- Commit the files (to save them)
    - Git takes a snapshot of the files at that point in timme
    - Allow to compare and revert files.
- Staging: like putting a letter in an envelope
- Committing: like dropping it in the mailbox.

Adding to the staging area

add a single file to the staging area
??

```
git add README.md
```

add all modified files in the current directory and its subdirectories
??

```
git add .
```

`.` :: all files in the current directory and sub-directories

Making a commit

to make a commit with a comment
??

```
git commit -m "Adding a README."
```

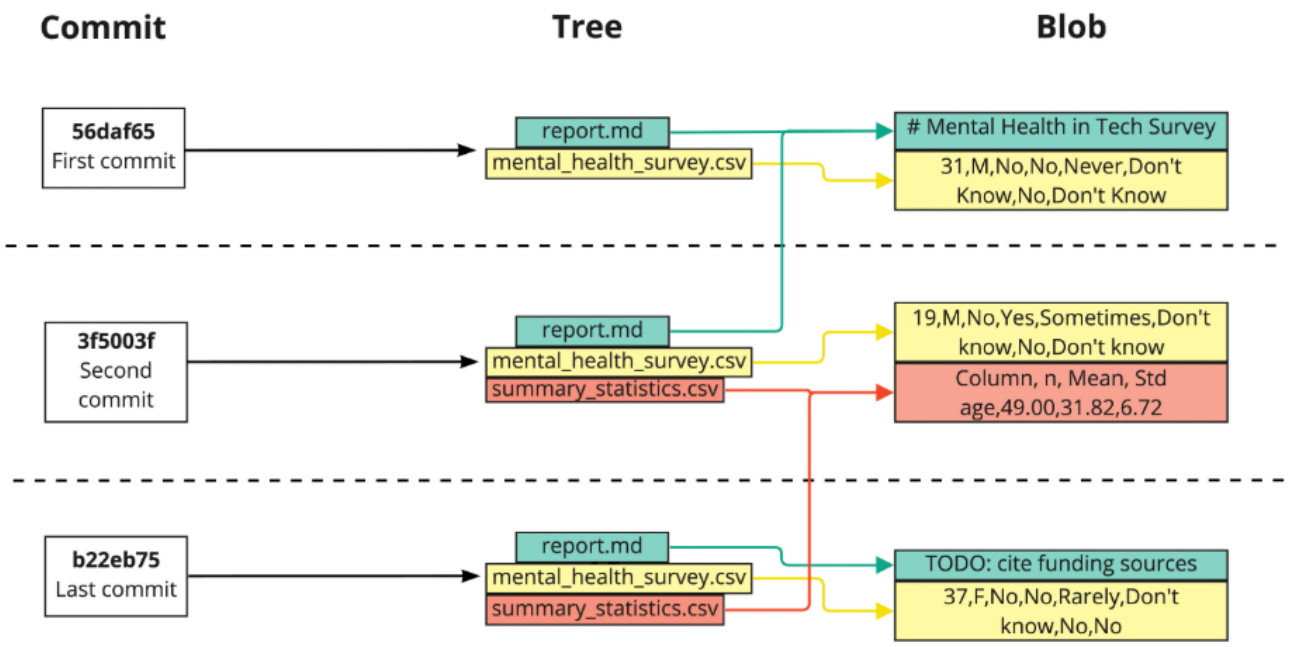
the `-m` flag :: allows you to include a log message without opening a text editor

Chapter 2: Version History

Video 2.1: Viewing the version history

The commit structure

Git commits have three parts ::: (1) Commit, (2) Tree, (3) Blob
the commit contains :: the metadata - author, log message, commit time
the tree ::: tracks the names and locations of files and directories in the repo; like a dictionary - mapping keys to files/directories
Blob stands for :: Binary Large Object and it may contain data of any kind. They contain a compressed snapshot of a file's contents when the commit happened,



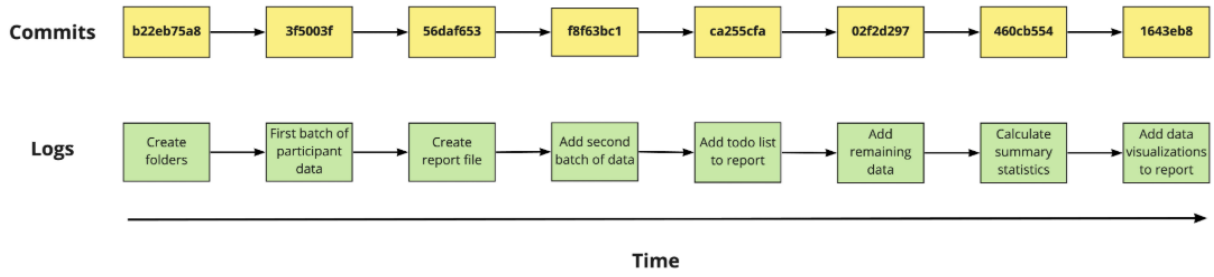
Git hash

- 40 character string of letters and numbers, ie `b22eb75a82a68b9c0f1c45b9f5a9b7abe281683a`
- Pseudo-random number generator—hash function
- Hashes allow data sharing between repos
  - If two files are the same,
    - then their hashes are the same
  - Git only needs to compare hashes
- show commits from newest to oldest ::: `git log`
- Press `space` to show more recent commits
- Press `q` to quit the log and return to the terminal

Video 2.2: Version history tips and tricks

Projects grow!

- Larger project = more commits = larger output



Restricting the number of commits

- We can restrict the number of commits displayed using `-`.
  Restrict `log` to the `3` most recent commits
  ??

```
git log -3
```

Restricting the file

To only look at the commit history of one file
??
`git log report.md`

Combining techniques

```
cd data
```

```
git log -2 mental_health_survey.csv
```

Git log output

```
commit f35b9487c063d3facc853c1789b0b77087a859fa
Author: Rep Loop <repl@datacamp.com>
```

```
Date: Fri Jul 26 15:14:32 2024 +0000

        Add two new participants' data.

commit 7f71eadea60bf38f53c8696d23f8314d85342aaf
Author: Rep Loop <repl@datacamp.com>
Date: Fri Jul 19 09:58:21 2024 +0000

        Adding fresh data for the survey.
```

Customizing the date range

restrict `git log` by date
??

```
git log --since='Month Day Year'
```

commits since 2nd April 2024:
??

```
git log --since='Apr 2 2024'
```

commits between 2nd and 11th April;
??

```
git log --since='Apr 2 2024' --until='Apr 11 2024'
```

Acceptable filter formats

- Natural language
  - `"2 weeks ago"`
  - `"3 months ago"`

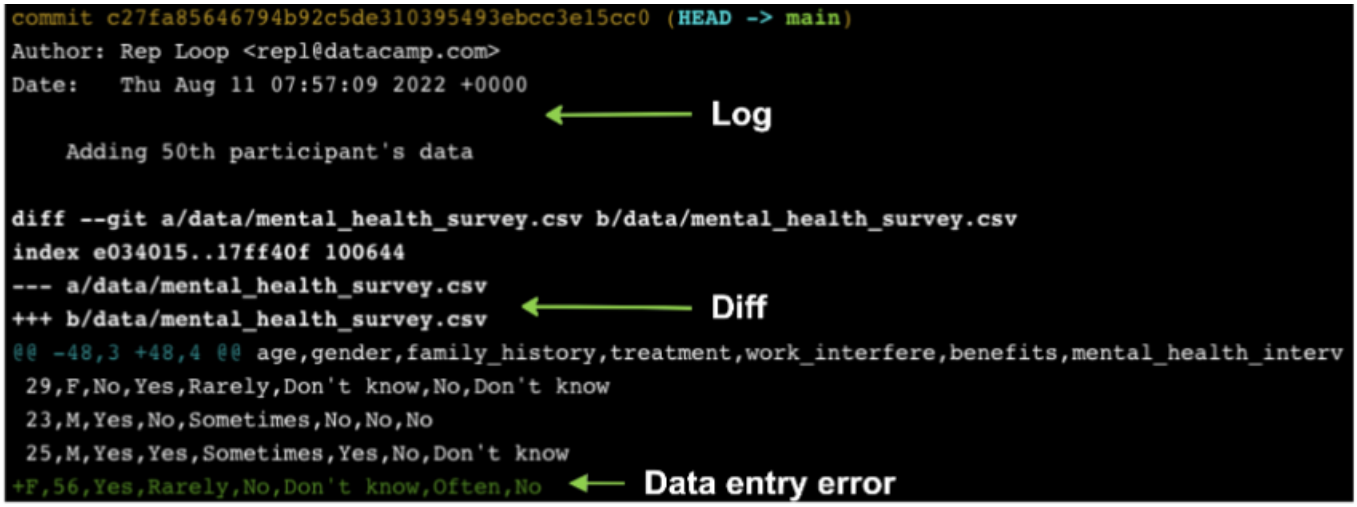
    - `"yesterday"`
- Date format
    - `"07-15-2024"`
        - recommend ISO Format 8601 `"YYYY-MM-DD"`
        - check system settings for compatibility, e.g. `12-06-2024` could be `6th Dec` or `12th June`!
    - `"15 Jul 2024"` or `"15 July 2024"`
        - Invalid: `15 Jul, 2024`

[Reference ISO 8601 Date Time Format]

Finding a particular commit

- only need the first 8-10 characters of the hash

```
git show c27fa856
```



Video 2.3: Comparing versions

`git diff` :: difference between versions
compare last committed version of `report.md` with latest version **not** in the staging area:
??

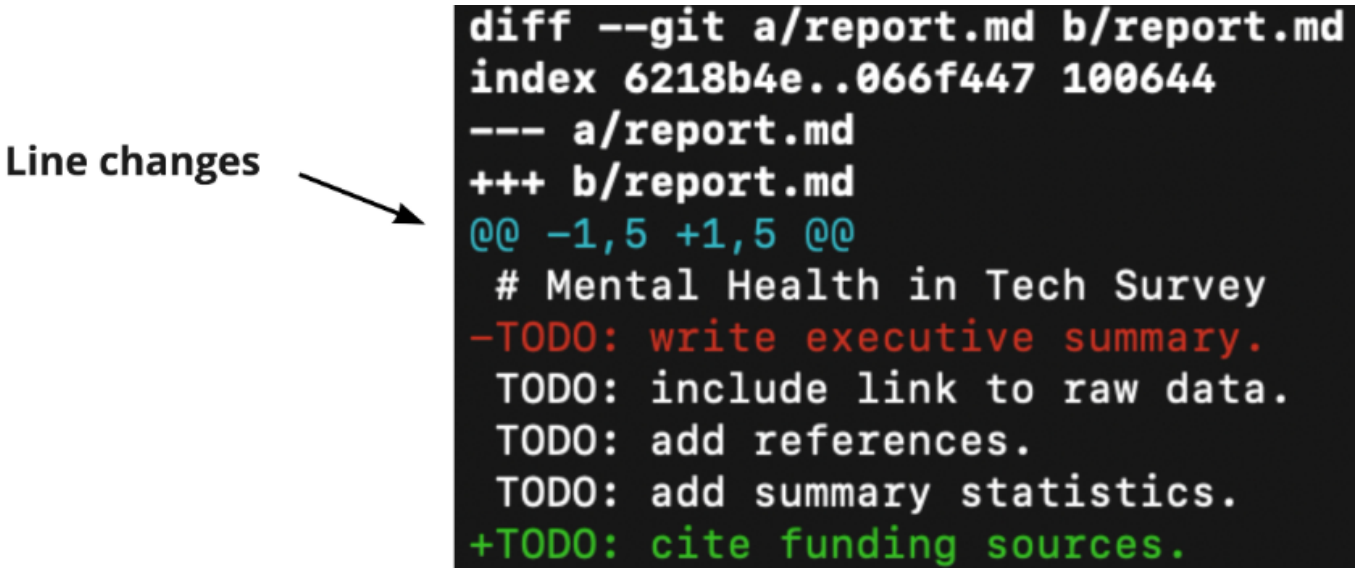
```
git diff report.md
```

Comparing to an unstaged file

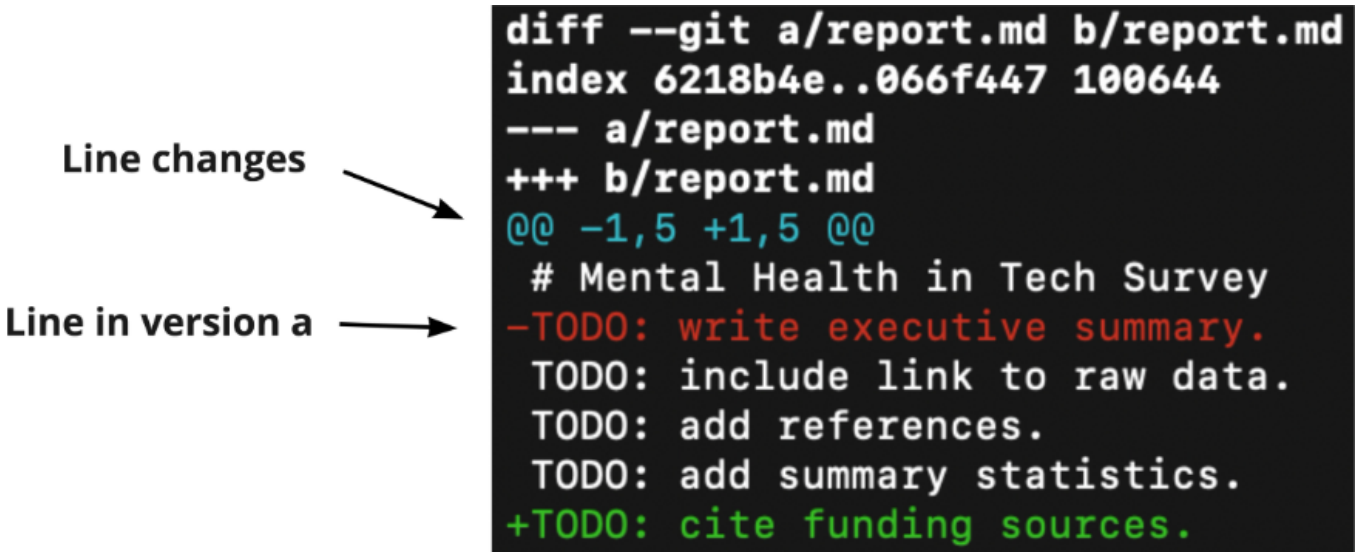
```
diff --git a/report.md b/report.md
index 6218b4e..066f447 100644
--- a/report.md
+++ b/report.md
@@ -1,5 +1,5 @@
 # Mental Health in Tech Survey
-TODO: write executive summary.
 TODO: include link to raw data.
 TODO: add references.
 TODO: add summary statistics.
+TODO: cite funding sources.
```

The output shows two versions of the report: A is the last version that has been committed, and B is the version that we still need to add to the staging area. Generally speaking, version B will be the newest version.

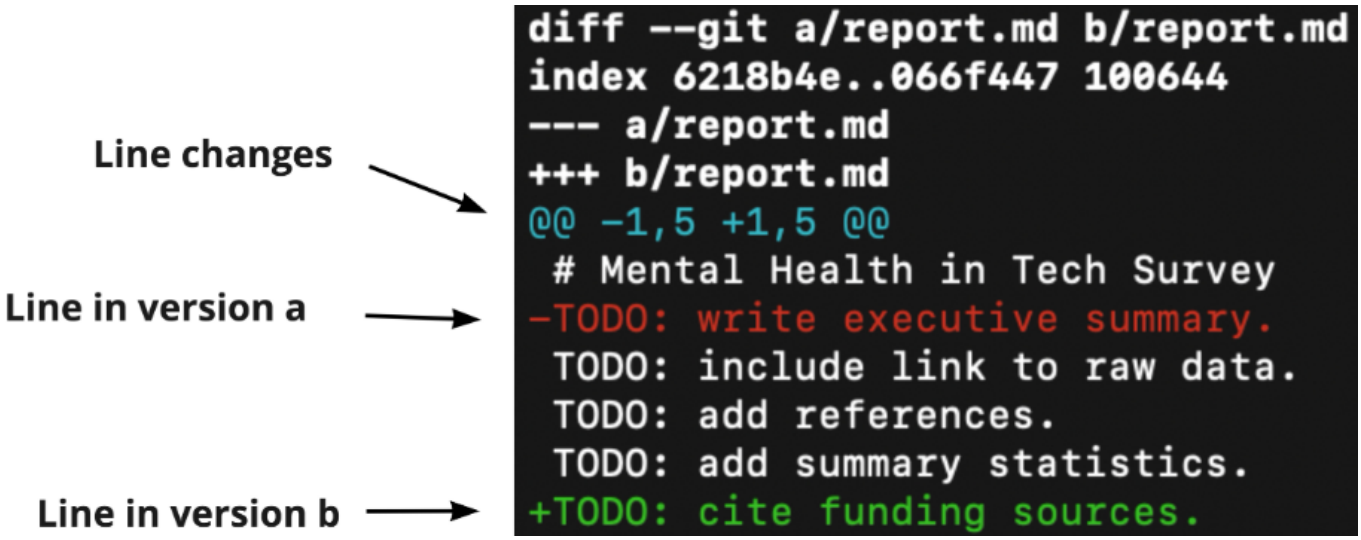
Git diff output



The line starting with two at symbols tells us what changed between the two versions. The minus 1 and 5 indicate that version A starts at line 1 and has 5 lines, and the plus 1 and 5 show that version B also starts at line 1 and has 5 lines. This makes sense when we look at the next part of the output.



We finished our executive summary and deleted this task from the report in our latest commit, for version A. This is shown by the red text starting with a minus symbol, representing a line in version A that is not in version B.



We added a task in version B of the report, which is not in version A. It is shown in the final line with green text starting with a plus symbol.

Comparing to a staged file

Add `report.md` to the staging area
??
`git add report.md`

Compare last committed version of `report.md` with the version **in the staging area**
??

```
git diff --staged report.md
```

We get the exact same output as previously, given no further changes were made to the file now in the staging area.

Comparing multiple staged files

Compare **all staged files** to versions in the last commit:
??

```
git diff --staged
```

```
diff --git a/mh_tech_survey.csv b/mh_tech_survey.csv
index 4208ed3..d758efb 100644
--- a/mh_tech_survey.csv
+++ b/mh_tech_survey.csv
@@ -47,3 +47,4 @@ age,gender,family_history,treatment,work_interfere,
ntal_health_interv
 28,M,No,Yes,Rarely,Yes,No,Yes
 29,F,No,Yes,Rarely,Don't know,No,Don't know
 23,M,Yes,No,Sometimes,No,No,No
+37,F,No,No,Rarely,Don't know,No,No
diff --git a/report.md b/report.md
index 6218b4e..066f447 100644
--- a/report.md
+++ b/report.md
@@ -1,5 +1,5 @@
 # Mental Health in Tech Survey
-TODO: write executive summary.
 TODO: include link to raw data.
 TODO: add references.
 TODO: add summary statistics.
+TODO: cite funding sources.
```

Comparing two commits

Two options: (1)
Find the commit hashes:
?

```
git log
```

Compare the commits:
?

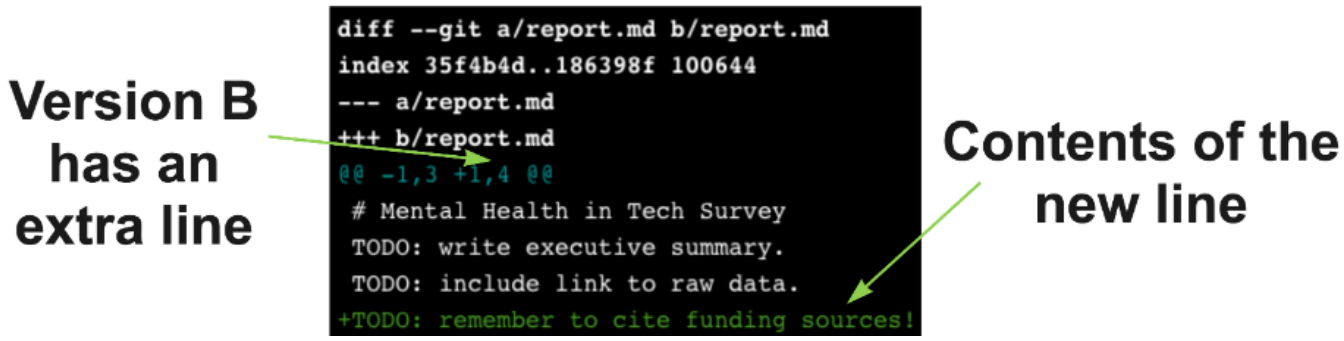
```
git diff 35f4b4d 186398f
```

- shows what changed from first hash to second hash
    - put most recent commit hash second.

(2) the word `HEAD` in capitals can be used to refer to the most recent commit.

compare second most recent with the most recent commit
??

```
git diff HEAD~1 HEAD
```



Summary

`git diff` ::: show changes between all unstaged files and the latest commit

`git diff report.md` ::: show changes between an unstaged file and the latest commit

`git diff --staged` ::: show changes between all staged files and the latest commit

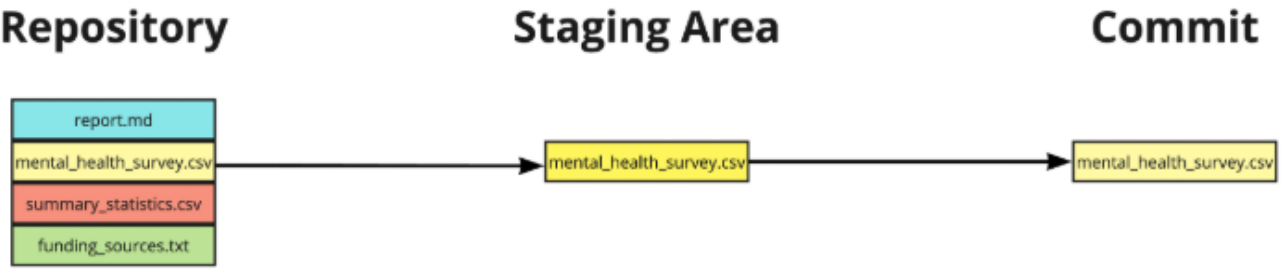
`git diff --staged report.md` ::: show changed between a staged file and the latest commit

`git diff 35f4b4d 186398f` ::: show changes between two commits using hashes

`git diff HEAD~1 HEAD~2` ::: show changes between two commits using `HEAD` instead of commit hashes

Video 2.4: Restoring and reverting files

Making an error



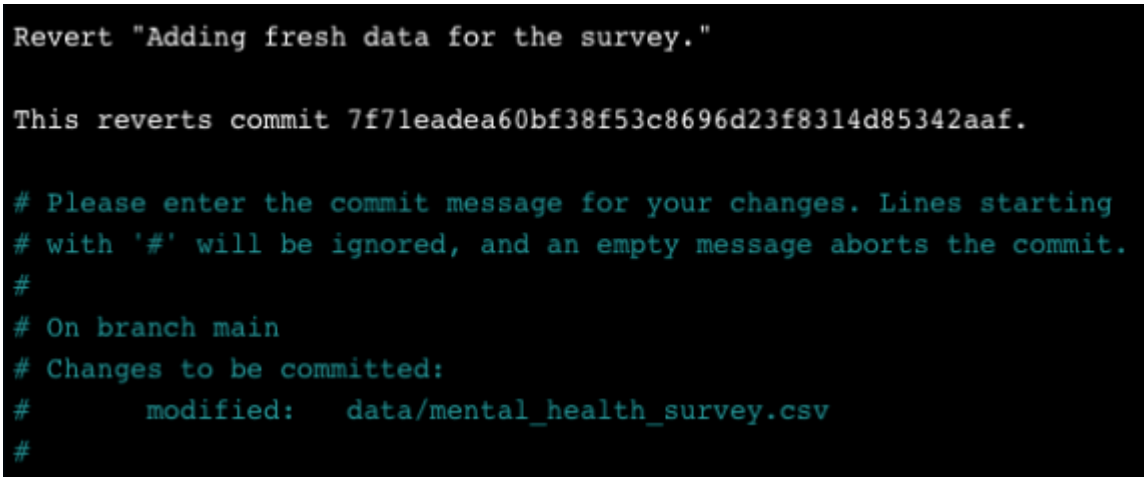
Suppose we've modified a file, added it to the staging area, and made a

commit. However, we've realized the last edit has a typo.

Reverting files

restoring a repo to the state prior to the previous commit
`git revert` ::: (1) reinstates previous versions and makes a commit, (2) restores all files updated in a given commit, and (3) we need to provide a reference through a commit hash or `HEAD` to the changes that we want to undo.

```
git revert HEAD
```



to save in the text editor, press ::: `Ctrl + O` then `Enter`
to exit the text editor ::: `Ctrl + X`
exiting the text editor, we see :: a terminal output confirming the revert, including the number of files and lines that were changed.

```
[main 7d11f79] Revert "Adding fresh data for the survey."
Date: Tue Jul 30 14:17:56 2024 +0000
1 file changed, 3 deletions(-)
```

git revert flags

avoid opening the text editor during `revert` ::: `git revert --no-edit HEAD`
revert last commit without committing (bring files into staging area) ::: `git revert -n HEAD`

Revert a single file

- `git revert` works on commits, not individual files
- to revert a single file: `git checkout` + use commit hash or head
  syntax:
  to revert `report.md`
  ??

```
git checkout HEAD~1 -- report.md
```

Checking the checkout

```
git status
```

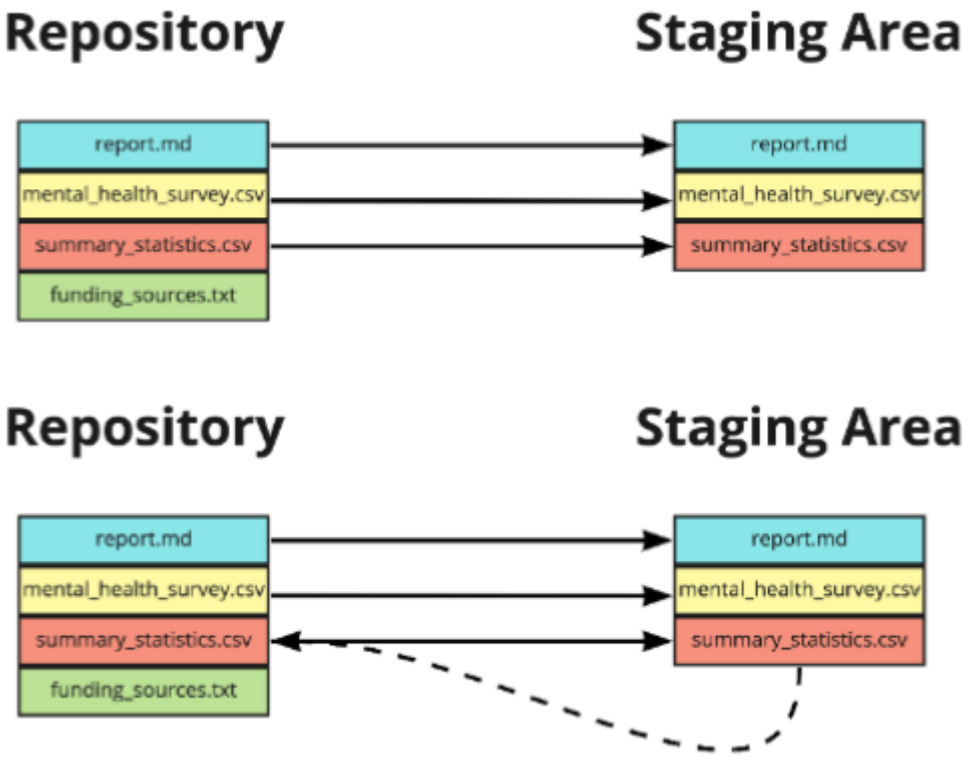
```
On branch main
Changes to be committed:
        (use "git restore --staged <file>..." to unstage)

                modified: report.md
```

Making a commit

```
git commit -m "Checkout previous version of report.md"
```

```
[main daa6c87] Checkout previous version of report.md
1 file changed, 1 deletion(-)
```

Unstaging a file



to unstage a single file:
??

```
git restore --staged summary_statistics.csv
```

- edit the file

```
git add summary_statistics.csv
```

```
git commit -m "Adding age summary statistics"
```

to unstage all files:
??

```
git restore --staged
```

Summary:

`git revert HEAD` ::: revert all files from a given commit

`git revert HEAD --no-edit` ::: revert without opening a text editor

`git revert HEAD -n` ::: revert without making a new commit

`-n` :: no commit

`git checkout HEAD~1 -- report.md` ::: revert a single file `report.md` from the previous commit

`git restore --staged report.md` ::: remove a single file from the staging area

`git restore --staged` ::: remove all files from the staging area