Working with `git` branches and looking around

- Create an empty directory, for example `git_experiment`.
- Create a text file in the directory, for example `fun.txt`. My initial content of that file is

  When my boss asked me who is the stupid one, me or him? I told him everyone knows he doesn't hire stupid people.

  Rating: 7

  ---

  a. Doctor, how many autopsies have you peformed on dead people?
  b. All my autopsies have been performed on dead people.

  Rating: 2

  Commit early, commit often. A tip for version controlling - not for relationships

  Rating: 6

  People used to laugh at me when I would say "I want to be a comedian", well nobody's laughing now.

  Rating: -1

Initialize the repository and add the first file.

- `git init` to initialize a repository.
- `git status` - it states that we have an untracked files.
- `git add fun.txt` so that git starts tracking the file.
- `git status`
- `git commit -m "first version of my best jokes"`
- `git status`
- `git log` to see the list of all the commits and their IDs

Add more jokes to the file and create another commit.

- Edit the `fun.txt` file to add a new joke. For example "How does the ocean say hello? It waves."
- `git status`
- `git commit -m "added new joke, my favorite" fun.txt`
- `git log`

Look at the differences between these two commits:

- `git diff ID1 ID2`
- `git diff ID2 ID1`

Add one more joke to the file, but do not commit right away.

- Edit the `fun.txt` file to add a new joke. For example “Why shouldn’t you write with a broken pencil? Because it’s pointless.”
- `git diff`
- `git add fun.txt`
- `git commit -m "another good one, this time about pencils"`

Create another file and add that to the repository.

- Create a file called, “no-jokes.txt”
- `git add no-jokes.txt`
- `git commit -m "created file for not funny jokes"`
- `git log` now shows four commits

How are these files stored?

- `git cat-file -p HEAD` shows basic info about the commit (HEAD is the most recent commit), for example

  tree 46c3efef78abfdabf6c8efda85d39570681830a7 parent 4911b524f27a269bc871a535207d728e8b2f16d8 author JoannaKl joannakl@cs.nyu.edu 1739131950 -0500 committer JoannaKl joannakl@cs.nyu.edu 1739131950 -0500

  created file for not funny jokes

- `git cat-file -p 46c3efef78` - use the ID for the tree to look at the content of the repo at that commit, for example

  100644 blob 243bb35f9278085764e8d3dca8beb705f685b1b9 fun.txt 100644 blob ba4dbb4d9a4fcc5969f38f6afbf9a150a38b6845 no-jokes.txt

- `git cat-file -p 243bb35` shows you the content of the particular file

Let’s accidently remove out files

- remove the two files we created before
- `git status` tells you they were removed, but git can restore them
- `git restore fun.txt`
- `git restore no-jokes.txt`

Let’s look at the current branch and create a new one

- `git reflog main`

- `git branch potential_jokes`
- `git branch` shows two branches, the one marked is our current branch
- `git checkout potential_jokes`
- `git branch`
- `git log --graph --decorate --oneline --all` shows a view of two branches (even though they do not look like two branches yet)

Modify the `potential_jokes` branch:

- Add new joke to the file `fun.txt` in the `potentia_jokes` branch.
- `git commit -m "add a joke about atoms" fun.txt` to commit this change
- Create a new file called `computer_jokes.txt` with “Why do programmers wear glasses? Because they can’t C#.”
- `git add computer_jokes.txt` tell git to add this file to tracking files and stage it
- `git commit -m "new file for computer jokes"`
- `git log --graph --decorate --oneline --all` now shows that two branches are separate: main is two commits behind potential jokes

See what happens with the content of the current working directory when we jump between branches and commits:

- `git checkout main`
- `git checkout potential_jokes`
- `git log --graph --decorate --oneline --all`

Make some changes in the main branch

- `git checkout main`
- create a file called cat_jokes.txt and add “Why was the cat afraid of the tree? Because of its bark!” to it
- `git add cat_jokes.txt`
- `git commit -m "jokes about cats"`
- `git log --graph --decorate --oneline --all` to look at both branches, pick a commit ID that is not HEAD
- `git checkout THAT_ID`
- READ WHAT GIT TELLS US, it seems that we are in a “detached head” state
- `git checkout potential_jokes` to get back to the most recent commit in the branch

Let’s combine the changes from the potential_jokes branch into the main branch.

- `git log --graph --decorate --oneline --all`

- `git merge potential_jokes` results in a smooth merge since we have no conflicts that git cannot resolve itself, this merge produces a new merge commit

We'll continue working with potential_jokes branch

- edit the file fun.txt by changing the ratings on the jokes
- `git commit -m "changed ratings" fun.txt`
- `git chouckout main`
- DO YOU THINK WE WILL HAVE A MERGE CONFLICT?
- `git merge potential_jokes`

Another attempt for a merge conflict:

- edit the file fun.txt in main branch by changing all ratings to zero
- `git commit -m "set all ratings to zero" fun.txt`
- edit the file fun.txt in potential_jokes branch by changing all ratings to five
- `git checkout potential_jokes`
- `git commit -m "set all ratings to five" fun.txt`
- go back to main branch and merge
- `git checkout main`
- `git merge potential_jokes` - read the output carefully, look at fun.txt tool
- either edit fun.txt directly to resolve the merge conflict or run `git mergetool` (to configure the mergetool used to resolve merge conflicts run `git config --global merge.tool meld` - in my case it is `meld`
- `git commit` to finish the merge