Advanced Exercises Set 3
Git

EECS 201 Winter 2021

Submission Instructions

This assignment is an "online assignment" on Gradescope, where you fill out your answers directly. These questions are reflected in the "Questions" section of each section.

1 Git Golf (10)

While I made some suggestions such as using branches as "backups" to make it easy to recover from issues that crop up during merges or rebases, sometimes mistakes are just made. Git can be remarkably forgiving: if you've ever told Git to remember something, then you can usually find a way to undo your mistake and get it back.

```
$ # Grab a copy of the files for this question
$
$ # If you want to use wget
$ wget https://www.eecs.umich.edu/courses/eecs201/files/assignments/adv2.tar.gz
$
$ # If you want to use curl
$ curl -O https://www.eecs.umich.edu/courses/eecs201/files/assignments/adv2.tar.gz
```

#1: Undeleting Files

In this repository, someone cleaning up ran `git rm *.png`, which deleted every image, and committed the result. Now the website is broken because an image that should have stayed was deleted. While you could use `git revert` to undo the commit, the commit also changed the website source, so you really don't want to undo everything.

Questions:

- ☐ What command did you use to recover the missing picture **without changing where `HEAD` and the current branch is** (e.g. without using `git revert` and `git reset` to undo a commit)? The only thing that should change in your directory is that the missing picture is now present: no other files should be modified as a result.

If you want a starting point, try looking at the log and using `git show` to see what a commit did. You can also try using `git diff --name-status` with a commit hash/reference to get a list of files whose status have changed.

#2: Undeleting Commits

Sometimes it is possible to lose a commit. This can happen because you deleted a branch, a rebase went poorly, or a reset went awry. Regardless of how it happened, there are ways of finding commits that "nothing" is currently pointing to. This repository has such a commit.

Questions:

- ☐ What command(s) did you use to recover the lost commit? Let's define "recover" as either moving your HEAD to the lost commit or being able to identify its hash.
- ☐ What is the hidden message? Note that it's in a file and it's not the commit message.

#3: Undeleting Changes

When working with Git, `git add my_file` stages a file, but it isn't actually committed until you run `git commit`. Sometimes you change your mind after a `git add` and run `git reset my_file` to unstage a file. The changes to that file are still there, however. To really undo changes, you use `git reset --hard` (or `git restore` or `git checkout`).

Once you've been using these commands for a little while, it can be a little too easy to accidentally type the wrong thing. In this repository, someone accidentally typed `git reset --hard` when they meant to just type `git reset`. Fortunately, because they had already run `git add` to stage their changes, the deleted changes can be recovered.
Figure out how to recover changes that had been staged for commit, but were then deleted, and then recompile the program.

Questions:

- ☐ What command(s) did you use to recover the initially staged contents?
- ☐ What is the output of running `$ ./main`? This isn't a trick question: if you successfully recover the contents of `main.c` it should now have code to print out a message.

2 and 3 involve deeper knowledge about the internals of Git. Hint: there is a section in *Pro Git* about data recovery.