Advanced Homework 4

Assigned: Friday, January 29, 11:00AM

Due: Before the first lecture on Friday

Submission Instructions

To receive credit for this assignment you will need to stop by someone's office hours, demo your running code, and answer some questions.

1 Git Golf

Sometimes, when using git, people make mistakes. 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
> wget http://c4cs.github.io/hw/c4cs-wk4-advanced/golf.tar.gz
```

#1: Undeleting Files

In this repository, someone got a little trigger happy and 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, it also changed the website source, so you really don't want to undo everything.
Demonstrate how to recover the missing picture without reverting the commit that deleted it.

#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.
Demonstrate how to recover a commit that nothing is currently pointing to.

#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`.

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.
Demonstrate how to recover changes that had been staged for commit, but were then deleted.

Submission checkoff:

- ☐ Explain how you solved each problem

2 Automating Professionalism

One feature of most version control systems is *hooks*. A hook is an automated script or tool that runs at various points in time. For example, later in this course we will show how you can automatically run test cases every time anyone commits code.

A slightly easier one to wire up, however, is a hook that will automatically check the spelling of your commit messages for you, letting you know if you made any mistakes. While spelling rarely counts for a class projects, it adds a nice bit of professionalism to any future work you'll share with others.

Like the regular homework, I recommend creating a temporary git repository to play with while you test things and try to get things working.

Git stores all of its information and configuration in a folder named `.git` in the root of each project. Navigate to `.git/hooks` and rename `commit-msg.sample` to `commit-msg`. This activates the commit message hook, which runs after you write and save your commit message. Now go back to your repository, make a change, and commit it. Hmm, doesn't look like anything changed. Make another change, but this time make this your commit message (**exactly this, capitalization matters**):

```
This is a test.

Signed-off-by: Me!
Signed-off-by: Me!
```

After trying to commit, type `git status`. What happened? Go back and look at the commit hook. Do you understand what it does? Try running `git commit --no-verify` with the same commit message. Go edit the commit hook and delete the line `exit 1`. Try making a new commit with the same commit message, what happens now? What is `$1` in this script? (not sure? try adding lines like `echo $1` or `echo $(file $1)` or `echo $(cat $1)` to the hook and making commits, what happens?)

`aspell` is a simple command-line utility that checks spelling. Install it and play with it a little.

Write a commit hook that checks the spelling of a commit message. Your hook should not prevent the commit from going through (that'd be annoying..), but it should print out any spelling errors. Your hook should also print a message that suggests running the command `git commit --amend`[1] if any errors are present.

Many commit messages will include snippets of code or a quote of a log or something else to help explain the commit. By convention, quoted blocks are indented with spaces.

```
Author: Brad Campbell <bradjc5@gmail.com>
Date:   Sun Apr 19 22:02:53 2015 -0700

Fix error on edit items page

Need to handle the string "None", which isn't a number, fixes:

  Traceback (most recent call last):
    ...<snip>...
    File "chezbetty-1.4.1.egg/chezbetty/templates/admin/item_edit.jinja2", line 56, in <module>
      <dt>Days Until Out</dt> <dd>{{ stats.until_out|round(0)|int }}</dd>
    File "Jinja2-2.7.3-py3.4.egg/jinja2/filters.py", line 657, in do_round
      return round(value, precision)
  TypeError: type str doesn't define __round__ method

Fixes #172.
```

It does not make sense to spell check the quoted material, however.

Modify your commit hook so that it does not spellcheck any line that begins with at least two spaces.

Hint: Having trouble? Don't be afraid to use `/tmp` to store some files. You don't need to make any temporary files to do this task, but it can make things easier, especially when you're first learning.

Submission checkoff:

- ☐ Show off your spellchecking hook in action
  - ☐ Demonstrate that it ignores errors in indented lines

[1] Amending a commit lets you change your most recent commit. This is fine when you are working locally, but can be a dangerous command if you are sharing your repository with anyone else. Use with caution.

Further exploration and some gotchas:

- Take a look at some of the other hooks, are any of them useful?
- For a list of all available hooks, type `man githooks`.
- Hooks have to be marked as executable to run (`chmod +x`). The sample hooks already had the executable bit set, which is why renaming the existing sample worked above.
- Hooks can call other scripts. Because invocation of hooks is controlled by the name of the script, if you want multiple scripts to run for a single hook, you'll need to have one script named correctly that calls your other hooks.