Advanced Exercises Set 5
More Bash

EECS 201 Winter 2020

Submission Instructions

To receive credit for this assignment you will need to stop by someone's office hours, demo your running code/system, and answer some questions. You may choose the exercises that you wish to do from this set. Each exercise is denoted by its point count. **Extra credit is given for early turn-ins of advanced exercises. These details can be found on the website under the advanced homework grading policy.**

Preface

I highly suggest that you do this in a Linux environment, be it WSL on Windows (on the Linux filesystem, not the Windows filesystem) or your Ubuntu virtual machine. The reasoning for this is that some tools that deal with regular expressions (namely for this homework: `sed` and `grep`) may differ in behavior depending on *nix system. Linux systems use GNU `sed` and `grep` while macOS (and FreeBSD) use BSD `sed` and `grep` which have some subtle differences in behavior.

There are also some provided example files.

```
curl -O https://www.eecs.umich.edu/courses/eecs201/files/assignments/adv5-files.tar.gz
tar xzf adv5-files.tar.gz
```

1 Automating security (5)

When you are developing a web application, sometimes you need to access the services of some API to access things like weather information from one source or a user's Google calendar from another.

Such services tend to require the use of an *API key* that is used to authenticate the client requesting services and can be used to do things like rate limiting and tracking API use.

If someone else (say, another competing application) gets a hold of your application's API key, they could mooch off of whatever quotas that API key has.

Another security issue is when people unwittingly put private SSH keys into a repository. Recall that when you generated an SSH key it created a private and public key pair (e.g. public: `id_rsa.pub`, private: `id_rsa`.) The public key, being "public" is provided to various servers to authenticate your computer with. Your private key is what verifies that you are you: if it gets out of your hands, others can access things that have saved your public key for authorization. (This is simplifying it greatly: just know that your private key is private!).

Such things shouldn't necessarily be put onto a public Git repository being hosted somewhere (unless you actually intended for the entire internet to be able to use your key).

In this exercise you'll be writing a Git **pre-commit hook** that stops the commit from happening when an API key is directly in the staged code files or if a private SSH key file is staged. If you are unfamiliar with Git hooks, check out ADV3's "Automating Professionalism" exercise and read through (and maybe play around with) its introduction. If your hook fails the commit, it should print out what files are the culprits: if there are multiple private keys or multiple API key assignments, each instance should be reported.

Here are some things to identify each component with (these are simplified for this exercise).

API keys:

- They are a string literal enclosed by double quotes (`"`) or single quotes (`'`)
- They are a string composed of alphanumeric characters of both cases (A-Z, a-z, 0-9)
- They are being assigned via '=' to some variable whose name ends in "`key`" or "`Key`". There can be an arbitrary amount of whitespace around the '='

SSH private keys:

- The file's first line is "`-----BEGIN OPENSSH PRIVATE KEY-----`"
- The file's last line is "`-----END OPENSSH PRIVATE KEY-----`"

Submission checkoff:

☐ Show your Git hook working.

☐ Show your Git hook's code. Be prepared to explain parts of it.

Helpful hints:

- You can get a list of staged files via `git diff -name-only -cached`.
- `head` and `tail` can get the first and last lines of a file.
- The `adv5-files` directory has an example private key and API key assignments.

2 More `sed` fun (10)

Pig Latin is language game in which English words are made to sound like a faux-Latin by moving around groupings of letters.
Here are the basic rules for this exercise's variant:

1. For words that begin with a single consonant before running into a vowel, the consonant is moved to the end and "ay" is added after.
   - "hello" = "ellohay"
   - "cat" = "atcay"
   - "liquid" = "iquidlay"
2. For words that begin with a multiple consonants before running into a vowel, the consonant group is moved to the end and "ay" is added after.
   - "string" = "ingstray"
   - "friend" = "iendfray"
   - "wrong" = "ongwray"
3. For words that begin with a vowel, "yay" is simply added at the end.
   - "I" = "Iyay"
   - "apply" = "applyyay"
   - "income" = "incomeyay"

For this exercise, write a script that takes its input and turns it into Pig Latin and outputs the translation. The big caveats are:

- Each word on a line will be translated.
- Words with a punctuation mark adjacent will be translated and retain the punctuation mark. For example: "goodbye!" = "oodbyegay!"
- Capitalization of the word is preserved. For example: "My name is John" = "Ymay amenay isyay Ohnjay".
- Hyphenated words consider each component to be a separate word. For example: "part-time" = "artpay-imetay".
- Spacing will be preserved. For example: "hello world" = "ellohay orldway"

The `adv5-files` directory has example text and their translated versions for reference.

Submission checkoff:

☐ Show that your script works.

☐ Show your script's code. Be prepared to explain parts of it.

Helpful hints:

- There are other anchors beside '^' and '$': the GNU `grep` manual documents some.
- GNU `sed` has extensions that deal with upper/lower case conversion. Its manual's section on the `s` command is enlightening.

3 Option parsing (5)

`getopts` is a Bash-builtin that allows for better management of optional parameters for a script. It is able to handle single character options. It *can* be a bit obtuse, so feel free to look at examples of its usage on the internet. Don't forget it also has a `man`page. Write a script that:

- Uses `getopts` to parse options.
- By default it will print out "Hello world!" once.
- `-s <string>` specifies string to print.
- `-n <num>` specifies how many times to print.
- `-r` causes the printed out string to be reversed.

Submission checkoff:

- ☐ Show that your script works.
- ☐ Show your script's code. Be prepared to explain parts of it.