Advanced - Regex

EECS 201 Winter 2024

Submission Instructions

This assignment will be submitted as a repository on the EECS GitLab server (see Basic - Git 1). Create a private, blank, **README-less (uncheck that box!)** Project on it with the name/path/URL `eecs201-adv-regex` and add `brng` as a Reporter. The submission branch will be `latin`. If this branch is not already the default initial branch, you initialize the local repo with an additional argument: `git init --initial-branch=latin` or `git init -b latin` if your version of Git is recent enough. Otherwise you can create a branch with this name after your first commit.

To finish the submission process and get your grade, you will need to run the autograder. You will need to SSH into the course server (see Basic - Intro):
`local$ ssh uniqname@peritia.eecs.umich.edu`
(the `local$` referring to a shell prompt on your system)

and then run `eecs201-test`:
`peritia$ eecs201-test adv-regex`
(the `peritia$` referring to a shell prompt on the server)

The repository should have the following directory structure, starting from the repository's root:

```
/
|-- report.txt
|-- pig-latinfy.sh
```

Preface

Really important preface here! Please read! Some implementations of POSIX standard tools go above and beyond what the specification describes, adding additional functionality. For many of the tools that the GNU Project implements, there are additional features that aren't in the POSIX spec. This assignment takes advantage of some of these additional features in GNU `sed`.

Why is this important? Linux systems use GNU utilities while macOS systems use BSD utilities (BSD being a Unix which macOS is ultimately derived from). **For macOS users, the next few sentences will be critical. For Windows/WSL/Linux users, it's additional fun knowledge.** Have no fear if you're a macOS user: you can use the Homebrew package manager to acquire the GNU versions of utilities (technically you can get the source code of the GNU utilities yourself, but Homebrew is pretty neat and if you're into that sort of thing, I'm confident you can figure out how to manually build them ;)! Chances are you already have installed Homebrew to install some other software on your Mac, but if you haven't you can find the instructions to install it here: `https://brew.sh/` (when the installation completes, be sure to follow the followup steps on the terminal for adding the Homebrew managed directories to your `PATH`!). The `gnu-sed` Homebrew package provides the GNU `sed` that will be extremely helpful for this assignment. This package will provide GNU `sed` as the `gsed` command. Now this gives us a slight issue: the reference system for this class is Linux, which simply has the `sed` command running GNU `sed` and no such `gsed` command. In this assignment you'll be writing a shell script: what you can do is add a bit of system detection at the beginning:

```
# for Mac users using gnu-sed from Homebrew
if [ $(uname) = "Darwin" ]; then
  shopt -s expand_aliases # needed if the shebang uses Bash
  alias sed=gsed
fi
```

This uses `uname` to get the OS of the system the script is running on: Linux systems will output "Linux" while macOS systems will output "Darwin". This will "alias" `sed` to `gsed`, allowing you to transparently refer to `gsed` as `sed` in our script. The `shopt -s expand_aliases` will allow aliases to work from within a script (non-interactive) environment if you're setting your script to use Bash.

If you don't want to use Homebrew or install GNU `sed` on your Mac, feel free to do the assignment on the course server, which serves as the reference environment.

In this assignment you'll be provided yet another zipped archive containing some sample input data as well as the expected data. Use your preferred tool to retrieve this file and extract it (see Basic 2 if you need a review).

`https://www.eecs.umich.edu/courses/eecs201/wn2024/files/assignments/adv-regex.tar.gz`

Initialize a Git repository in the extracted `adv-regex` directory, following the submission instructions.

1 More `sed` fun (10)

Pig Latin is a 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"
4. For this Pig Latin variant, the letter 'y' is always a consonant. For your convenience, consonants are
   `BCDFGHJKLMNPQRSTVWXYZ`
   and vowels are
   `AEIOU`
5. If the word contains no vowels, then leave the word alone.

For this exercise, write an **executable shell/Bash** script called `pig-latinfy.sh` that takes **its input on standard input** and turns it into Pig Latin and outputs the translation **on standard output**. **Be sure to set the shebang appropriately!** If you are on WSL and doing this on the Windows filesystem, be sure to have Git set the execute bits (look up how to do this)! 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" = "My 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"
- You don't have to worry about contractions (e.g. words shortened with an apostrophe). You only have to worry about words that are sequences of just Latin alphabetic characters.

The `adv-regex` directory has example text and their translated versions for reference.

Helpful hints

- There are other positional matches beside '`^`' and '`$`': the GNU `grep` manual documents some.
- If you find yourself making loops and if-statements, you might be overcomplicating things. My current solution consists entirely of four variable assignments to pre-defined strings and a `sed` invocation running multiple `s` subcommands. My old solution consists entirely of four variable assignments to pre-defined strings and a pipeline of `sed` invocations.
- `sed` is already capable of reading from standard input without your intervention: you don't need to do anything special to read input. Simply running `sed` without an input file will cause it to read from standard input.
- GNU `sed` has extensions that deal with upper/lower case conversion. Its manual's section on the `s` command is enlightening.
- `diff` is able to show differences in files. Combine it with a process substitution (see the Shell Lecture) and you have an easy way to test your script...
- Remember that the `g` flag for the `s` command lets you find and replace multiple matches on the same line!
- Remember that this script reads from standard input: you can redirect input to read from a file in your invocation, but if you don't, you can directly type in strings to process when you run the script! Ctrl-D will signal an end of file (EOF) when you're done providing strings.
- You can use `diff` to take advantage of the expect outputs inside of the provided tarball. Try using a process substitution ;)

2 Conclusion

1. Add and commit any changes you intend to submit.
2. Create a file called `report.txt`.
3. On the first line provide an integer time in minutes of how long it took for you to complete this assignment.
4. On the second line and beyond, write down what you learned while doing this assignment. If you already knew how to do all of this, put down "N/A".
5. Add and commit this `report.txt` file.