Basic - Unix
Unix and the Shell

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-basic-unix` and add `brng` as a Reporter. The submission branch will be `main`. If this branch is not already the default initial branch, you initialize the local repo with an additional argument: `git init --initial-branch=main` or `git init -b main` 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 basic-unix
```

(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
|-- 1-redirection
  |-- p1.sh
  |-- p2.sh
  |-- p3.sh
|-- 2-pipeline
  |-- p1.sh
  |-- p2.sh
|-- 3-files
  |-- p1.sh
  |-- p2.sh
```

Preface

This assignment should be fine on Linux and macOS. That being said, if you run into issues on macOS, like a command completely not running as expected, try using the course server or an Ubuntu 22.04 VM.

In this assignment you'll be provided yet another zipped archive containing some starter empty files and scripts. Use your preferred tool to retrieve this file and extract it (see Basic - Git 1 if you need a review).

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

Initialize a Git repository *inside of* the extracted `basic-unix` directory as per the submission instructions.

1 Redirection

Recall from class that we can *redirect* the inputs and outputs of a command. `echo` is a command that will print out a string (taking into account any expansions you embed in it). For example, I could use `echo "Welcome $HOME"` to print out `Welcome /home/brandon`. Try modifying this command so that it saves that output to a file.

`cd` into the `1-redirection` directory. Inside you will see three shell script files: `p1.sh`, `p2.sh`, and `p3.sh`.

Open the files in your preferred text editor and follow their instructions. Remember that shell scripts are literally shell commands you type at a the command line, just saved inside of a file. You can test and evaluate things at the terminal itself! To run files, as an example you can run `$ ./p1.sh` to run `p1.sh`.

2 Pipeline

Recall from class that commands can be chained together to form a pipeline, where one process passes its output to the input of the next. For example, I could create this pipeline `cat file1 file2 file3 | rev` to concatenate three files and output the result with `cat` and then reverse the output of `cat` with `rev`.

`cd` into the `2-pipeline` directory. Inside you will see two shell script files: `p1.sh` and `p2.sh`, as well as `baby_names` and `grades`, which serve as input data for the two scripts, respectively. Likewise, open the files in your preferred text editor and follow their instructions. Be sure to check out the `baby_names` and `grades` files to see their structure as you figure out your solutions.

3 Files

Recall from class that files have metadata that correspond to what permissions various users have for interacting with them: read, write, execute. While in the videos I didn't want to throw another wrench into the number system discussion, these bits are grouped together to form a 3-bit octal (base 8) digit (which decimal, base 10, can represent regardless), in the order of `rwx` (read, write, execute). If you are unfamiliar with binary (base 2) representation, it's like decimal except each place is a power of 2, and the only option for each place is 0 or 1. For example, `110` is

$$1 * 2^2 + 1 * 2^1 + 0 * 2^0 = 4 + 2 + 0 = 6$$

.
If we look at what that means for permission bits, `110` maps to `rwx` (read, write, execute), where `1` means permission and `0` means no permission, so that means that an octal `6` means "readable and writable".
These files also have metadata that track ownership: what user owns the file and what group owns the file. Together, a file has three sets of these permission bits, one set of three corresponding to permissions for the user owner, one set for the group owner, and one set for everyone else not the user and not in the group ("other"). Thus, we can have an octal triplet such as `740`, with `7` (`rwx`) for the user who owns it, `4` (`r--`) for the group that owns it, and `0` (`---`) for others.

`cd` into the `3-files` directory. Inside you will see two shell script files: `p1.sh` and `p2.sh`. Likewise, open the files in your preferred text editor and follow their instructions. All you need to do for these two files is to provide the octal triplet on the right hand side of the assignment of the `mode` variable. If you want to test out your triplet, you can either create a file (e.g. via `touch`) and directly use `chmod` with the octal triplet on that file e.g. `$ chmod 777 file-path-here`, or create a file and then run the script with the created file as an argument e.g. `$ ./p1.sh file-path-here`.

4 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.
6. Run the autograder!