CS Homework 0

J. Holly DeBlois

February 5, 2025

General Idea

This "assignment" covers a handful of basic concepts/practices/commands that are used at a Linux command-line prompt and in C programming. Of course, there are tons of online/print resources available for learning/reviewing these things. Use any that suits you. For example, here are some such resources provided by the CS department:

- https://www.cs.umb.edu/~ghoffman/linux/linux_help.html
- https://www.cs.umb.edu/~ghoffman/linux/nano_text_editor.html
- https://www.cs.umb.edu/~ghoffman/linux/common_unix_commands.html
- https://www.cs.umb.edu/~ghoffman/linux/unix_essentials.html
- https://www.cs.umb.edu/~ghoffman/linux/remote_access_windows.html
- https://www.cs.umb.edu/~ghoffman/linux/remote_access_mac.html
- https://www.cs.umb.edu/~ghoffman/linux/unix_cs_students.html

Simple commands: `pwd`, etc.

Many commands print out basic information. Run:

- `whoami` to print your username
- `pwd` to print the working directory
- `hostname` to print the name of the host you're currently on

```
>>> whoami
bob
>>> pwd
/home/bob
>>> hostname
someserver
```

- `>>>` represents a prompt at which commands are entered
- a user's prompt often contains helpful information and can be customized by editing, e.g., the `~/.bashrc` file in the user's home directory
- info on command `foo` can be viewed on its "man" page by typing `man foo`
- get a little "meta" by typing `man man`
- man pages are also easily viewable on the web, e.g., https://linux.die.net/man/1/whoami

Homework 0 output

Entering the commands in this "assignment" will result in a single output file being built line-by-line. At the end, we will compare it to the instructor-provided example output with a `diff` command. Let's start. Run:

- `cd` with no arguments to change to your home directory
- `cd csNNN` to change to your course directory
- `mkdir hw0` to create a directory for the this assignment
- `cd hw0` to change into it
- `touch output.txt` to create the output file we will build
- `ls -la` to view things, which should look something like:

```
>>> ls -la
drwx------ 2 bob bob 4096 Sep  2 19:50 .
drwx------ 3 bob bob 4096 Sep  2 19:50 ..
-rw------- 1 bob bob    0 Sep  2 19:50 output.txt
```

- note that the file is initially empty, i.e., 0 bytes (`man touch` for more)
- review the meanings of the information printed from the long form of `ls`, e.g., here:

  https://www.gnu.org/software/coreutils/manual/html_node/What-information-is-listed.html

Building, Step 1: output redirection

We will use output redirection to form the first few lines of our file. Review file handles, redirection, pipes, and related concepts, e.g., here: https://en.wikipedia.org/wiki/Redirection_(computing). Now:

- print a message to stdout with `echo "Hello World!"`
- redirect the message to our file with `echo "Hello World!" > output.txt`
- effectively do both with `echo "Hello again!" | tee -a output.txt`
- append to our file with `echo "my username is:" >> output.txt`
- and again with `whoami >> output.txt` ... the output should be looking like this:

```
>>> echo "Hello World!"
Hello World!
>>> echo "Hello World!" > output.txt
>>> echo "Hello again!" | tee -a output.txt
Hello again!
>>> echo "my username is:" >> output.txt
>>> whoami >> output.txt
```

- to start over at any point, use the one-liner `rm -f output.txt && touch output.txt`
- check contents with `cat output.txt` and number of lines with `wc -l output.txt`
- review newline subtleties, e.g., here: https://en.wikipedia.org/wiki/Newline

Building, Step 2: text editor

We can run a text editor from the command line interface (CLI) to add to our file. We will illustrate with `nano`, but feel free to substitute `vim`, `emacs`, etc. Now:

- examine your PATH variable with `echo $PATH`
- determine the location/existence of the executable with `which nano`
- is the `nano` executable's location contained in the PATH?
- type executable name's first two characters: `na` and tap `TAB` twice — what happens?
- type `nan` and hit `TAB` — what happens?
- type `nano ou` and hit `TAB` — what happens?
- run `nano output.txt` to add a single (unique, non-empty) line of your choosing at the end of the file; save, exit
- edit again, this time running `nano` by its full path `/usr/bin/nano output.txt`; append another (unique, non-empty) line of your choosing, save, exit
- the version of a program is often easy to get at the CLI — type `nano -V`
- get help with `nano -h` or `nano --help` or `man nano`
- check the number of lines in our file with `wc -l output.txt` — you should have 6
- fyi, our file's final version will have 11 lines

Building, Step 3: (dis)assembly, another login

Run:

- `head -n 3 output.txt > part_A.txt` to copy the first three lines
- `tail -n 3 output.txt > part_C.txt` to copy the last three lines
- `tail -n 4 output.txt | head -n 1 > part_B.txt` to copy just the third line
- `rm output.txt` to delete the original output file
- `cat part_*.txt > new.txt` to assemble the parts
- `uniq new.txt > output.txt` to recreate the output file
- `rm -f part_*.txt && rm -f new.txt` to clean up

Now — without logging out of your original session — SSH into the server from another tab/window/machine and run `top` to monitor system processes. Then, in the original session, run the following replacing `bob` with your username:

- `ps -u bob` (sample output is shown below)
- `ps -u bob | grep top >> output.txt` to capture your `top` command's process id, etc.

```
  PID TTY          TIME CMD
...
11130 ?        00:00:00 sshd
11132 pts/0    00:00:00 bash
13554 ?        00:00:00 sshd
13555 pts/1    00:00:00 bash
17854 pts/0    00:00:00 top
17855 pts/1    00:00:00 ps
```

Reviewing C Concepts

Before we compile a small C program, let's review relevant C concepts and what distinguishes C from other languages — we'll contrast with C++ and Python

- compilation
  - what does it mean for computer code to be compiled? interpreted?
  - identify the language-choice tradeoffs with regards to:
    - code-writing speed
    - execution speed
    - ease of installation on a system
    - open/closed source, intellectual property, proprietary software, etc.
  - review the basics of gcc, GNU Make, and related tools
- memory management
  - compare/contrast how memory is handled in C/C++/Python
  - later in this "assignment," we will work with symbolic links in a filesystem... in what way is a symbolic link similar to a pointer in C code?
  - find a good Youtube video (or similar) and review C data types, pointers, arrays, `malloc`, etc.
  - identify references (text, e.g., Kernighan & Ritchie, or online, e.g., cplusplus.com) to use when writing C/C++ code
- object-oriented programming (OOP)
  - think of a few software design cases that are perfectly suited to OOP
  - find a good Youtube video (or similar) and review OOP concepts, e.g., classes, members, methods, encapsulation, inheritance, etc.
  - what C data structures are most similar to C++ classes?
  - in what cases might you want to do some OOP in Python and then port it to C++? how difficult would the porting process be?

Building, Step 4: compiling a substitute `wc`

The `wc` ("word count") system command counts the words (or lines, or characters) in the file it is passed as an argument (see `man wc`). We will compile a very small C program that amounts to a *crude* version of this program. Note that **our version will read from standard input**. Now run:

- `q` and then `exit` to close the `top` command/window from before (if still open)
- `wget https://www.cs.umb.edu/~hdeblois/hw0/crude_wc.c` to download the source code file to current directory (`hw0`)
- `less crude_wc.c` to view its contents and `q` to exit the `less` command
- `gcc -o crude_wc crude_wc.c` to compile
- `ls -la` to view directory contents — should look something like this:

```
total 36
drwx------ 2 bob bob  4096 Sep  3 02:43 .
drwx------ 3 bob bob  4096 Sep  3 02:46 ..
-rwx------ 1 bob bob 16664 Sep  3 02:43 crude_wc
-rw------- 1 bob bob   635 Sep  3 02:43 crude_wc.c
-rw------- 1 bob bob    57 Sep  3 01:48 output.txt
```

- run `wc output.txt` to perform counts with the system `wc`
- run `./crude_wc < output.txt` to perform counts with our crude substitute and compare
- run `./crude_wc < output.txt >> output.txt` three times in a row
- `cat output.txt` and make sense of what you see
- run `ldd crude_wc` to view the shared library dependencies of the executable

Reviewing symbolic links

Symbolic links ("symlinks") require some getting used to, but can be very handy for, e.g.:

- "marking" one version of some program/library as the one currently being used
- making an executable available system-wide by putting a link to it in, e.g., `/usr/bin/`
- "regrouping" things, e.g., in the way that each student in a class CS NNN has a link `csNNN` in his/her home directory to the corresponding course directory in `/courses`
- doing some ordering, e.g., of e-books, in the order you'd like to read them:

```
lrwxrwxrwx 1 bob bob      11 Jan 27 16:35 1 -> macbeth.pdf
lrwxrwxrwx 1 bob bob      17 Jan 27 16:35 2 -> anna_karenina.pdf
lrwxrwxrwx 1 bob bob      23 Jan 27 16:35 3 -> wind_in_the_willows.pdf
-rw------- 1 bob bob 6890957 Jan 27 16:38 anna_karenina.pdf
-rw------- 1 bob bob 2956256 Jan 27 16:39 macbeth.pdf
-rw------- 1 bob bob  391313 Jan 27 16:38 wind_in_the_willows.pdf
```

When creating/modifying/using symlinks, keep in mind:

- either absolute or relative paths can be involved — review these as necessary
- the `ln` command creates a hardlink by default (use `ln -s ...` for symlinks)
- like pointers in C, symlinks can exist in a broken state — this is the price we pay for avoiding the duplication of data in making a copy
- programs that interact with the filesystem often provide CLI options to specify the handling of symlinks (e.g., run `man chown` and see the section on the `-h` option)

Building, Step 5: symbolic links

Now run:

- `mkdir somedir` to create a subdirectory and change into it with `cd somedir`
- `ln -s ../crude_wc crude_wc_ptr` to create a symlink to our new executable
- `ln -s ../output.txt` to create a symlink to our output file (note lack of 2nd argument)
- `ls -la` should show something like:

```
total 8
drwx------ 2 bob bob 4096 Sep  3 03:50 .
drwx------ 3 bob bob 4096 Sep  3 03:38 ..
lrwxrwxrwx 1 bob bob   11 Sep  3 03:39 crude_wc_ptr -> ../crude_wc
lrwxrwxrwx 1 bob bob   13 Sep  3 03:50 output.txt -> ../output.txt
```

- run `echo "this is a test" | ./crude_wc_ptr` to test running the link-to-executable
- run `./crude_wc_ptr < output.txt` to run link-to-executable on link-to-output-file
- run `chmod 770 crude_wc_ptr` and use `ls -la` on both current and parent directories to discern which permissions have changed (note that this behavior may be OS-dependent)
- run `readlink -f crude_wc_ptr >> output.txt` to dereference the link-to-executable and append its full path to our output file

Almost Done!

Finally:

- remove the two symlinks with `rm crude_wc_ptr` and `rm output.txt`
- change to parent directory with `cd ..`
- remove empty directory with `rmdir somedir`
- verify that output file has 11 lines with `wc -l output.txt`
- download instructor's output file with
  `wget https://www.cs.umb.edu/~hdeblois/hw0/sample_output.txt`
- run `diff -y output.txt sample_output.txt` to perform a side-by-side comparison of your output file with the instructor's; make sense of any differences