Lab: Command Line Basics

CSci 430: Introduction to Operating Systems

Summer 2021

Questions

- What is a command shell and why would I use one?
- How can I move around on my computer on the command line?
- How can I see what files and directories I have?
- How can I specify the location of a file or directory on my computer?
- How can I create, copy, and delete files and directories?
- How can I combine existing commands to do new things?
- How can I find files?
- How can I find things in files?

Objectives

- Explain how the shell relates to the keyboard, the screen, the operating system and users' programs.
- Explan when and why command-line interfaces should be used instead of graphical interfaces.
- Explain the similarities and differences between a file and a directory.
- Translate an absolute path into a relative path and vice versa.
- Construct absolute and relative paths that identify specific files and directories.
- Use options and arguments to change the behaviour of a shell command
- Create directories and files using command line tools.
- Delete, copy and move specified files and/or directories.
- Redirect a command's output to a file.
- Process a file instead of keyboard input using redirection.
- Construct command pipelines with two or more stages.
- Explain Unix's 'small pieces, loosely joined' philosophy.
- Use `grep` to select lines from text files that match simple patterns.
- Use `find` to find files and directories whose names match simple patterns.

Suggested Additional Activities

It is suggested that you use one or both of the following suggested resources before or while performing this lab. These additional tutorials or lecture videos will usually go into more detail than what we discuss in our labs for this class, but the investment will usually be well worth your time to better understand the lab topic.

- Software Carpentry Lesson on The Unix Shell
  - This is an excellent tutorial in a web/written format, that also includes exercises and their answers along with the tutorials. The typical time is about 4 hours and 30 minutes to go through the whole tutorial and do the exercises. For our class, parts 1-4 and 7 are most relevant. Parts 5 and 6 can be skipped for this class, as these get into writing shell scripts and scripting actions
- MIT Missing Semester of CS Education Playlist
  - The entire set of course lecture videos is excellent, and has links to exercises. For this lab, the first 2 lectures on Lecture1 : The Shell and Lecture 2: Shell Tools and Scripting are both 45 minutes and very good to watch.

Overview and Setup

All of the labs can be performed from your DevBox, or any other Unix based distribution command line. We will assume the DevBox Ubuntu Debian Linux distribution, and the particular version of the tools and options available on that distribution. But in short you can and should use a terminal from in your DevBox to work on the tutorials. If you have not done so already, you can open a terminal window in your DevBox VSCode server by going to `Menu->Terminal->New Terminal`, or use the keyboard shortcut `ctrl-~` (thats a `ctrl-shift` backtick).

NOTE: If you do a lot of command line terminal usage form the VSCode terminal, it can be useful to change some settings to ensure keybinds go to the shell instead of VSCode when working in a terminal. Search for `Terminal -> Integrated: Allow Chords` and disable this setting, and also for `Terminal -> Integrated: Send Keybindings To Shell` and enable this setting. Especially enabling sending keybindings to shell really helps to use standard bash key bindings to edit shell commands you are working on.

Lab Tasks

- In a terminal, change into the `lab01-1` directory. The `lab01-1` directory is in your `sync` folder. How you navigate to it depends on your current location. For example, if you open up a terminal, you will usually be in your home directory. You can check your current working directory with the `pwd` command.

  ```
  vagrant@devbox:~$ pwd
  /home/vagrant
  ```

  You can specify an absolute or a relative path name to change into this labs subdirectory. For example, if you are in your home, a relative path name that works is

  ```
  vagrant@devbox:~$ cd sync/lab/lab01-1/
  vagrant@devbox:~/sync/lab/lab01-1$
  ```

  Or you could specify a full absolute path

  ```
  vagrant@devbox:~$ cd /home/vagrant/sync/lab/lab01-1/
  vagrant@devbox:~/sync/lab/lab01-1$
  ```

  Learn the difference between absolute and relative path names, and also practice using the `cd`, `ls` and `pwd` commands to move around and look at your filesystem from the command line.

- Use the `mkdir` command to create a new subdirectory named `mylabwork`, and change into this directory.

- We will usually have you create a timestamp of when you begin and end a lab. Using redirection and the date command, record the time when you began working on the lab, like this

  ```
  vagrant@devbox:~/sync/lab/lab01-1/mylabwork$ date > start-timestamp.txt
  vagrant@devbox:~/sync/lab/lab01-1/mylabwork$ cat start-timestamp.txt
  Sun Jul 18 20:15:04 CDT 2021
  ```

  The `cat` command is short for conCATenate a file. One simple use of cat is to display the contents of a file on your terminal, like this: You can do this to check the timestamp of the file you just created.

- The `ls` command can be used to list the files in your current working directory. Use it to list out the files in your current `mylabwork` directory. Now use it to list out the files in your sync directory, without changing directories. You can use a relative or absolute path name to do this. For example, using an absolute path name

  ```
  vagrant@devbox:~/sync/lab/lab01-1/mylabwork$ ls /home/vagrant/sync/
  README.md  Vagrantfile  assg  config  docs  example  include  lab  libs  log  scripts
  ```

  Or notice that the sync subdirectory is up 3 levels from your current working directory. The `..` in a path refers to the parent directory of your current directory. So we could list the `sync` directory using a relative path name like this

  ```
  vagrant@devbox:~/sync/lab/lab01-1/mylabwork$ ls ../../..
  README.md  Vagrantfile  assg  config  docs  example  include  lab  libs  log  scripts
  ```

- Use redirection to save the list of files in your `sync` directory into a file named `my-sync-directory.txt`. You can redirect the output of any file from your terminal into a file instead using the output redirection symbol `>`, just like you did to redirect the output of the `date` command into a file as one of the first tasks.
- You can usually get more information about a command by asking the comand for help

  ```
  vagrant@devbox:~/sync/lab/lab01-1/mylabwork$ ls --help
  Usage: ls [OPTION]... [FILE]...
  List information about the FILEs (the current directory by default).
  Sort entries alphabetically if none of -cftuvSUX nor --sort is specified.
  ...
  ```

  We didn't show all of the help you get about `ls` here, just the first few lines. But for example, you will get a list of all of the command line arguments the command can accept. So, if you want to get a long listing with more information about your files, use the `-l` flag.

  ```
  vagrant@devbox:~/sync/lab/lab01-1/mylabwork$ ls -l
  total 8
  -rw-r--r-- 1 vagrant vagrant 42 Jul 18 20:27 my-sync-directory.txt
  -rw-r--r-- 1 vagrant vagrant 29 Jul 18 20:15 start-timestamp.txt
  ```

  Use the `-l` flage and output redirection to save the long listing of files to a file named `my-lab-long-listing.txt`. Question, did the name of this new file end up in the listing you captured as output? Did you expect that? Hint: remember you can use `cat` to display the contents of a text file. Practice using other command line flags with the `ls` command that you see from its help.
- Notice that the `--help` command line argument used two `--` before the command, while the `-l` argument used a single `-`. Most commands take what are called long and short command line argument flags. Sometimes there is both a short and long version of a command line argument, though sometimes there is only one or the other. For example long listings only have the `-l` short command line argument. But the `-h` short argument, to display file sizes using human readable units, has a corresponding long argument `--human-readable`. Try using both with the `-l` flag on your home directory:

  ```
  vagrant@devbox:~/sync/lab/lab01-1/mylabwork$ ls -l -h /home/vagrant
  total 24M
  -rw-r--r-- 1 vagrant vagrant  24M Jul 18 18:48 cpptools-linux.vsix
  drwxrwxr-x 1 vagrant vagrant 4.0K Jul 18 18:52 sync

  vagrant@devbox:~/sync/lab/lab01-1/mylabwork$ ls -l --human-readable /home/vagrant
  total 24M
  -rw-r--r-- 1 vagrant vagrant  24M Jul 18 18:48 cpptools-linux.vsix
  drwxrwxr-x 1 vagrant vagrant 4.0K Jul 18 18:52 sync
  ```

  Notice the result is the same. In both cases human readable file sizes are given, the `cpptools-linux.vsix` file is 24 megabytes in size, and the `sync` directory is 4 kilobytes in size on the file system. You can usually combine short command line arguments together for most commands, like this

  ```
  vagrant@devbox:~/sync/lab/lab01-1/mylabwork$ ls -lh /home/vagrant
  total 24M
  -rw-r--r-- 1 vagrant vagrant  24M Jul 18 18:48 cpptools-linux.vsix
  drwxrwxr-x 1 vagrant vagrant 4.0K Jul 18 18:52 sync
  ```

  Notice you still need a `-` before them, but you can have 2 or more short command line flags specified in this manner.