S[&]T Unix/Linux Course Exercises

Ian Price, Leo Breebaart

August 11, 2022

1 Getting help

The most difficult part of the Linux command line can sometimes be figuring what command you need in the first place.

Once you have the name of a command that *might* do what you need, you have more options. Start by executing the command with the option `-h` or `--help`. If they are supported you will at least be given the set of possible options. If this is not sufficient, or you're not certain what the command is *intended* to do, then see if there is a *man page* by running the command `man command`. The man page will give you much more information. It will likely also have a "See Also" section near the end which you can use when the current trail is going cold. If you are on a modern Linux system, you can also find documentation files for any package in the `/usr/share/doc/` directory. 'Package' is not always the same as 'command', and sometimes the documentation is just a copyright file – but it's definitely a place to look for more info.

And of course, there is also always Google.

Try the following commands to see how these systems work.

```
man -k compress
man -h
man man
cd /usr/share/doc/manpages
```

2 File tools

2.1 File attributes

I'm trying to clean up a directory from an old project but I get a "permission denied" message when trying to delete a file ...

Every filesystem item has an owner, a group, read–write–execute permissions of the owner, group and others, as well as timestamps for creation, last accessed and last modified. The `ls` command can show you most of this information. The permissions are shown as a mildly cryptic 10 character string (i.e. `-rwxr--r--`). The user and group are shown by name or number (user id (aka uid) group id (aka gid)). In order to see if you have the required permission to change or delete a file you need to understand the gory details.

When authorising an action, the permissions are considered in the order: user, then group, then other. If you are the owner then the user permissions apply. If you're not the owner but are a member of the group then the group permissions apply. In all other cases the other permissions apply. The actual permissions are indicated by a 3 characters block in the permissions string (starting at 0, user 1–3, group 4–6, other 7–9). The characters r, w, x indicate read, write and execute permission respectively. Directories must have execute permission in order to see the filetable (the content within) and removing a file requires a change to the filetable. Ok, so now you can see why you couldn't remove the files, but what now? The `chmod` command allows the permissions to be changed, but you need to be the owner (or superuser) in order to change them.

The commands `chown` and `chgrp` allow the owner and group to be changed. In general the original problem can only be resolved by the superuser if they involve files owned by someone else. However, they are still useful for protecting yourself against accidental deletion.

Exercise 8: Use the commands `touch`, `mkdir`, `chmod` to make a new directory that contains the files *readonly.txt* and *readwrite.txt* with appropriate permissions.

2.2 Finding your way around

All filesystems in UNIX are tree structures and you need to be able to navigate these structures. So where are you now, what else is there and how do you get to where you want to be?

Every process has a *current working directory* and the shell is no exception. The `pwd` command will echo the absolute path of this directory to stdout. To see what else is located in this directory use the `ls` command. Run the command `ls -laF` now. There are many columns with useful detail but for the moment focus on the last column. It shows the directory content, including two special directories (`./` and `../`). These items and members of *every* directory, referring to the directory itself and the parent directory respectively. They are essential components of *relative path* specifications. The current working directory can be changed with the `cd` command, based on the argument you supply it. The new location can be specified in an absolute or relative form.

Exercise 1: Find out the owner and the permissions of the `ls` command.

Exercise 2: Try and do likewise for the `cd` command. This may prove to be difficult. Figure out why!

2.3 Finding files

Two years ago you made a presentation for some project but you don't remember exactly what the file was called, nor the project name. How do you find it without `cd`ing to every directory in your home area?

If your directory structure is not too deep, the command `tree` may help, which presents a (much) nicer version of `ls -R`.

The commands `find` and `locate` can also help. The later uses a database (regularly and automatically updated by the system) to provide very fast searches of the entire filesystem. The former does a recursive search which is slower than `locate` but a lot faster than manually searching.

```
locate "*.py" | less
find . -name "*.py" | less
```

Take care to prevent the shell from expanding any wildcards(*) by using quotes. Also be sure you understand *what* the wildcard will be tested against.

Exercise 3: Find all the 'pictures' in the *exercise-3* subdirectory that are owned by user '1054' and have the extension '.jpg'.

So you've found a bunch of files, but often you will want to run a command on each of them, for example fixing spelling mistake, or changing the permission of each file.

The `find` command has a built-in mechanism for doing this using the `-exec` option. It is rather cumbersome and limited to use however, so piping to the `xargs` command, while more complex to master, is far more powerful. Use what works for you.

Exercise 4: Check that all the compressed tar files (*.tgz) in your home area are actually compressed tar files. Hint: maybe there is a command that can identify what a file type is? Perhaps 'man -k "file type"' could help?

2.3.1 Counting lines of code

Problem: You need a ballpark figure for the number of lines of code in your project. The files (let's assume they are Python source files, so can be described

by a "`.py`" filename extension) are located in a big nested directory hierarchy, and some directories (e.g. those containing tests or generated code files) should *not* be counted.

Solution:

```
find ./exercises/exercise-5 -name "*.py" | sed '/test/d' | xargs wc
```

`find` collects the files, `sed` filters out the lines containing the "test" string (how does that work?), `xargs` collects the filenames and feeds them as arguments to `wc`, which outputs the line-, character-, and byte-count of the files.

Exercise 5: The above solution will unfortunately also filter out e.g. the file named *the-greatest.py*. Try to make things more robust, and ensure that only directories that are named "test" are excluded from the count – but nothing else.

Intermezzo 1: Come up with a one-liner command that will create the (empty) directory *./unix-course/exercise-5/deep/deeper/deepest/*

Intermezzo 2: In the directory you just made, create a file named *this filename has spaces in it.py*.

Exercise 6: The solution given at the beginning of this section (and probably the one you came up with for Exercise 5) will now no longer work. Why not? Try to figure out a new solution (using the same basic command chain) that will solve that issue.

Exercise 7: Increase the accuracy of the result by excluding both empty lines and comments (in Python, that's any line starting with a hash '#') from the count. (There's no need to stick to the given set of commands for this one. Go wild!)

Epilogue: It is 2035, and a new version of Python has been released which changes the default extension for all source code files from *.py* to *.snake*. You'd like to change the extension of each and every file in the *exercise-5* directory accordingly. Using command-line tools, this is perhaps not the easiest thing to do, but once you've got it working, the same pattern will be applicable for many other usecases. Hint: you will probably need the `basename` command for this one.

3 Text tools

I have a text file containing a header section followed by columns of numbers and I want to load this into a spreadsheet program such as OpenOffice or Excel to do some simple analysis. The problem is that OpenOffice chokes on the header and misaligns the columns. What now?

With a bit of preprocessing you can reformat the data so OpenOffice/Excel will get it right. We can use a chain of commands, each solving one specific

problem. The following text is the first 8 lines of the file provided as *TemperatureSensor.txt* in the exercises directory.

```
# Temperature Sensors no. 12234, 12239
# Last calibrated : 2008-04-01
# Columns are
#     Time (s)
#     inlet fluid temperatures (C)
#     outlet fluid temperature (C)
0.0, 132.60    168.7
5.1, 87.56   93.779
```

Exercise 9: Use `grep` to generate a new file that has the header lines removed (those that start with #).

Exercise 10: How could you achieve the same result with `wc` and `tail`, or `sed`?

With the headers removed, the resulting columns need to be adjusted and use a consistent column separator.

Exercise 11: Tidy up the output from the previous exercise by separating the numeric fields on each line by a single space using the `sed` command.

The `sed` command is very powerful, but a little tricky to master. You need to escape (\) many control characters in regular expressions and trial and error is sometimes the order of the day (info sed).

In any event, don't be too concerned about chaining together a lot of commands that each do a part of the work. It is still highly likely this will be more efficient than manually fixing things in an editor.

I now have the data in columns but I need it ordered based on the value of the second column.

Exercise 12: Use the sort command to order the lines of the file produced in the previous exercise by their numeric value.

I have a written a bunch of HTML files as user documentation but I'm not sure if written everything I've referenced. How will I check this?

Extract all of the lines that have a `href` attribute, filter out the filename and eliminate the duplicates. Finally, compare this to the list of HTML files in the current directory.

Exercise 13: Try this on the html files in the *exercises* directory.

4 Processes and users

Interactive applications running on your desktop machine have become sluggish in response to input. It seems that some process is consuming excessive resources but how do you determine if this is really the case and which process is the culprit?

The `top` command will give you a summary of the system resource usage and the most resource intensive processes. Load averages near or greater than the total number of cores you have indicate the CPU is heavily loaded. It will also give you the *process id (PID)* and the owner of the processes listed. Top is a valuable utility, but be careful not to *over-interpret* the statistics it presents. And use `htop` if available – its output is much easier to read.

Having found a process that is consuming hefty resources is step one, but you still need to consider if this is a problem that needs resolving or if it is normal and you can wait for the process to complete. This is rather subjective but fist consider the owner of the process. It is unusual for root processes to be resource intensive, but it is also "out of the scope" for a normal user to stop and restart a root owned process (i.e. ask for help). For your processes you can try and stop them normally or stop them with a signal via the `kill` command. Kill have many modes of attack. Start gently as it gives the process itself the best chance of shutting down cleanly (i.e. an editor might just manage to save before stopping when given the TERM signal but it certainly will not if given signal number 9).

```
kill -TERM 1234
kill -QUIT 1234
kill -9 1234
```

The special filesystem `/proc` contains files that provide information about the state of the system. You can use these files as input to commands, and as such the `/proc` filesystem is the best source for script processing. Try and determine the amount of swap space that is being used based on the `/proc` filesystem.

If you can't resolve the situation by killing stuck processes you might need to resort to a reboot. Before doing this see if anyone is logged in to the system remotely or running a processes in the background (not logged in) so you can warn them of the pending reboot. You can get a list of logged in users with the `who` command.

Exercise 13: Try and get a list of all users that have a process running (start with the `ps` command).

Exercise 14: From the list you obtained in Exercise 13, figure out how to derive a list of the full names of the users in question, (i.e. 'Leo Breebaart' and 'Ian Price' instead of 'leo' or 'ian'). Create an executable shell script called *running-users.sh* that will output this list of full names.

Exercise 15: You would like this shell script to be executed and mailed to you every Tuesday and Friday at 6am. Sounds like a job for the *crontab* system. Use `man 5 crontab` (Why is the 5 in there? Why doesn't plain `man crontab` work?) to figure out how to install a cron job that does exactly this. Device and run a test to see if this indeed works (we don't just want to wait until Friday rolls by, so think of something else).

5 Networking

A logger daemon is supposed to be running on a server and accepting connections on port 3434, but messages sent from the normal client are failing to arrive in the log file. You've checked that the log server process is running.

Exercise 16: Work out how to use `telnet` to see if the server is listening for connections. On the server, use `netstat` to figure out what processes are listening.