WITS UNIVERSITY

School of Computer Science and Applied Mathematics
UNIVERSITY OF THE WITWATERSRAND, JOHANNESBURG

COMS4040ACOMS7045A: High Performance Computing & Scientific Data Management
Linux tutorial

2020-2-7

This tutorial presents a quick introduction [1] to the basic command line operations in Linux Operating Systems. Students who are not familiar with Unix or Linux OS should go over this tutorial.

1 Files

- The `ls` command gives you a list of files that are in your present location.
- **Exercise:** Try commands `ls`, `ls -l`, `ls -s` etc. What are the outcomes?
- You can read the man pages of Unix commands using `man commandname`. The `man` command puts you in a mode where you can view long text documents. This viewer is common on Unix systems, so memorize the following ways of navigating: Use the space bar to go forward, `u` key to go back up, `g` to go to the beginning of the text, and `G` for the end, `q` to exit the viewer. If none of these is working, `ctrl-c` will get you out.
- The `cat` command is often used to display files, but it can also be used to create simple content.
- **Exercise:** Type `cat > newfilename` (where you can pick any filename) and type some text. Conclude with `ctrl-d` on **a line by itself**. Now use `cat` to view the contents of that file: `cat newfilename`.
- The `touch` command creates an empty file, updates the time-stamp of a file if it already exists. Use `ls -l` to confirm this behavior.
- Three more useful commands for files are: `cp` for copying, `mv` for renaming, and `rm` for deleting. Experiment with them.
- For displaying parts of a file or information about a file, `head`, `tail`, `more`, `wc` commands are useful. Experiment these commands on an existing text file. `head` displays the first couple of lines of a file, `tail` the last, and `more` uses the same viewer that is used for man pages. The `wc` ('word count') command reports the number of lines, words, characters in a file.

2 Directory

- The root of the Unix directory tree is indicated with a slash '/'.
- The `pwd` command gives you your current working directory.
- The `mkdir` command creates a new directory.
- **Exercise:** make a new directory with `mkdir newdirectoryname`.
- The `cd` (change directory) command moves you around different directories.
  - `cd` without any arguments, takes you to your home directory.
  - The single dot is the current directory, and the double dot is the directory one level back.
  - Shortcuts for your home directory: `cd`, `cd ~`, `cd $HOME`.
  - `cd <path>`
  - If you use `cp` to copy a directory, you need to add a flag `-r` to indicate you recursively copy the contents.

3 Permissions

- Three types of actions can be taken with files or directories:
  - reading `r`: any access to a file that does not change it.
  - writing `w`: access to a file that changes its content.
  - executing `x`: if the file is executable, to run it; if it is a directory, to enter it.
- Three classes of people who potentially access a file or a directory
  - the user `u`: the person owning the file;
  - the group `g`: a group of users to which the owner belongs;
  - other `o`: everyone else.

  The permissions are rendered in sequence:

user	group	other
rwx	rwx	rwx

  For instance `rw-r--r--` means that the owner can read and write a file, the owner's group and everyone else can only read.

  Permissions are also rendered numerically in groups of 3 bits, by letting $r = 4$ (or 100 in binary), $w = 2\,(010)$, $x = 1\,(001)$.
- You can set permission by `chmod < permission > file` for one file or `chmod -R < permission > directory` for a directory recursively. Examples:

```
chmod 766 filename
chmod g+w filename
chmod g=wx filename
chmod o-w filename
chmod g+r, o-x filename
```

4 Text searching

- The `grep` command can be used to search for a text in a file.
- **Exercise:** search for the letter q in your text file with `grep q yourfile` and search for it in all files in your directory with `grep q *`. Try some other searches.
- In addition to searching for literal strings, you can look for more general expressions.

^	the beginning of the line
$	the end of the line
.	any character
*	any number of repetitions
[*xyz*]	any of characters `xyz`

- More examples: you can find
  - All lines that start with an 'a' with `grep` "^a" `yourfile`
  - All lines that end with a digit with `grep` "[0-9]$" `yourfile`.

  Exercise: Construct the search strings for finding
  - lines that start with an uppercase character
  - line that contain exactly one character.
- The stream editor `sed` is a tool for processing text files on a line-by-line basis. For instance
  sed 's/foo/bar/' myfile >mynewfile
  will apply the substitute command `s/foo/bar` to every line of `myfile`, i.e. search for `foo` in every line and replace it with `bar`. The output is then redirected (the expression >) to a new file.

5 Command execution

- How Unix determines what to do when you type a command name? Search paths: If you type a command such as `ls`, the shell actually goes searching for a program by the name `ls`. This means that you can have multiple different commands with the same name, and which one gets executed depends on which one is found first. The locations where Linux (or Unix) searches for commands is the 'search path', which is stored in the environment variable `PATH`.
- **Exercise:** Do `echo $PATH`. Add your own path to the search path by `export PATH=''yourpath:$PATH''`.
- How to feed one command into another? And how to connect commands to input and output files? Use redirection and pipelines.
  - Output redirection: More usefully, `grep string yourfile > outfile` will take what normally goes to the terminal and send it to `outfile`. The output file is created if it didn't already exist, otherwise it is overwritten. To append, use `grep text yourfile >> outfile`.
  - **Exercise:** Take one of the grep commands from the previous section, and send it to a file. Check that the contents of the file are identical to what appeared on your screen before. Search for a string that does not appear in the file and send the output to a file. What happens?
  - Standard files: Unix has three standard files that handle input and output.

  - `stdin` is the file that provides input for processes.
  - `stdout` is the file where the output of a process is written.
  - `stderr` is the file where error output is written.

  In an interactive session, all three files are connected to the user terminal. Using input or output redirection then means that the input is taken or output is sent to a different file than the terminal.

- Command redirection: Instead of taking input from a file, or sending output to a file, it is possible to connect two commands together, so that the second command takes the output of the first as input. The syntax for this is `commandone | commandtwo`, and this is called *pipeline*. For example, `grep a myfile | grep b` finds all lines that contains both an `a` and a `b`.

  Exercise: Construct a pipeline that counts how many lines there are in your file that contains the string `th`. Use the `wc` command to do the counting.

- Processes: The Unix operating system can run many programs at the same time, by rotating through the list and giving each only a fraction of a second to run each time. The command `ps` can tell you everything that is currently running.

- **Exercise:** Type `ps`. How many programs are currently running? By default, `ps` gives you only the programs that you explicitly started. Do `ps guwax` for a detailed list of everything that is running. How many programs are running? How many belongs to the root, how many to you? (You can use pipeline here.) Usually, you can kill a running process by using `kill processnumber` command.

Finally, the Unix shells are also programming environments. You can learn more about this aspect of Unix by finding an online tutorial.

References

[1] Victor Eijkhout. *Introduction to High Performamnce Scientific Computing*. 2nd edition, 2015.