Shell tutorial

Oana Cocarascu

This tutorial is intended to give you an overview of Linux and basic shell commands.

The shell is a program that takes commands and gives them to the Operating System (OS) to perform. On Unix systems (e.g. Linux), command line interfaces (CLIs) such as the shell used to be the only available user interfaces. Nowadays, there are also graphical user interfaces (GUIs). Both Linux and Unix provide various types of shell:

- sh (Bourne shell)
- bash (Bourne Again shell)
- csh (C shell)/tcsh (TC shell)
- ksh (Korn shell)

You can interact with the shell using a program called terminal emulator, shortly a "terminal".

There are several terminals available on the lab machines such as *terminal* and *terminator*. You can also press `Alt + F2` to open the "Run Command" dialog box where you type the name of the terminal you would like to use. After opening a terminal, you will see a shell prompt.

To get a clean window to work on (i.e. clearing the terminal) use the command `clear` or use `Ctrl + L`.

Exercise 1: Type some characters and press enter. For example, if I type `abcde`, I would get an error message as follows: `abcde: Command not found`. Clear the terminal.

Shell:

- To check the shells installed on your Linux type `cat /etc/shells`
- To check which shell you are using type `ps -p $$`
- Another option to check which shell you are using is `echo $0`

Command parsing: First token is the command whereas the remaining tokens are arguments to the command. Example: For `cat etc/shells`, `cat` is the command and `etc/shells` is the argument for the command.

For any command, you can check out the documentation (called *man page*) by typing `man commandName`. Example: `man ps` will show the documentation for the `ps` command. To exit the documentation, type `q`.

Exercise 2: Type the three commands above and check what they return.

History: You can see the list of the last commands by typing `history`.

- If you type `history N` and replace `N` with a number you will get the last `N` remembered commands.
  Example: `history 10` will show the last 10 commands.
- To move through the previous commands, press the up-arrow key to go back, and the down-arrow key to go forward.

Exercise 3: Type the command `history` and inspect the result. Navigate through the commands using the arrow keys and click on one of the commands. Type `history` and check the new results.

The files are organised in a hierarchical directory structure (a *tree-like* pattern of directories) where the first directory is called the "root" directory. The root directory contains files and subdirectories, which in turn can contain files and subdirectories.

File paths:

- `/` is the "root" directory
- $\sim$ is your home (and initial) directory
- `.` (period) references the current directory
- `..` references the parent directory

At any given time you are located in a single directory called "working directory" whose name you can find using the command `pwd`. Upon logging in, the working directory is set to your home directory (each user has his own home directory). Listing the files in the working directory is done using the command `ls`. Changing the working directory is done using the command `cd` followed by the *pathname* of the directory you want to change to. Pathnames can be of two types: *absolute pathnames* and *relative pathnames*.

Basic commands:

- `pwd` prints the working directory
- `ls` lists files and directories
- `cd` changes directory
  - `cd ..` to go up one directory
  - `cd` ∼ to go to your home directory
  - `cd` is same as above
  - `cd ../../someDirectory` to go up two directories then to `someDirectory`

An absolute pathname begins with the root directory and follows the tree branches until it reaches the path of the wanted directory. `/usr/bin` is a directory where most user program executables are found. This means that, from the root directory, there is a directory called "usr" which contains a directory called "bin".

Exercise 4: Type the following commands and check the output:

```
cd /usr/bin
pwd
ls
```

In contrast with an absolute pathname that starts from the root directory, a relative pathname starts from the working directory. It uses special notations to represent relative positions: `.` (the current directory) and `..` (the parent directory).

Exercise 5: Type the following commands and check the output:

```
cd /usr/bin
pwd
```

We can change the working directory to `/usr` (i.e. the parent of `/usr/bin`) in two ways:

1. using absolute pathnames:
   ```
   cd /usr
   pwd
   ```
2. using relative pathnames:
   ```
   cd ..
   pwd
   ```

Next, type the following:

```
cd /usr
pwd
```

We can change the working directory from `/usr` to `/usr/bin` in two ways:

1. using absolute pathnames:
   ```
   cd /usr/bin
   pwd
   ```
2. using relative pathnames:
   ```
   cd bin
   pwd
   ```

Variations of `ls`:

- `ls someDirectory` lists the files in the `someDirectory` directory (e.g. `ls Downloads`)
- `ls -l` lists the files in the working directory with detailed contents

Along with arguments, you can also pass options to commands. Optional arguments (such as the one in `ls -l`) are preceded by a minus sign.

Exercise 6: Check the man page of `ls` to find out which argument orders the files by time of last modification.

Exercise 7: Go to the `Downloads` directory and type the command `ls` followed by the command `ls -l`.

For example, one of the results I get is:

```
-rw-r--r-- 1 oc511 lai 2385836 Sep 13 2018 report.pdf
```

The first entry is a 10-character string representing the file permissions. The first character represent the file type: '-' for a regular file and 'd' for a directory. The next 3 groups of 3 characters indicate the file permissions for (1) the user, (2) group members, (3) anyone else. The 3 characters set represent the rwx (read, write and execute) rights. Dashes indicate that a particular permission in the rwx sequence has not been enabled.

The next entry represents the number of links to another program or file elsewhere on the system (do not worry about this).

Then, we have the name of the user who owns the file, the name of the group that has file permissions in addition to the file's owner, file size (in bytes), the last time the file was modified, and finally the name of the file /directory.

Files: File names in Linux are case sensitive: `File.txt` is different from `file.txt`. When typing a filename into the shell, once you have typed enough characters to uniquely select a single file, press the TAB key and the shell will complete the filename for you. If there are several alternatives, CTRL-D will list all the matching files and then it will redisplay the partial command line. If the file name has spaces, you need to escape them with backslash. Some examples of filename extensions are:

- text files `.txt`
- generic file `.dat`
- comma separated file `csv`
- shell script (bash/sh) `.sh`

To create an empty file: you can use the `touch fileName` command, replacing `fileName` with the name you want to give to the file.

In the following, you can write to files using your editor of choice. For a command line text editor, check the end of the tutorial for a quick introduction to `vim`.

To view text files: you can use the command `less fileName` where `fileName` is the name of the file you want to inspect. You can use the Page Up, Page Down, and the arrow keys to move through the text file. To exit `less` type `q`. Alternatively you can use the command `cat fileName`.

Exercise 8: Create a file, write some characters in it, and then view its contents using the `less` command.

Manipulating files:

- `mkdir dirName` creates a directory
- `rmdir dirName` removes a directory (must be empty)
- `rm fileName` removes a file. It can have the following optional arguments:
  - `-i` representing interactive mode where you are asked if you are sure you want to remove the file (press `y` if you want the file to be removed and any other key otherwise)
  - `-r` representing recursive in which case all its directories and contents are removed
- `cp`
  - `cp file1 file2` creates a copy of a file
  - `cp file1 file2 file3 ... directory` copies one or more files to a directory
  - optional arguments:
    - `-i` interactive, with warnings about over-writing
    - `-r` recursive, copies directories and contents
    - `-p` preserves file modification times (otherwise timestamp is current time)
- `mv`
  - `mv file1 file2` renames a file (i.e. "moves" it)
  - `mv file1 file2 file3 ... directory` moves one or more files to a directory
  - optional argument for interactive `-i`

Be careful with `rm`. Linux **does not have a command to undo a delete**. Once you delete a file using `rm`, it is gone.

Wildcards: There are special characters, called wildcards, that allow you to select filenames based on patterns of characters.

- `*` matches any string
  Example: `ls *.txt` refers to all files ending `.txt`
- `?` matches any single character
  Example: `ls file?.txt` and `ls file??.txt`
  `ls file?.txt` can refer to `file1.txt`, `file2.txt`, or any other similar file
- `[...]` matches any one of the enclosed characters
  Example: `ls file[12].txt` matches `file1.txt` and/or `file2.txt`
  `ls f*[12].txt` matches any file beginning with `f` and ending with `1.txt` or `2.txt`

Exercise 9: Create two directories in your home, `temp1` and `temp2`. Create a few files in each directory, with different extensions. Use the `ls` command to show all files in the directories and the files with a particular extension. Then delete some of them. Use `ls` to see the new contents of the directory.

Fundamental commands:

- `head fileName` shows the first few lines of a file (default 10)
- `tail fileName` shows the last few lines of a file (default 10)
- For both `head` and `tail` you can add `-n #` to show # lines instead of default number of lines
  Example: `head -n 20 file.txt`
- `cat` concatenates files
  Example: `cat file1.txt file2.txt file3.txt`
- `wc fileName` shows line, word, and character count

Output redirection: Shell can redirect output using the following:

- `>` standard output to file (overwrite)
- `>>` standard output to file (append)

Filters:

- A filter reads input from the standard input (stdin) and/or files, performs an operation (e.g. searching, sorting, summarising) and then writes the resulting output to the standard output (stdout).
- Filters can be linked together into a *pipeline* so that the output from one filter is the input to another filter.

Exercise 10: Try the following examples:

- Save directory listing
  `ls -l > temp.txt`
- Look at the 10 most recently modified files
  `ls -lt | head`
- Look at only the 10 oldest files
  `ls -lt | tail -n 10`
- Concatenate several files into a new one
  `cat file1.txt file2.txt > allfiles.txt`

Show only one scroll length of content at a time
`cat file1.txt | less`

Exercise 11: Putting everything together. Do the following:

- create a directory in your home directory calling it `temp`
- create several files and directories in the newly created directory, and then create files and directories in those directories
- rename several of the files and directories
- delete one of the directories that has other files and directories in them
- copy a file from one of your subdirectories into `temp`

VIM

To create and edit a file, you can use `vim`, a command line text editor. In `vim`, everything is done via the keyboard.

There are two modes in `vim`: *insert* and *edit*. In input mode you can enter content into the file. In edit mode you can move around the file, delete, copy, search and replace, etc. A common mistake is to start entering commands without first going into edit mode or to start typing input without going into insert mode.

The command `vim file.txt` opens the specified file or creates a new file with the given name and opens it. You always start off in *edit mode* so you need to switch to *insert mode* by pressing `i`. You can tell when you are in insert mode as the bottom left corner will tell you. Type a few characters. To get back to edit mode press `ESC`.

Once back in edit mode, there are several ways in which you can save the file and exit (need to press `enter` to complete the command):

- `:w` saves the file but does not exit
- `:wq` saves the file and exits
- `:q!` discards all changes since the last save and exits

To undo the last action, use `u`.

Commands to navigate the file:

- Arrow keys move the cursor around
- `$` moves cursor to the end of the current line
- `G` moves to the last line
- `w` moves to the beginning of the next word
- `nw` moves forward n words (eg `2w` moves two words forward)
- `b` moves to the beginning of the previous word
- `nb` moves back n words

Commands to delete content:

- `x` deletes a single character
- `nx` delete n characters
- `dd` deletes the current line

SSH

Secure shell (ssh) is a network protocol that allows secure and remote machine access. To access DoC Linux servers from outside the college network, you can use the following command:
`ssh username@shell1.doc.ic.ac.uk`
where `username` is your DoC user-name. You can use shell2, shell3 or shell4 instead of shell1.

To copy data to, from, or between different hosts, you can use `scp`. It uses ssh for data transfer and provides the same authentication and same level of security as ssh. The following commands may be of use:

- Copy the file `f.txt` from a remote host to the local host
  `scp username@remotehost:f.txt /some/local/directory`
- Copy the file `f.txt` from the local host to a remote host
  `scp f.txt username@remotehost:/some/remote/directory`
- Copy the directory `foo` from the local host to a remote host's directory `bar`
  `scp -r foo username@remotehost:/some/remote/directory/bar`

Example: `scp f.txt username@shell1.doc.ic.ac.uk:/homes/username/Downloads/`