Module 2

Getting Started

2.1 Files and Directories

- A **directory** is a collection of files and/or other directories
  - Because a directory can contain other directories, we get a directory **hierarchy**
- The ‘top level’ of the hierarchy is the **root directory**
- Files and directories can be named by a **path**
  - Shows programs how to find their way to the file
  - The root directory is referred to as `/`
  - Other directories are referred to by name, and their names are separated by slashes (`/`)
- If a path refers to a directory it can end in `/`
  - Usually an extra slash at the end of a path makes no difference

2.2 Examples of Absolute Paths

- An **absolute path** starts at the root of the directory hierarchy, and names directories under it:

  ```
  /etc/hostname
  ```

  - Meaning the file called *hostname* in the directory *etc* in the root directory
- We can use `ls` to list files in a specific directory by specifying the absolute path:

  ```
  $ ls /usr/share/doc/
  ```

2.3 Current Directory

- Your shell has a **current directory** — the directory in which you are currently working
- Commands like `ls` use the current directory if none is specified
- Use the `pwd` (print working directory) command to see what your current directory is:

```
$ pwd
/home/fred
```

- Change the current directory with `cd`:

```
$ cd /mnt/cdrom
$ pwd
/mnt/cdrom
```

- Use `cd` without specifying a path to get back to your home directory

2.4 Making and Deleting Directories

- The `mkdir` command makes new, empty, directories
- For example, to make a directory for storing company accounts:

```
$ mkdir Accounts
```

- To delete an empty directory, use `rmdir`:

```
$ rmdir OldAccounts
```

- Use `rm` with the `-r` (recursive) option to delete directories and all the files they contain:

```
$ rm -r OldAccounts
```

- Be careful — `rm` can be a dangerous tool if misused

2.5 Relative Paths

- Paths don't have to start from the root directory
  - A path which doesn't start with `/` is a **relative path**
  - It is relative to some other directory, usually the current directory
- For example, the following sets of directory changes both end up in the same directory:

```
$ cd /usr/share/doc

$ cd /
$ cd usr
$ cd share/doc
```

- Relative paths specify files inside directories in the same way as absolute ones

2.6 Special Dot Directories

- Every directory contains two special filenames which help making relative paths:
  - The directory `..` points to the parent directory
    - `ls ..` will list the files in the parent directory
  - For example, if we start from */home/fred*:

```
$ cd ..
$ pwd
/home
$ cd ..
$ pwd
/
```

- The special directory `.` points to the directory it is in
  - So `./foo` is the same file as `foo`

2.7 Using Dot Directories in Paths

- The special `..` and `.` directories can be used in paths just like any other directory name:

```
$ cd ../other-dir/
```

  - Meaning "the directory *other-dir* in the parent directory of the current directory"
- It is common to see `..` used to 'go back' several directories from the current directory:

```
$ ls ../../../../far-away-directory/
```

- The `.` directory is most commonly used on its own, to mean "the current directory"

2.8 Hidden Files

- The special `.` and `..` directories don't show up when you do `ls`
  - They are **hidden files**
- Simple rule: files whose names start with `.` are considered 'hidden'
- Make `ls` display all files, even the hidden ones, by giving it the `-a` (all) option:

```
$ ls -a
.    ..    .bashrc    .profile    report.doc
```

- Hidden files are often used for configuration files
  - Usually found in a user's home directory
- You can still read hidden files — they just don't get listed by `ls` by default

2.9 Paths to Home Directories

- The symbol ˜ (tilde) is an abbreviation for your home directory
  - So for user ‘fred’, the following are equivalent:

    ```
    $ cd /home/fred/documents/
    $ cd ~/documents/
    ```
- The ˜ is **expanded** by the shell, so programs only see the complete path
- You can get the paths to other users’ home directories using ˜, for example:

  ```
  $ cat ~alice/notes.txt
  ```
- The following are all the same for user ‘fred’:

  ```
  $ cd
  $ cd ~
  $ cd /home/fred
  ```

2.10 Looking for Files in the System

- The command `locate` lists files which contain the text you give
- For example, to find files whose name contains the word ‘mkdir’:

  ```
  $ locate mkdir
  /usr/man/man1/mkdir.1.gz
  /usr/man/man2/mkdir.2.gz
  /bin/mkdir
  ...
  ```
- `locate` is useful for finding files when you don’t know exactly what they will be called, or where they are stored
- For many users, graphical tools make it easier to navigate the filesystem
  - Also make file management simpler

2.11 Running Programs

- Programs under Linux are files, stored in directories like */bin* and */usr/bin*
  - Run them from the shell, simply by typing their name
- Many programs take options, which are added after their name and prefixed with `-`
- For example, the `-l` option to `ls` gives more information, including the size of files and the date they were last modified:

  ```
  $ ls -l
  drwxrwxr-x    2 fred   users     4096 Jan 21 10:57 Accounts
  -rw-rw-r--    1 fred   users      345 Jan 21 10:57 notes.txt
  -rw-r--r--    1 fred   users     3255 Jan 21 10:57 report.txt
  ```
- Many programs accept filenames after the options
  - Specify multiple files by separating them with spaces

2.12 Specifying Multiple Files

- Most programs can be given a list of files
  - For example, to delete several files at once:
    ```
    $ rm oldnotes.txt tmp.txt stuff.doc
    ```
  - To make several directories in one go:
    ```
    $ mkdir Accounts Reports
    ```
- The original use of `cat` was to join multiple files together
  - For example, to list two files, one after another:
    ```
    $ cat notes.txt morenotes.txt
    ```
- If a filename contains spaces, or characters which are interpreted by the shell (such as `*`), put single quotes around them:
  ```
  $ rm 'Beatles - Strawberry Fields.mp3'
  $ cat '* important notes.txt *'
  ```

2.13 Finding Documentation for Programs

- Use the `man` command to read the manual for a program
- The manual for a program is called its **man page**
  - Other things, like file formats and library functions also have man pages
- To read a man page, specify the name of the program to `man`:
  ```
  $ man mkdir
  ```
- To quit from the man page viewer press `q`
- Man pages for programs usually have the following information:
  - A description of what it does
  - A list of options it accepts
  - Other information, such as the name of the author

2.14 Specifying Files with Wildcards

- Use the `*` wildcard to specify multiple filenames to a program:

  ```
  $ ls -l *.txt
  -rw-rw-r--    1 fred   users      108 Nov 16 13:06 report.txt
  -rw-rw-r--    1 fred   users      345 Jan 18 08:56 notes.txt
  ```

- The shell expands the wildcard, and passes the full list of files to the program
- Just using `*` on its own will expand to all the files in the current directory:

  ```
  $ rm *
  ```

  - (All the files, that is, except the hidden ones)
- Names with wildcards in are called **globs**, and the process of expanding them is called **globbing**

2.15 Chaining Programs Together

- The `who` command lists the users currently logged in
- The `wc` command counts bytes, words, and lines in its input
- We combine them to count how many users are logged in:

  ```
  $ who | wc -l
  ```

- The `|` symbol makes a **pipe** between the two programs
  - The output of `who` is fed into `wc`
- The `-l` option makes `wc` print only the number of lines
- Another example, to join all the text files together and count the words, lines and characters in the result:

  ```
  $ cat *.txt | wc
  ```

2.16 Graphical and Text Interfaces

- Most modern desktop Linux systems provide a **graphical user interface** (GUI)
- Linux systems use the X window system to provide graphics
  - X is just another program, not built into Linux
  - Usually X is started automatically when the computer boots
- Linux can be used without a GUI, just using a command line
- Use `Ctrl+Alt+F1` to switch to a text console — logging in works as it does in X
  - Use `Ctrl+Alt+F2`, `Ctrl+Alt+F3`, etc., to switch between virtual terminals — usually about 6 are provided
  - Use `Ctrl+Alt+F7`, or whatever is after the virtual terminals, to switch back to X

2.17 Text Editors

- Text editors are for editing plain text files
  - Don't provide advanced formatting like word processors
  - Extremely important — manipulating text is Unix's *raison d'être*
- The most popular editors are Emacs and Vim, both of which are very sophisticated, but take time to learn
- Simpler editors include Nano, Pico, Kedit and Gnotepad
- Some programs run a text editor for you
  - They use the `$EDITOR` variable to decide which editor to use
  - Usually it is set to `vi`, but it can be changed
  - Another example of the component philosophy

2.18 Exercises

1. a. Use the `pwd` command to find out what directory you are in.

   b. If you are not in your home directory (*/home/USERNAME*) then use `cd` without any arguments to go there, and do `pwd` again.

   c. Use `cd` to visit the root directory, and list the files there. You should see *home* among the list.

   d. Change into the directory called *home* and again list the files present. There should be one directory for each user, including the user you are logged in as (you can use `whoami` to check that).

   e. Change into your home directory to confirm that you have gotten back to where you started.

2. a. Create a text file in your home directory called *shakespeare*, containing the following text:

   ```
   Shall I compare thee to a summer's day?
   Thou art more lovely and more temperate
   ```

   b. Rename it to *sonnet-18.txt*.

   c. Make a new directory in your home directory, called *poetry*.

   d. Move the poem file into the new directory.

   e. Try to find a graphical directory-browsing program, and find your home directory with it. You should also be able to use it to explore some of the system directories.

   f. Find a text editor program and use it to display and edit the sonnet.

3. a. From your home directory, list the files in the directory */usr/share*.

   b. Change to that directory, and use `pwd` to check that you are in the right place. List the files in the current directory again, and then list the files in the directory called *doc*.

   c. Next list the files in the parent directory, and the directory above that.

   d. Try the following command, and make sure you understand the result:

   ```
   $ echo ~
   ```

   e. Use `cat` to display the contents of a text file which resides in your home directory (create one if you haven't already), using the `~/` syntax to refer to it. It shouldn't matter what your current directory is when you run the command.

4. **a.** Use the `hostname` command, with no options, to print the hostname of the machine you are using.

b. Use `man` to display some documentation on the `hostname` command. Find out how to make it print the IP address of the machine instead of the hostname. You will need to scroll down the manpage to the 'Options' section.

c. Use the `locate` command to find files whose name contains the text 'hostname'. Which of the filenames printed contain the actual `hostname` program itself? Try running it by entering the program's absolute path to check that you really have found it.

5. **a.** The `*` wildcard on its own is expanded by the shell to a list of all the files in the current directory. Use the `echo` command to see the result (but make sure you are in a directory with a few files or directories first)

b. Use quoting to make `echo` print out an actual `*` symbol.

c. Augment the *poetry* directory you created earlier with another file, *sonnet-29.txt*:

```
When in disgrace with Fortune and men's eyes,
I all alone beweep my outcast state,
```

d. Use the `cat` command to display both of the poems, using a wildcard.

e. Finally, use the `rm` command to delete the *poetry* directory and the poems in it.