Content

File System Management

First of all some useful command (described later) -

1. `mkdir` - create a directory
2. `rmdir` - delete a directory
3. `ls` - list files/directories in current directory
4. `cd` - change directory
5. `cp` - copy a file
6. `cat` - examine the contents of the file
7. `rm` - delete a file

Keyboard Control

Essential control signals

```
^C - interrupt command
^Z - suspend command
^D - end of file
^H - (^?) [DELETE] - delete last character
^W - delete last word
^U - delete line
^S - suspend output (rarely used)
^Q - continue output (rarely used)
```

Unix / Linux Tutorial

What's Unix?

The Unix operating system is a set of programs that act as a link between the computer and the user.

The computer programs that allocate the system resources and coordinate all the details of the computer's internals is called **operating system** or the **kernel**.

Users communicate with the kernel trough a program know as the **shell**. The shell is a command line interpreter; it translates commands entered by the user and converts them into a language that is understood by the kernel.

Unix Files

Files are stored within the Unix file system and contain **streams of bytes** without structure. Files may contain *data*, *text* or *executable*. Most Unix systems allow up to *256* characters to be used in a filename.

Listing Directories, Files and Processes

All data in Unix is organized into files and all files are organized into directories. These directories are organized into a *tree-like structure* called the *filesystem*. Starting from an initial top-level (*root*) directory sub-directories successively organise information into categories, and then sub-categories.

Every Unix system are setup with an *on-line* manual. To use it type in your terminal the following command -

```
$ man nameOfTheCommand
```

The `man` provides a means of determining how commands works (not very useful if the user is unsure of the command's name). You can achieve this using the `-k` option -

```
$ man -k nameOfTheCommand
```

Listing Directories and Files

To list the files and directories in the current directory, use the following command -

```
$ ls
```

here is the sample output of the above command -

```
$ ls
bin     hosts   lib       res.03
docs    hw3     res.02    work
```

The `ls` command supports the `-l` option which would help you to get more information about the listed files -

```
$ ls -l
total 2
drwxrwxr-x 2 JackSparrow JackSparrow  4096 Dec 23 09:59 uml
```

The information about the listed columns

1. column - represents the **file type** and the permission given on the file.
2. column - represents the **number of memory blocks** taken by the file or directory.
3. column - represents the **owner of the file**. The Unix user who created the file.
4. column - represents the **group of the owner**. Every Unix user will have an associated group.
5. column - represents the **file size in bytes**.
6. column - represents the **date and the time when this file was created or modified** for the last time.
7. column - represents the **file or the directory name**.

Prefix & Description

1. **-** -> regular file, such an ASCII text
2. **b** -> block special file. Block I/O device file such as a physical hard drive.
3. **c** -> character special file. Raw I/O device file such as a physical hard drive.
4. **d** -> directory file that contains a listing of other files and directories.
5. **p** -> named pipe. A mechanism for interprocess communications.
6. **s** -> socket used for interprocess communication.
7. **l** -> symbolic link file. Links on any regular file.

More Listing Options

Option	Description
ls -a	list all files including hidden files starting with '.'
ls -A	list almost all files including hidden files starting with '.', **except the single '.' and '..'**
ls -d	the directory file not its contents
ls -F	show file type
ls -i	list files i-node index number
ls -l	list with long format - shows permissions
ls -la	list long format including hidden files
ls -lh	list long format with readable file size
ls -ls	list with long format with file size
ls -r	list in reverse order
ls -R	list recursively directory tree
ls -s	list file size
ls -S	sort by file size in ascending order
ls -t	sort by time & date
ls -X	sort by extension name
ls -g	used with -l for group ownership

Listing Processes

To list out processes use `ps` command -

```
$ ps
```

To see more information about the processes use `ps -f` instead of simple `ps` command-

```
$ ps -f
```

Using the `-u uid` (own user ID) option it displays all processes owned by the user. Using the `ps -ef` options displays all processes on the system. In order to see the size of all programs, use `ps -ely` (`ps -el` on macOS).

```
$ ps -e
```

Running `ps` with the `-e` option will show you all of the active processes on the system, regardless of who their owner is. Since there are usually a large number of active system processes, it is wise to pipe the results of this command to the more command.

You can use the `-f` and `-e` options together.

Who Are You

While logged into the system, you might be willing to know **Who am I**?

The easiest way to find out "*whou you are*" is to enter the `whoami` command -

```
$ whoami
JackSparrow
```

Basically `who` is a function which takes 2 arguments. The `am` and the `I` .

Who is Logged in?

There are three commands available to get you know who is loggen into the system, based on how much you wish to know about the other users:

1. `users`
2. `who`
3. `w`

```
$ users
JackSparrow WillTurner LizSwann

$ who
JackSparrow ttyp0 Oct 8 14:10 (limbo)
WillTurner  ttyp0 oct 4 0:45  (calliope)
LizSwann    ttyp0 Oct 8 12:11 (dent)
```

Metacharacters - Shell Wildcards

Use '*****' to match 0 or more characters, a question mark ('**?**') matches with single character and the '**[ab]**' *a* or *b* or *specified range*.

For exmaple -

```
$ ls ch*.doc
```

Displays all the files, the names of which start with '**ch**' and end with '**.doc**'.

Hidden Files

An invisible file is one, which start with a single dot character ('**.**').

To list the invisible files, use `-a` option to `ls`

```
$ ls -a
.       .profile    docs    lib     test_results
..      .rhtosts    hosts   pub     users
.emacs  bin         hw1     res.01  work
```

Single dot ('.') - represents the **current** directory.

Double dot ('..') - represents the **parent** directory.

Creating Files

To create files in Unix system you can use the following commands:

1. `$ vi fileName`
2. `$ > fileName`
3. `$ touch fileName`

Display Content of a File

Use `cat` (*concatenate*) command to see the content of a file.

```
$ cat myfile
Hello World
```

To display the row numbers use `-b` option along with the `cat` command.

```
$ cat -b myfile
1. Hello World
```

Note: `-b` will add numbers to only non-empty lines. If you want to add row number to all lines, including empty lines too, use `-n` option -

```
$ cat -n fileName
```

You can list out the content of the files almost the same way with `more` and with `less` .

Displaying Parts of Files

`head` displays the first line of a file, by default the first 10 lines. Argument allows different number to be shown.

```
$ head [-number] fileName
```

`tail` displays the last lines of a file, by default the last 10.

```
$ tail +|- number [-f] [-files ...]
```

`tail` is more poweful than head, because it is able to output the end of the file relative to the start or to the end. Using `tail -n` it operates relative to the end, using `tail +n` it operates relative to the start. Using `tail -f` it outputs the end of the file and then waits.

Searching Files

`grep` searches files for strings.

```
$ grep [-cilnvw] <RegEx> [files ...]
```

`grep` options -

1. `-c` - display count of matching lines
2. `-i` - case insensitive
3. `-l` - list names of files containing matching lines
4. `-n` - precede each line by its line number
5. `-v` - only display lines that do **not** match
6. `-w` - search for the expression as a word

Regular expressions which can be used in `grep` -

1. `.` - match any character
2. `*` - match zero or more occurrences of previous character
3. `^` - match beginning of line
4. `$` - match end of line
5. `[abc]` - match any one of *a*, *b*, *c*
6. `[a-z]` - match any character in range

Translating Characters

`tr` translates characters from the standard input. Characters mentioned in *first* argument translated to corresponding character in second argument.

```
$ tr '[a-z]' '[A-Z]'
hello
HELLO
```

Note:

```
$ tr '[:lover:]' '[:upper:]'
```

will do the same.

- Delete characters from an input stream

```
$ tr -d '[aeiou]'
Hello World
Hll wrld
```

- Use complement of input character pattern

```
$ tr -cd '[aeiou]'
Hello Worlds
eoo
```

- use `-s` (squeeze) option to remove duplicate replaced characters. The `tr` command needs some input so you can simply redirect the `ls` command's output into the `tr` . For example -

```
$ ls | tr -s " "
```

Sorting Files

`sort` organises files into alpha-numeric order, by default ordering is increasing alphabetic order, entire line used as a key.

Lines are divided into fields

- space / tab characters used as separator
- allow more specific definition of sort keys

Options for sorting files -

- `-n` - sort on numeric value
- `-r` - reverse sort
- `-ka,b` - sort key starts at field **a** (*1-based*) and ends at field **b**.

```
$ sort -nr -k4,5 fileName
```

Comparing Files

`diff` displays only the differences between two files.

- Lines in the **first** file which differ from the second are prefixed with '**<**'.
- Lines in the **second** file which differ from the first are prefixed with '**>**'.

```
$ diff file1 file2
```

The `-e` option causes `diff` to generate output which describes exactly how **file1** can be generated from **file2**.

Cutting Fields

`cut` removes selected fields from each line of the file

- uses tab as field separator character.
- can also use character offsets specify cut region.

```
$ cut -d" " -f1 fileName
```

In the above example, `cut` selects field one from *fileName*, where fields are delimited by a single space. By default the field delimiter is a (single) tab.

You can use characters range to select which column you want to cut.

```
$ cut -c17-22 myFile
```

Counting Words in a File

Use `wc` (*word counter*) command to get a count of the total number of lines, words, and characters contained in a file.

```
$ cat myfile
Hello World

$ wc myfile
1 2 12 myfile
```

1. column represents the total number of **lines** in the file.
2. column represents the total number of **words** in the file.
3. column represents the total number of **bytes** in the file. This is the actual size of the file.
4. column represents the name of the file.

Options for `wc` -

- `-l` - just report lines
- `-w` - just report words
- `-c` - just report characters

Pasting Files

`paste` allows you to combine the contents of two or more files, concatenating lines from the files into single lines in the output. If we reach the end of one of the files, then empty string is inserted into the result.

```
$ paste file1 file2
```

We can also use the filename '**-**' to represent using standard input as one of the files to read from.

```
$ cat file1 | paste file2 -
```

Duplicate Lines

`uniq` removes duplicate adjanced lines from files.

```
$ uniq fileName
```

Options for `uniq` -

- `-c` - count number of duplicate lines
- `-d` - only print repeated lines
- `-u` - only print non-repeated lines

Non ASCII Files

Octal dump `od` displays the contents of any file in *octal*, *decimal*, *hexadecimal* or *ASCII* format.

Options for `od` -

- `-x` - display in hexadecimal
- `-c` - display printable characters where possible
- `-d` - display in decimal

```
$ od -c fileName
```

Deleting Files

To delete an existing file, use `rm` (*remove mapping*) command.

```
$ rm fileName
```

Caution - a file may contain useful information. It is always recomennded to be careful while using this `rm` command. It is better to use the `-i` option along with `rm` command. *Remember: in Unix once deleted file never can be retrieved*.

Remove multiple files at a time -

```
$ rm fileName1 fileName2 ... fileNameN
```

The `rm` command also has the `-r` option similar to chose of `cp` . The `-r` option is necessary if directory structures must be deleted.

```
$ rm -r nameOfDir
```

Copying Files

`cp` copies files and directories around the file system. Copy means duplicating the bytes on disk representing the contents of the files being copied. `cp` is used with two arguments when copying from one file to another and with many arguments when copying a collection of files into a directory. `cp` is also be used to copy the contents of one directory to an another. In this case `-r` (*recursive*) option must be supplied. When copying directories, if the target **d2** exist, then the source **d1** is created within it. A file **f1** within **d1** may also be accessed as **d2/d1/f1**. If, however, the target does not exist, the file **f1** within **d1**, may now also be accesseed as **d2/d1**.

By default, the `cp` command overwrites any files which already exist with the target name. The `-i` (*interactive*) option may be used in order to get `cp` to promt prior to overwriting any existing files.

To preserve a file's modification time and permission bits, use `-p` option.

```
$ cp [-ip] f1 f2
$ cp [-ip] f1 f2 ... fn d
$ cp -r [-ip] d1 d2
```

Copying multipe files it is important that the last argument must be a directory.

Moving / Deleting Files

`mv` is used to rename files and directories. It does not cause the contents of the file to be physically moved, only the file's name is changed in its directory.

The new name may be a path to another directory, so `mv` can in fact move a file or directory from one place to another.

There is no need to recursive option. The `-i` option may be used if there is a danger of overwriting

existing files.

```
$ mv [-i] f1 f2
$ mv [-i] f1 f2 ... fn d
$ mv [-i] d1 d2
```

Linking Files

All directories have at least two names, their name in the parent directory and in themselves. With each sub-directory, a new name is created for the parent.

Hard Links

Use the `ln` command to create links -

```
$ ln existingFile linkName
```

It will increase the reference number to the given file or directory for which the link was created for. Links are destroyed by using the `rm` command. When the number of links referring to a file is zero, the actual file contents are removed.

Symbolic Links - Symlinks

Look like hard links but implemented by reference. Symbolic file contains pathname of file it refers to, each has a distinct i-node number. To create symbolic link use the following command -

```
$ ln –s existingFile linkName
```

To reach the file or the directory it is always faster via hardlink than symlink.

Standard Unix Streams

Under normal circumstances, every Unix program has three streams (files) opened for when it starts up -

1. **stdin** - this is referred to as the **standard input** and the associated file descriptor is **0**.
2. **stdout** - this is referred to as the **standard output** and the associated file descriptor is **1**.
3. **stderr** - this is deferred to as the **standard error** and the associated file descriptor is **2**.

Home Directory

The directory in which you find yourself when you first login is called your home directory.

You can go in your home directory anytime using the following command -

```
$ cd ~
```

Here `~` indicates the home directory. If you want to go to another user's home directory use the following command -

```
$ cd ~anotherUserName
```

Go back to your last directory use -

```
$ cd –
```

Go to the root directory -

```
$ /
```

Unix using a hierarchial structure for oraganzing files and directories. This structure is often referred to as a directory tree. The tree has a single root node, the slash character ('**/**').

Go back to a parent directory -

```
$ cd ..
```

Caution - in the root directory ('**/**') the '**.**' and the '**..**' are the same. No difference between

```
$ cd .
```

and

```
$ cd ..
```

Absolute / Relative Pathnames

Directories are arranged in a hierarchy with root ('**/**') at the top. The position of any file within the hierarchy is described by its pathname.

Elements of a pathname are separated by a '**/**'. A pathname is **absolute, if it is described in relation to root**, thus **absolute pathnames always begin with a '/'**. Absolute pathnames are unique. See in below example -

```
/etc/passwd/
/dev/rdsk0s3/
```

A pathname can also be relative to your current working directory. **Relative pathnames never begin with ' / '**. Relative to user amrood's home directory. Relative pathnames are not unique since they depend on the directory in which the path is specified. Some pathnames might look like this -

```
chem/notes
personal/res
```

To determine where you are within the filesystem hierarchy at any time, enter the command `pwd` (*print working directory*) to print the current working directory -

```
$ pwd
/usr/local/JackSparrow
```

Listing Directories .(dot) and .. (dot dot)

The **filename .** (dot) represents the current working directory; and the **filename ..** (dot dot) represents the directory one level above the current working directory (expect when you are in the root [see above example]) often referred to as the parent directory.

If we enter the command to show a listing of the current working diretories/files and use the `-a` option to list all the files and the `-l` option to provide the long listing, you will receive the following result -

```
$ ls -la
drwxrwxr-x 4 teacher class 2048 Jul 16 17.56 .
```

Listing Directories

To list the files in a directory use the following syntax -

```
$ ls dirName
```

To list the current directory, simply use `ls` .

```
$ ls
```

You can add subdirectories in a directory. See below example -

```
$ ls usr/local/home
```

Creating Directories

Directories are created by the following command -

```
$ mkdir dirName
```

Here, directory is the absolute or relative pathname of the directory you want to create. For exmaple -

```
$ mkdir myDir
```

Creates the directory *myDir* in the current directory. Another example -

```
$ mkdir /tmp/test
```

This command creates the directory *test* in the */tmp* directory.

Using `-p` option, `mkdir` is able to create missing parent directories as needed -

```
$ mkdir -p first/seconds/third
```

will create the missing parent directories *first* and *second* if they do not already exist.

If you give more than one directory on the command line, `mkdir` creates each of the directories. For example -

```
$ mkdir docs pub
```

Creates the directories **docs** and **pub** under the current direcotry.

Removing Directories

Directories can be deleted using the `rmdir` command as follows -

```
$ rmdir dirName
```

Note - to remove a directory, make sure it is empty which means there should not be any file or sub-direcotry inside this directory.

If you want to remove the sub-directories use the `-r` with `rmdir`.

```
$ rmdir -r dirName
```

It will remove all sub-directories recursively which are in *dirName*.

Remove multiple directories at a time as follows -

```
$ rmdir dirName1 dirName2 .. dirNamN
```

The above command removes the directories if they are empty. The `rmdir` command produces no output if it is succesfull.

Changing Directories

Use `cd` command to change to any directory by specifying a valid absolute or relative path.

```
$ cd dirName
```

dirName is the name of the directory that you want to change to.

Use absolut pathname to change the directory -

```
$ cd /usr/local
```

Renaming Directories

The `mv` (*move*) can also be used to rename directory -

```
$ mv oldDir newDir
```

Permission Bits

Every file carries permissions for its owner (*user*), group and others (*everyone else*)

Permissions for files

- **r** *read* - read contents
- **w** *write* - alter contents
- **x** *execute* - attempt to run program
- **s** *set ID* - change the UID or GID of process

Permission for directories

- **r** *list* - list contents
- **w** *write* - create and delete *any* files

- **x** *search* - access the directory (with a *path*, `cd`)
- **t** *sticky* - finer control over write access to directories

The permission bits are organized in three sets of three bits, following the file type field.

type	user	group	others
d	r w x	r w -	r - -
	4 2 1	4 2 -	4 - -

Note: about types see more in chapter: *Listing Directories, Files and Processes > Prefix & Description*.

Sticky Bit

The *Sticky Bit* allows a file to be deleted if

- the user has write permission on the directory, and
- the user owns the file, or ownd the directory, or is the superuser

To add a sticky bit to your file use the following command -

```
$ chmod +t fileName
```

for remove

```
$ chmod -t fileName
```

Setting Permissions

`chmod` is used to change permissions. Only the owner may change the permissions, or the superuser.

```
$ chmod [-R] <permission-mode> file [files ...]
```

The `-R` option is enables recursion through a directory hierarchy.

The permission-mode may by specified using

- a **symbolic** notation.
- a **numeric** notation.

Symbolic Notation

The symbolic notation allows file permissions to be changed relative to the current permission.

```
$ chmod u + r files...
        g - w
        o = x
        a   s
            t
```

The permission is applied to **u** (*user*), **g** (*group*), **o** (*others*), **a** (*all of them*) or any combination thereof. Permissions may be added, substracted or assigned

- **+** - addd new permission to the existing bits
- **-** - remove new permission from the existing bits
- **=** - override existing permission with new permission

Example -

```
$ chmod ug+rx, o-w myFile
```

Numeric notation

Treats the permission as a bit pattern.

```
$ chmod 777
```

(r = 4 ,w = 2 ,x = 1 -> 4 + 2 + 1 = 7)

An extra leading octal digit may be used in the numeric notation. This provides for set

- **UID** (4) [User ID]
- **GID** (2) and
- **sticky** (1).

Setting a file's permissions to *7777* would enable everything, *rwsrwsrwt*.

Note: a lowercase *s* or *t* implies that the *x* permission is also enabled. If *x* is absent, then the *S* and *T* appear as capital letter.

Default permissions

`umask` defines the default permission bits. Default for files and directories created by process. Not applied when files or directories are copied.

Command pipelines

The output of one command is directed into the input of another to from a pipeline. Always executed from **left to rigth**.

```
$ ls | grep '^d' | sort
```

The `tee` command enables the stream to be diverted.

```
$ ls | tee myFile | sort
```

The `tee` commands reads from **STDIN** (*0*) and writes to **STDOUT** (*1*) and to any files which are names as arguments. In the example above `tee` reads the output from the `ls` and writes this to *myFIle* and passes the data to `sort` .

Composing Commands

Sequential composion

```
$ cmd1; cmd2
```

Parallel composion

```
$ cmd1 & cmd2
```

Conjuctive composion - the condition is true if both of them is true

```
$ cmd1 && cmd2
```

Disjunctive composion - the condition is true if one of them is true

```
cmd1 || cmd2
```

Foreground and Background Jobs

- **A foreground job has access to the terminal for its input and output**
    - shell waits for foreground job to finish before promting for the next command
- **A background job has no access to the terminal for input**
    - may have access for output
    - shell will **not** wait for background job to complete before promting for the next command.

Foreground and background jobs can be:

- terminated
- suspended
- resumed

To start a process in the background use the ampersand character ('**&**') after the command. For example -

```
$ sleep 30 &
```

Job Numbers

When a background processes are invoked the shell tags them with a *job number*.

```
$ sleep 30 &
[1] 10708
```

Job number is **not** the same as process id. To kill a backgound process use the `kill` command and the process ID like below -

```
$ kill 10708
```

If you want to see how many process running in the backgorund use the `jobs` keyword to list them out -

```
$ jobs
[1]+ Running            sleep 30 &
```

the **[1]** is the job number which is running at the moment. The '**+**' sign indicates that it is the most recent background job and at the end of the line there is a process name itself.

Keyboard combinations for jobs:

```
^Z      suspends the foreground job
^C      terminates the foregorund job
fg      resumes a suspended job in the foreground
bg      resumes a suspended job in the background
jobs    list currently active jobs
stop    suspends the specified background job (ksh)
kill    send signal to job (ksh and bash)
```