Introduction to UNIX: Lecture Three

3.1 Objectives

This lecture covers:

- File and directory permissions in more detail and how these can be changed.
- Ways to examine the contents of files.
- How to find files when you don't know how their exact location.
- Ways of searching files for text patterns.
- How to sort files.
- Tools for compressing files and making backups.
- Accessing floppy disks and other removable media.

3.2 File and Directory Permissions

Permission	File	Directory
read	User can look at the contents of the file	User can list the files in the directory
write	User can modify the contents of the file	User can create new files and remove existing files in the directory
execute	User can use the filename as a UNIX command	User can change into the directory, but cannot list the files unless (s)he has read permission. User can read files if (s)he has read permission on them.

Fig 3.1: Interpretation of permissions for files and directories

As we have seen in the previous chapter, every file or directory on a UNIX system has three types of permissions, describing what operations can be performed on it by various categories of users. The permissions are read (`r`), write (`w`) and execute (`x`), and the three categories of users are user/owner (`u`), group (`g`) and others (`o`). Because files and directories are different entities, the interpretation of the permissions assigned to each differs slightly, as shown in Fig 3.1.

File and directory permissions can only be modified by their owners, or by the superuser (`root`), by using the `chmod` system utility.

- `chmod` (change [file or directory] mode)

  ```
  $ chmod options files
  ```

  `chmod` accepts options in two forms. Firstly, permissions may be specified as a sequence of 3 octal digits (octal is like decimal except that the digit range is 0 to 7 instead of 0 to 9). Each octal digit represents the access permissions for the user/owner, group and others respectively. The mappings of permissions onto their corresponding octal digits is as follows:

---	**0**
--x	**1**
-w-	**2**
-wx	**3**
r--	**4**
r-x	**5**
rw-	**6**
rwx	**7**

  For example the command:

  ```
  $ chmod 600 private.txt
  ```

  sets the permissions on `private.txt` to `rw-------` (i.e. only the owner can read and write to the file).

Permissions may be specified symbolically, using the symbols `u` (user), `g` (group), `o` (other), `a` (all), `r` (read), `w` (write), `x` (execute), `+` (add permission), `-` (take away permission) and `=` (assign permission). For example, the command:

```
$ chmod ug=rw,o-rw,a-x *.txt
```

sets the permissions on all files ending in `*.txt` to `rw-rw----` (i.e. the owner and users in the file's group can read and write to the file, while the general public do not have any sort of access).

`chmod` also supports a `-R` option which can be used to recursively modify file permissions, e.g.

```
$ chmod -R go+r play
```

will grant group and other read rights to the directory `play` and all of the files and directories within `play`.

- `chgrp` (change group)

  `$ chgrp` *group files*

  can be used to change the group that a file or directory belongs to. It also supports a `-R` option.

3.3 Inspecting File Content

Besides `cat` there are several other useful utilities for investigating the contents of files:

- `file` *filename(s)*

  `file` analyzes a file's contents for you and reports a high-level description of what type of file it appears to be:

```
$ file myprog.c letter.txt webpage.html↵
myprog.c:      C program text
letter.txt:    English text
webpage.html:  HTML document text
```

file can identify a wide range of files but sometimes gets understandably confused (e.g. when trying to automatically detect the difference between C++ and Java code).

- `head, tail` *filename*

  `head` and `tail` display the first and last few lines in a file respectively. You can specify the number of lines as an option, e.g.

  ```
  $ tail -20 messages.txt↵
  $ head -5 messages.txt↵
  ```

  `tail` includes a useful `-f` option that can be used to continuously monitor the last few lines of a (possibly changing) file. This can be used to monitor log files, for example:

  ```
  $ tail -f /var/log/messages↵
  ```

  continuously outputs the latest additions to the system log file.

- `objdump` *options binaryfile*

  `objdump` can be used to disassemble binary files - that is it can show the machine language instructions which make up compiled application programs and system utilities.

- `od` *options filename* (octal dump)

  `od` can be used to displays the contents of a binary or text file in a variety of formats, e.g.

  ```
  $ cat hello.txt↵
  hello world
  $ od -c hello.txt↵
  0000000  h  e  l  l  o     w  o  r  l  d \n
  0000014
  $ od -x hello.txt↵
  ```

```
0000000 6865 6c6c 6f20 776f 726c 640a
0000014
```

There are also several other useful content inspectors that are non-standard (in terms of availability on UNIX systems) but are nevertheless in widespread use. They are summarised in Fig. 3.2.

File type	Typical extension	Content viewer
Portable Document Format	.pdf	acroread
Postscript Document	.ps	ghostview
DVI Document	.dvi	xdvi
JPEG Image	.jpg	xv
GIF Image	.gif	xv
MPEG movie	.mpg	mpeg_play
WAV sound file	.wav	realplayer
HTML document	.html	netscape

Fig 3.2: Other file types and appropriate content viewers.

3.4 Finding Files

There are at least three ways to find files when you don't know their exact location:

- `find`

  If you have a rough idea of the directory tree the file might be in (or even if you don't and you're prepared to wait a while) you can use `find`:

  `$ find` *directory* `-name` *targetfile* `-print` ↵

  `find` will look for a file called *targetfile* in any part of the directory tree rooted at *directory*. *targetfile* can include wildcard characters. For example:

  `$ find /home -name "*.txt" -print 2>/dev/null` ↵

  will search all user directories for any file ending in "`.txt`" and output any matching files (with a full absolute or

relative path). Here the quotes (`"`) are necessary to avoid filename expansion, while the `2>/dev/null` suppresses error messages (arising from errors such as not being able to read the contents of directories for which the user does not have the right permissions).

`find` can in fact do a lot more than just find files by name. It can find files by type (e.g. `-type f` for files, `-type d` for directories), by permissions (e.g. `-perm o=r` for all files and directories that can be read by others), by size (`-size`) etc. You can also execute commands on the files you find. For example,

```
$ find . -name "*.txt" -exec wc -l '{}' ';'
```

counts the number of lines in every text file in and below the current directory. The `'{}'` is replaced by the name of each file found and the `';'` ends the `-exec` clause.

For more information about `find` and its abilities, use `man find` and/or `info find`.

- `which` (sometimes also called `whence`) *command*

  If you can execute an application program or system utility by typing its name at the shell prompt, you can use `which` to find out where it is stored on disk. For example:

  ```
  $ which ls ↵
  /bin/ls
  ```

- `locate` *string*

  `find` can take a long time to execute if you are searching a large filespace (e.g. searching from `/` downwards). The `locate` command provides a much faster way of locating all files whose names match a particular search string. For example:

  ```
  $ locate ".txt" ↵
  ```

will find all filenames in the filesystem that contain "`.txt`" anywhere in their full paths.

One disadvantage of `locate` is it stores all filenames on the system in an index that is usually updated only once a day. This means `locate` will not find files that have been created very recently. It may also report filenames as being present even though the file has just been deleted. Unlike `find`, `locate` cannot track down files on the basis of their permissions, size and so on.

3.5 Finding Text in Files

- `grep` (General Regular Expression Print)

  `$ grep` *options pattern files* ↵

  `grep` searches the named files (or standard input if no files are named) for lines that match a given pattern. The default behaviour of `grep` is to print out the matching lines. For example:

  ```
  $ grep hello *.txt ↵
  ```

  searches all text files in the current directory for lines containing "hello". Some of the more useful options that `grep` provides are:
  `-c` (print a count of the number of lines that match), `-i` (ignore case), `-v` (print out the lines that don't match the pattern) and `-n` (printout the line number before printing the matching line). So

  ```
  $ grep -vi hello *.txt ↵
  ```

  searches all text files in the current directory for lines that do not contain any form of the word hello (e.g. Hello, HELLO, or hELlO).

  If you want to search all files in an entire directory tree for a particular pattern, you can combine `grep` with `find` using backward single quotes to pass the output from `find`

into `grep`. So

```
$ grep hello `find . -name "*.txt" -print`↵
```

will search all text files in the directory tree rooted at the current directory for lines containing the word "hello".

The patterns that `grep` uses are actually a special type of pattern known as **regular expressions**. Just like arithemetic expressions, regular expressions are made up of basic subexpressions combined by operators.

The most fundamental expression is a regular expression that matches a single character. Most characters, including all letters and digits, are regular expressions that match themselves. Any other character with special meaning may be quoted by preceding it with a backslash (`\`). A list of characters enclosed by '`[`' and '`]`' matches any single character in that list; if the first character of the list is the caret `^', then it matches any character not in the list. A range of characters can be specified using a dash (`-`) between the first and last items in the list. So `[0-9]` matches any digit and `[^a-z]` matches any character that is not a digit.

The caret `^' and the dollar sign `$' are special characters that
match the beginning and end of a line respectively. The dot '.' matches any character. So

```
$ grep ^..[l-z]$ hello.txt↵
```

matches any line in `hello.txt` that contains a three character sequence that ends with a lowercase letter from l to z.

`egrep` (extended grep) is a variant of grep that supports more sophisticated regular expressions. Here two regular expressions may be joined by the operator `|'; the resulting regular expression matches any string matching either subexpression. Brackets '`(`' and '`)`' may be used for

grouping regular expressions. In addition, a regular expression may be followed by one of several repetition operators:

`?' means the preceding item is optional (matched at most once).
`*' means the preceding item will be matched zero or more times.
`+' means the preceding item will be matched one or more times.
`{N}' means the preceding item is matched exactly N times.
`{N,}' means the preceding item is matched N or more times.
`{N,M}' means the preceding item is matched at least N times, but not more than M times.

For example, if `egrep` was given the regular expression

```
'(^[0-9]{1,5}[a-zA-Z ]+$)|none'
```

it would match any line that either:

- begins with a number up to five digits long, followed by a sequence of one or more letters or spaces, or
- contains the word `none`

You can read more about regular expressions on the `grep` and `egrep` manual pages.

Note that UNIX systems also usually support another `grep` variant called `fgrep` (fixed grep) which simply looks for a fixed string inside a file (but this facility is largely redundant).

3.6 Sorting files

There are two facilities that are useful for sorting files in UNIX:

- `sort` *filenames*

`sort` sorts lines contained in a group of files alphabetically (or if the `-n` option is specified) numerically. The sorted output is displayed on the screen, and may be stored in another file by redirecting the output. So

```
$ sort input1.txt input2.txt > output.txt↵
```

outputs the sorted concentenation of files `input1.txt` and `input2.txt` to the file `output.txt`.

- `uniq` *filename*

  `uniq` removes duplicate adjacent lines from a file. This facility is most useful when combined with `sort`:

  ```
  $ sort input.txt | uniq > output.txt↵
  ```

3.7 File Compression and Backup

UNIX systems usually support a number of utilities for backing up and compressing files. The most useful are:

- `tar` (tape archiver)

  `tar` backs up entire directories and files onto a tape device or (more commonly) into a single disk file known as an archive. An archive is a file that contains other files plus information about them, such as their filename, owner, timestamps, and access permissions. `tar` does not perform any compression by default.

  To create a disk file `tar` archive, use

  ```
  $ tar -cvf archivenamefilenames
  ```

  where *archivename* will usually have a `.tar` extension. Here the `c` option means create, `v` means verbose (output filenames as they are archived), and `f` means file.To list the contents of a `tar` archive, use

  ```
  $ tar -tvf archivename
  ```

To restore files from a `tar` archive, use

```
$ tar -xvf archivename
```

- `cpio`

  `cpio` is another facility for creating and reading archives. Unlike `tar`, `cpio` doesn't automatically archive the contents of directories, so it's common to combine `cpio` with `find` when creating an archive:

  ```
  $ find . -print -depth | cpio -ov -Htar > archivename
  ```

  This will take all the files in the current directory and the directories below and place them in an archive called *archivename*.The `-depth` option controls the order in which the filenames are produced and is recommended to prevent problems with directory permissions when doing a restore.The `-o` option creates the archive, the `-v` option prints the names of the files archived as they are added and the `-H` option specifies an archive format type (in this case it creates a `tar` archive). Another common archive type is `crc`, a portable format with a checksum for error control.

  To list the contents of a `cpio` archive, use

  ```
  $ cpio -tv < archivename
  ```

  To restore files, use:

  ```
  $ cpio -idv < archivename
  ```

  Here the `-d` option will create directories as necessary. To force `cpio` to extract files on top of files of the same name that already exist (and have the same or later modification time), use the `-u` option.

- `compress`, `gzip`

  `compress` and `gzip` are utilities for compressing and

decompressing individual files (which may be or may not be archive files). To compress files, use:

```
$ compress filename
```

or

```
$ gzip filename
```

In each case, *filename* will be deleted and replaced by a compressed file called *filename*.Z or *filename*.gz. To reverse the compression process, use:

```
$ compress -d filename
```

or

```
$ gzip -d filename
```

3.8 Handling Removable Media (e.g. floppy disks)

UNIX supports tools for accessing removable media such as CDROMs and floppy disks.

- `mount, umount`

  The `mount` command serves to attach the filesystem found on some device to the filesystem tree. Conversely, the `umount` command will detach it again (it is very important to remember to do this when removing the floppy or CDROM). The file `/etc/fstab` contains a list of devices and the points at which they will be attached to the main filesystem:

```
$ cat /etc/fstab↵
/dev/fd0   /mnt/floppy  auto    rw,user,noauto  0 0
/dev/hdc   /mnt/cdrom   iso9660 ro,user,noauto  0 0
```

  In this case, the mount point for the floppy drive is `/mnt/floppy` and the mount point for the CDROM is `/mnt/cdrom`. To access a floppy we can use:

```
$ mount /mnt/floppy↵
$ cd /mnt/floppy↵
$ ls (etc...)
```

To force all changed data to be written back to the floppy and to detach the floppy disk from the filesystem, we use:

```
$ umount /mnt/floppy
```

- `mtools`

  If they are installed, the (non-standard) `mtools` utilities provide a convenient way of accessing DOS-formatted floppies without having to mount and unmount filesystems. You can use DOS-type commands like "`mdir a:`", "`mcopy a:*.* .`", "`mformat a:`", etc. (see the `mtools` manual pages for more details).

(BACK TO COURSE CONTENTS)
