Sheridan

SYST13416
Introduction to Linux
Operating

Working with Files

FILE ATTRIBUTES

Objectives

- File concept: a uniform logical view of information storage
- Special Files for hardware, terminals, and pseudo-devices
- File attributes: name, identifier, type, location, size, protection (permissions), time, date, user identification (ownership)
- File operations: create, write, read, edit, delete, truncate
- File types: executable, object, source code, batch, text, word processor, library, print, view, archive, multimedia
- Protection through permissions

pseu·do [soo-doh]

–adjective

1. *not actually but having the appearance of; pretended; false or spurious; sham.*
2. *almost, approaching, or trying to be.*

File concept: a uniform logical view of information storage

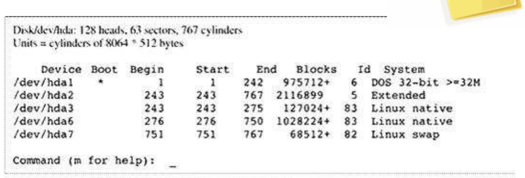
- **Within Unix, a file is any source, with a name, from which data can be read; or any target, with a name, to which data can be written.**
- When you use Linux, the term 'file' refers not only to the repository of data like a disk file, but to any physical device. For example, a keyboard can be accessed as a file (a source of input), as can a monitor (an output target).
- There are also files that have no physical presence whatsoever, but accept input or generate output in order to provide specific services. This is significant because Unix programs can use simple procedures to read from any input source and write to any output target.
- All operating systems are made up of various types of files, including **Text** files, **Special** files, **Binary** files, **Executable** files, **Database** files, **Kernel** files, **E-mail** files, **Media** files (such as **Graphics** files, **Sound** files, and **Movie** files).
- Although each of these files produces different results when executed, they **<u>all share common characteristics and managerial functionality</u>**.
- **Data text files** can be reports, memos, database tables, program, scripts, and other data using by various programs. These are stored generally using ASCII encoding.
- **Media files** are non-text Files and generally contain graphics information. Graphic information, such as images, can include strings of bits representing white and black dots, where each black dot represents a 1, each white dot a 0.
- **Directories** are nothing more than files that contain the hierarchical relationship needed to support its contents and relationships.
- **Executable files** contain pure binary or machine language that the computer can immediately use or execute
- **Special files** are the mechanism used for input and output. Most special files are in /dev and are divided into:
  - Character-oriented devices (serial input/output devices, such as printers, keyboard, etc.),
  - Block-oriented devices (disk, printer, etc.).

Special files

- Special files are **pseudo** files that represent physical devices.

Special files for hardware

- All devices connected to the computer are accessible via special files.
- The first **IDE hard disk** is referred to as **/dev/hda**, the second is **/dev/hdb**, and so on.
- Hard disks are organized into one or more **partitions**, which act as separate devices. The **first partition of the first hard disk** is referred to as **/dev/hda1**. If there is a **second partition**, it si **/dev/hda2**.
- SCSI and SATA hard drives have their own names, the first is **/dev/sda**, the second is **/dev/sdb**, and so on. Again partitions are numbered, so the first partition on the first SCSI or SATA drive is **/dev/sda1**.
- USB flash memory is treated as if it were a removable SCSI disk, and it would be named **/dev/sda1**, or something similar.

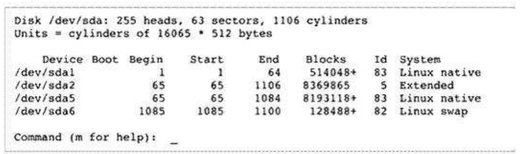


```
Disk/dev/hda: 128 heads, 63 sectors, 767 cylinders
Units = cylinders of 8064 * 512 bytes

    Device Boot  Begin     Start    End   Blocks   Id  System
/dev/hda1    *       1         1    242   975712+   6  DOS 32-bit >=32M
/dev/hda2          243       243    767  2116899    5  Extended
/dev/hda3          243       243    275   127024+  83  Linux native
/dev/hda6          276       276    750  1028224+  83  Linux native
/dev/hda7          751       751    767    68512+  82  Linux swap

Command (m for help): _
```

This partition table is from a Linux system with an IDE drive



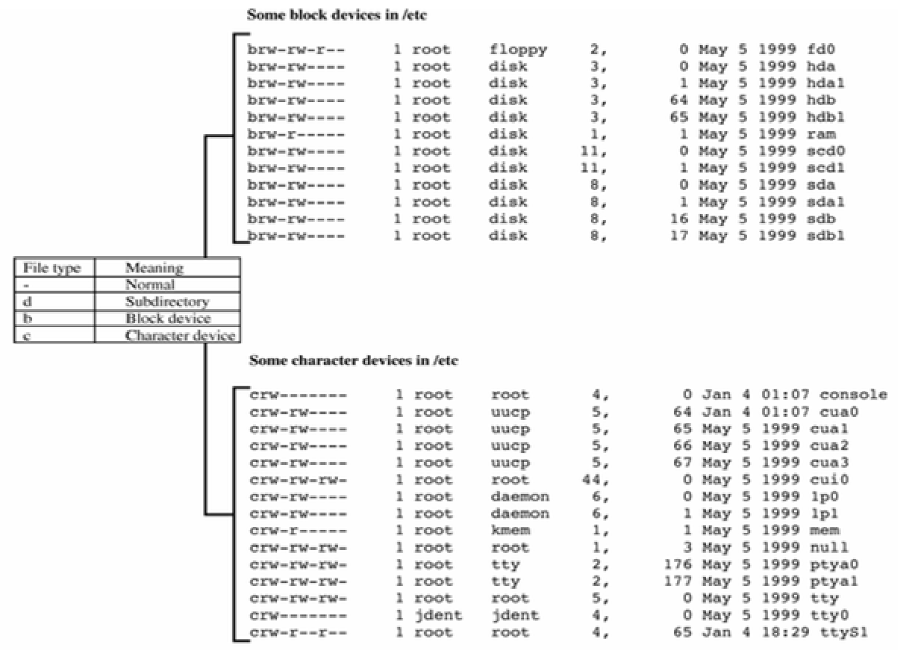
```
Disk /dev/sda: 255 heads, 63 sectors, 1106 cylinders
Units = cylinders of 16065 * 512 bytes

    Device Boot  Begin     Start      End     Blocks    Id  System
/dev/sda1            1         1       64     514048+   83  Linux native
/dev/sda2           65        65     1106    8369865     5  Extended
/dev/sda5           65        65     1084    8193118+   83  Linux native
/dev/sda6         1085      1085     1100     128488+   82  Linux swap

Command (m for help): _
```

This partition table is from a Linux system with a SCSI drive

Hardware	
/dev/fd0	Floppy disk
/dev/hda	IDE Hard disk
/dev/hda1	IDE Hard disk: partition 1
/dev/sda	SCSI Hard disk
/dev/sda1	SCSI Hard disk: partition 1
/dev/sda	USB flash memory card
/dev/lp0	printer
/dev/usb/lp0	USB printer
Terminals	
/dev/tty	Current terminal
/dev/pts/0	Pseudo terminal
Pseudo-devices	
/dev/null	Discard output, input returns nothing (eof)
/dev/zero	Discard output, input returns nulls
/dev/random	Random number generator
/dev/urandom	Random number generator



Some block devices in /etc

```
brw-rw-r--     1 root    floppy     2,         0 May 5 1999 fd0
brw-rw----     1 root    disk       3,         0 May 5 1999 hda
brw-rw----     1 root    disk       3,         1 May 5 1999 hda1
brw-rw----     1 root    disk       3,        64 May 5 1999 hdb
brw-rw----     1 root    disk       3,        65 May 5 1999 hdb1
brw-r-----     1 root    disk       1,         1 May 5 1999 ram
brw-rw----     1 root    disk      11,         0 May 5 1999 scd0
brw-rw----     1 root    disk      11,         1 May 5 1999 scd1
brw-rw----     1 root    disk       8,         0 May 5 1999 sda
brw-rw----     1 root    disk       8,         1 May 5 1999 sda1
brw-rw----     1 root    disk       8,        16 May 5 1999 sdb
brw-rw----     1 root    disk       8,        17 May 5 1999 sdb1
```

File type	Meaning
-	Normal
d	Subdirectory
b	Block device
c	Character device

Some character devices in /etc

```
crw-------     1 root    root       4,         0 Jan 4 01:07 console
crw-rw----     1 root    uucp       5,        64 Jan 4 01:07 cua0
crw-rw----     1 root    uucp       5,        65 May 5 1999 cua1
crw-rw----     1 root    uucp       5,        66 May 5 1999 cua2
crw-rw----     1 root    uucp       5,        67 May 5 1999 cua3
crw-rw-rw-     1 root    root      44,         0 May 5 1999 cui0
crw-rw----     1 root    daemon     6,         0 May 5 1999 lp0
crw-rw----     1 root    daemon     6,         1 May 5 1999 lp1
crw-r-----     1 root    kmem       1,         1 May 5 1999 mem
crw-rw-rw-     1 root    root       1,         3 May 5 1999 null
crw-rw-rw-     1 root    tty        2,       176 May 5 1999 ptya0
crw-rw-rw-     1 root    tty        2,       177 May 5 1999 ptya1
crw-rw-rw-     1 root    root       5,         0 May 5 1999 tty
crw-------     1 jdent   jdent      4,         0 May 5 1999 tty0
crw-r--r--     1 root    root       4,        65 Jan 4 18:29 ttyS1
```

Device Standards

Integrated Drive Electronics (IDE) or Parallel ATA (PATA):
is an interface standard for the connection of storage devices, such as hard disks, solid-state drivers, and CD-ROM drives in computers.

Serial ATA (SATA)
newer version (2007) of the PATA interface.

Small Computer System Interface (SCSI)
(pronounced scuzzy), is a set of standards for physically connecting and transferring data between computers and peripheral devices.

Special files for terminals

- In the past, terminals were separate physical devices that were connected to a host computer. Such terminals were represented by special files named **/dev/tty1**, dev/tty2, and so on. (**TTY** is synonym for terminal, very first terminals were **T**ele**TY**pe machines.)
- The /dev/tty1 naming convention is still used today for terminals that act like hardware devices. Your keyboard and monitor (console) act as a built-in text-based terminal. When you use a virtual console within a desktop environment, it too acts like an actual terminal. Linux supports six such consoles, represented by the special files /dev/tty1 through /dev/tty6.
- Everything is different when you use a GUI to run a terminal emulation program within a window. Because there isn't an actual terminal, Unix creates what we call **a pseudo terminal (PTY)** to simulate a terminal. There are two different systems for creating pseudo terminals. If your version of Unix uses the first system, the special files will have names like **/dev/ttyp0**. If you use the other system, the names will be **/dev/pts/1**, and so on.

Note:

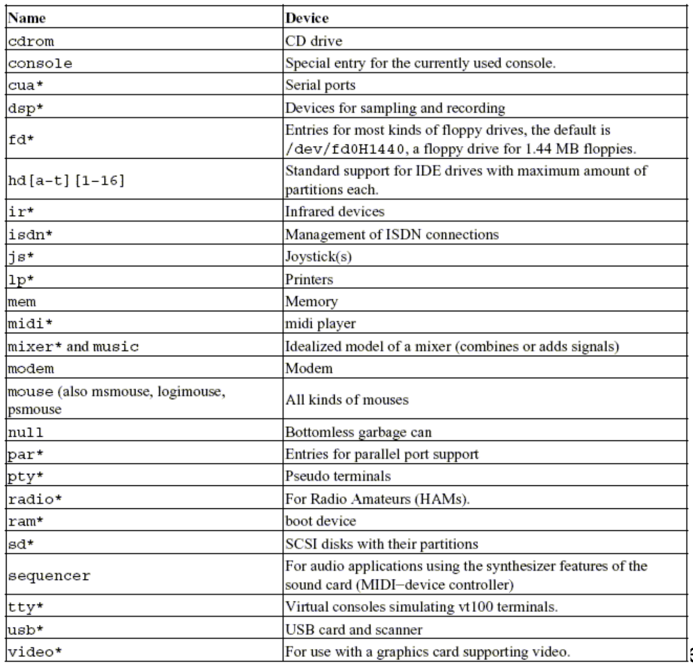
The terminology is a bit confusing: the null file returns nothing, while the zero file returns nulls. Such is life.

As strange as it seems, a constant source of null characters can be useful. For instance, for security reasons, it is often necessary to wipe out the contents of a file or an entire disk. In such cases, you can overwrite the existing data with nulls simply by copying as many bytes as necessary from /dev/zero to the output target.

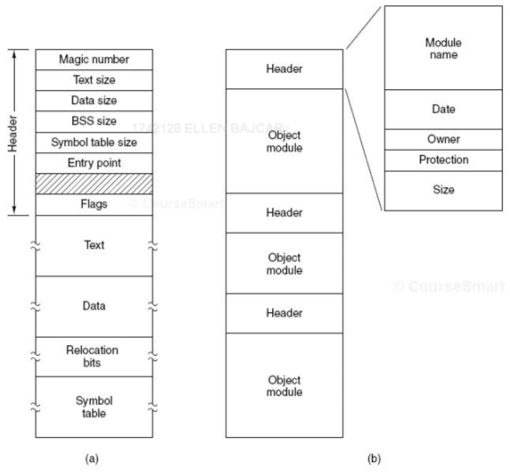
Hardware	
/dev/fd0	Floppy disk
/dev/hda	IDE Hard disk
/dev/hda1	IDE Hard disk: partition 1
/dev/sda	SCSI Hard disk
/dev/sda1	SCSI Hard disk: partition 1
/dev/sda	USB flash memory card
/dev/lp0	printer
/dev/usb/lp0	USB printer
Terminals	
/dev/tty	Current terminal
/dev/pts/0	Pseudo terminal
Pseudo-devices	
/dev/null	Discard output, input returns nothing (eof)
/dev/zero	Discard output, input returns nulls
/dev/random	Random number generator
/dev/urandom	Random number generator

Special files for pseudo-devices

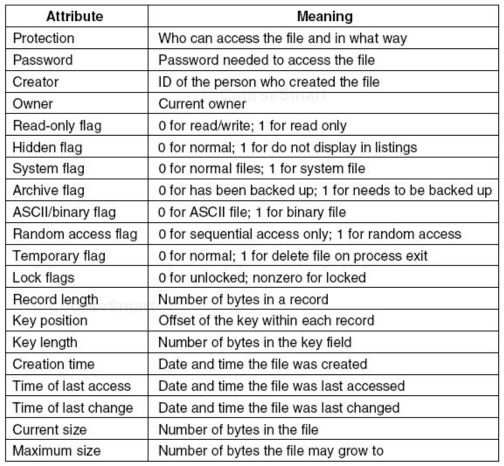
- A **pseudo-device** is a file that acts as an input source or output target, but does not correspond to an actual device, either real or emulated.
- The two most useful pseudo-devices are the NULL FILE and the ZERO FILE.
- The null file is **/dev/null**
- The zero file is **/dev/zero**
- Any output that is written to these devices is thrown away. For this reason, these files are sometimes referred to whimsically as “bit buckets”. Both work the same. The only difference is what happens when they are used for input.
- When a program reads from /dev/null, no matter how many bytes of input are requested, the result is always an **eof** signal. In other words, reading from **/dev/null returns nothing**.
- When a program reads from /dev/zero, the file **generates as many bytes as are required**. However, they all have the value 0 (the number zero). In Unix, this value is considered to be the null character or simply, a **NULL**.



Name	Device
cdrom	CD drive
console	Special entry for the currently used console.
cua*	Serial ports
dsp*	Devices for sampling and recording
fd*	Entries for most kinds of floppy drives, the default is /dev/fd0H1440, a floppy drive for 1.44 MB floppies.
hd[a-t][1-16]	Standard support for IDE drives with maximum amount of partitions each.
ir*	Infrared devices
isdn*	Management of ISDN connections
js*	Joystick(s)
lp*	Printers
mem	Memory
midi*	midi player
mixer* and music	Idealized model of a mixer (combines or adds signals)
modem	Modem
mouse (also msmouse, logimouse, psmouse	All kinds of mouses
null	Bottomless garbage can
par*	Entries for parallel port support
pty*	Pseudo terminals
radio*	For Radio Amateurs (HAMs).
ram*	boot device
sd*	SCSI disks with their partitions
sequencer	For audio applications using the synthesizer features of the sound card (MIDI–device controller)
tty*	Virtual consoles simulating vt100 terminals.
usb*	USB card and scanner
video*	For use with a graphics card supporting video.



(a) An executable file. (b) An archive.



Attribute	Meaning
Protection	Who can access the file and in what way
Password	Password needed to access the file
Creator	ID of the person who created the file
Owner	Current owner
Read-only flag	0 for read/write; 1 for read only
Hidden flag	0 for normal; 1 for do not display in listings
System flag	0 for normal files; 1 for system file
Archive flag	0 for has been backed up; 1 for needs to be backed up
ASCII/binary flag	0 for ASCII file; 1 for binary file
Random access flag	0 for sequential access only; 1 for random access
Temporary flag	0 for normal; 1 for delete file on process exit
Lock flags	0 for unlocked; nonzero for locked
Record length	Number of bytes in a record
Key position	Offset of the key within each record
Key length	Number of bytes in the key field
Creation time	Date and time the file was created
Time of last access	Date and time the file was last accessed
Time of last change	Date and time the file was last changed
Current size	Number of bytes in the file
Maximum size	Number of bytes the file may grow to

Some possible file attributes.

File Operations

- Files exist to store data and allow it to be retrieved later. Different systems provide different operations to allow storage and retrieval. This list is the most common **system calls** relating to files:
  1. **Create**. The file is created with no data. The purpose of the call is to announce that the file is coming and to set some of the attributes.
  2. **Delete**. When the file is no longer needed, it has to be deleted to free up disk space. There is always a system call for this purpose.
  3. **Open**. Before using a file, a process must open it. The purpose of the open call is to allow the system to fetch the attributes and list of disk addresses into main memory for rapid access on later calls.
  4. **Close**. When all the accesses are finished, the attributes and disk addresses are no longer needed, so the file should be closed to free up internal table space. Many systems encourage this by imposing a maximum number of open files on processes. A disk is written in blocks, and closing a file forces writing of the file's last block, even when the block may not be entirely full.
  5. **Read**. Data are read from file. Usually, the bytes come from the current position. The caller must specify how many data are needed and must also provide a buffer to put them in.
  6. **Write**. Data are written to the file again, usually at the current position. If the current position is the end of the file, the file's size increases. If the current position is in the middle of the file, existing data are overwritten an lost forever.
  7. **Append**. This call is a restricted form of write. It can only add data to the end of the file. Systems that provide a minimal set of system calls do not generally have append, but many systems provide multiple ways of doing the same thing, and these systems sometimes have append.
  8. **Seek**. For random access files, a method is needed to specify from where to take the data. Once common approach is a system call, seek, that repositions the file pointer to a specific place in the file. After this call has completed, data can be read from, or written to, that position.
  9. **Get attributes**. Processes often need to read file attributes to do their work. For example, the Unix make program is commonly used to manage software development projects consisting of many source files. When make is called, it examines the modification times of all the source and object files and arranges for the minimum number of compilations reured to bring everything up to date. To do its job, it must look at the attributes, namely, the modification times.
  10. **Set attributes**. Some of the attributes are user settable and can be changed after the file has been created. This system call makes that possible. The protection mode information is an obvious example. Most of the flags also fall in this category.
  11. **Rename**. It frequently happens that a user needs to change the name of an existing file. This system call makes that possible. It is not always strictly necessary, because the file can usually be copied to a new file with the new name, and the old file then deleted.

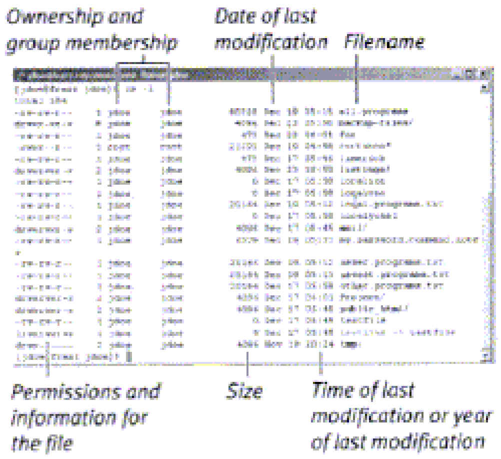
Symbol	Meaning
–	Regular file
d	Directory
l	Link
c	Special file
s	Socket
p	Named pipe

File Types Summarized

- As you explore Linux, you will encounter various file types.

  ```
  ls -F
  ```

  indicates four common types:
  - Regular file: no trailing character
  - Directory: / (example: `homework/`)
  - Executable: * (example: `/usr/bin/ls*`)
  - Symbolic link: @ (example: A4@)
- For more types, see man ls under the –F option.


Ownership and group membership
Date of last modification
Filename
Permissions and information for the file
Size
Time of last modification or year of last modification

Ownership and Group Membership

- When you obtain a Linux account, you obtain a username and a password. The administrator also provides you with a login shell and a home directory. Everything you do through your account becomes associated with your username. For example, every file you create, you become its owner. Every command you run, you are the owner of that process. You also become a member of at least one group. This makes the administration of the system easier because groups, such as a department, or type of employee, share the same set abilities and access to computing resources.

Date

- The date command displays the current time and date. There are three (3) dates associated with each file: creation date, last modification date, and last access date. The long listing of directory displays the last modification date.
- You can change the last modification date and last access date using the touch command.
- To view the last access date instead of last modification date:

  ```
  ls -ul
  ```

Size

- To show sizes in a "human-readable" format, use the h option:

```
ls -h
```

Inode

- See Handout on ln command.
- To show i-node number with the ls command, use the i option:

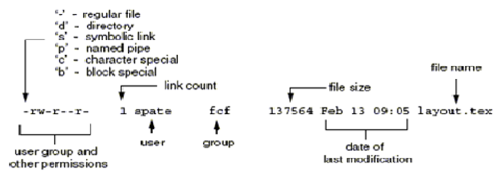
```
ls -i
```

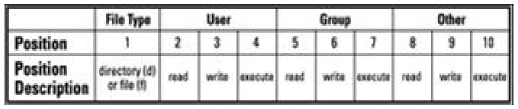
File Names

- Keeping in mind the Linux naming strategy (case sensitivity, name length, and use of alphanumeric characters), the Linux shells provides file-matching meta-characters (or wildchards), which are characters that represent patterns of text in filenames:
- **Asterisk *:** represents 0 or more instances of any character. If the filename begins with a dot (.), the * willnot match the name unless you say .*
- **Question mark ?:** matches exactly one character
- **Square brackets []:** select any character between the brackets. You can use a hyphen (-) to express a continuous ranges, such as a-z, D-S, or 1-5. Backward ranges are unpredictable. If you would like to provide a list of characters to excluse, use the caret(^) as the first character in the list (i.e., [^a-z] excludes all lowercase letters)
- **Backslash \:**interpret the next character as a literal character. For example,

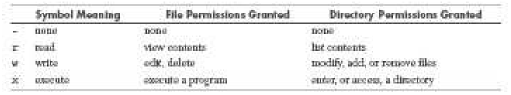
```
cat a\ b
```

will view a file called "a b"





	File Type	User			Group			Other		
Position	1	2	3	4	5	6	7	8	9	10
Position Description	directory (d) or file (f)	read	write	execute	read	write	execute	read	write	execute



	Symbol Meaning	File Permissions Granted	Directory Permissions Granted
-	none	none	none
r	read	view contents	list contents
w	write	edit, delete	modify, add, or remove files
x	execute	execute a program	enter, or access, a directory

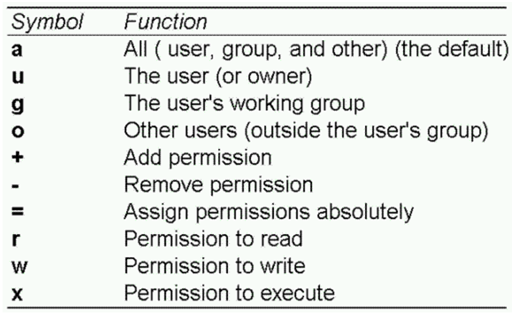
Protecting Your Files

Changing Permissions with chmod

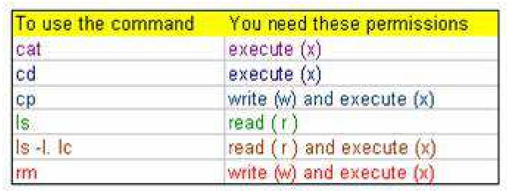
- When everyone can read, write, and execute an ordinary file, the permissions look like this:
  - r w x r w x r w x
- When everyone can read, write, and execute a directory, the permissions are:
  d r w x r w x r w x
- When everyone is no rights to read, write, or execute an ordinary file, the permissions look like this:
  - - - - - - - - - -

Symbolic Representation

- To change the protection mode on all files in your current directory to allow the following access: read, write, and execute permission for the owner; read permissions only for the other users of the system; read permissions only for the members of your own group:
  ```
  chmod u+rwx,go+r-wx  *
  ```
- To change the protection mode on file called alice.txt to take away read permission from the owner:
  ```
  chmod u-r alice.txt
  ```
- To change the protection mode on alice.txt to restore read permission and take away write permission from the owner:
  ```
  chmod u+r-w alice.txt
  ```



Symbol	Function
a	All (user, group, and other) (the default)
u	The user (or owner)
g	The user's working group
o	Other users (outside the user's group)
+	Add permission
-	Remove permission
=	Assign permissions absolutely
r	Permission to read
w	Permission to write
x	Permission to execute



To use the command	You need these permissions
cat	execute (x)
cd	execute (x)
cp	write (w) and execute (x)
ls	read (r)
ls -l. lc	read (r) and execute (x)
rm	write (w) and execute (x)

Directory permissions:

Octal Representation

- These permissions are usually referred to by their octal numbers, particularly when you want to change permissions: The octals 4,2,1,0 are used as follows:

Read = 4 Write = 2 Execute = 1

rwx	(Read, write and execute)	= 7 (4+2+1)
rw-	(Read and write only)	= 6 (4+2)
r-x	(Read and execute)	= 5 (4+1)
r--	(Read only)	= 4
-wx	(Write and execute)	= 3 (2+1)
-w-	(Write only)	= 2
--x	(Execute only)	= 1
---	(No privileges)	= 0

- To change permissions so that the group and others have have read rights only and the user has read and write privileges ...

```
chmod 644 tfile
- r w - r - - r - - tfile
  ↓     ↓     ↓
user group others
  6     4     4
```

- To change permissions so that you, the owner, have all permissions, the group has read and write and others have no rights ...

```
chmod 760 tfile
- r w x r w - - - - tfile
  ↓     ↓     ↓
user group others
  7     6      0
```

FILE MANAGEMENT

Objectives

- Copy a file: **cp**
- Copy files to a different directory: **cp**
- Copy a directory to a different directory: **cp –r**
- Delete a file: **rm**
- Move a file: **mv**
- Rename a file or a directory: **mv**
- Command: **find** - search for files in a directory hierarchy.
- Command: **locate**
- Compress and uncompress files with commands **gzip, zip, tar,** and **unzip**

Copy a file: cp

Syntax:

```
cp [-ip] file1 file2
```

where file1 is the name of an existing file, and file2 is the name of the destination file.

Examples:

```
cp data extra
```

copies existing file data to a new file called extra

```
cp /etc/group ~/group
```

copies the system file group in /etc sub-directory to your home directory, under the same name

```
cp –i data extra
```

The option –i tells the cp command to ask you for permission to overwrite the existing file, if the file called extra already exists

The option –p tells the cp command to preserve the destination file the same modification time, access time, and permissions as the source file.

Copy files to a different directory: cp

Syntax:

```
cp [-ip] files directory
```

where file1 is the name of an existing file, and file2 is the name of the destination file.

Examples:

```
cp data backups
```

copies existing file data to the directory named backups

```
cp data1 data2 data3 backups
```

copies existing files, data1, data2, and data3, to the directory named backups

```
cp ../../papers/adventure .
```

copies existing file called adventure to your current working directory (**note the dot as the second argument!**)

Copy a directory to a different directory: cp -r

Syntax:

```
cp –r [-ip] directory1 directory2
```

where directory1 is the name of an existing directory, and directory2 is the name of the destination directory

Examples:

```
cp –r essays backups
```

copies the directory itself including all its files and subdirectories

```
cp documents/* backups
```

copies all the files in a directory named documents to a directory named backups

Delete a File: rm

Syntax:

```
rm [-fir] file
```

where file is the name of a file you want to delete

Examples:

```
rm data
rm ~/essay
rm bin/somefile
rm data[123]
rm *
rm data.[beo]*
```

The last example removes files data.backup, data.extra, data.old, but not data.important.

Move a file: mv

Syntax:

```
mv [-if] files directory
```

where files is the name of one or more existing files, and directory is the name of the existing destination directory

Examples:

```
mv data archive
mv data1 data2 data3 archive
mv data[123] archive
mv –i data archive
```

The option –i tells the mv command to ask you for permission to replace a file that already exists

The option –f tells the mv command to replace a file without checking with you. The –f option will override the –i option. Use with care.

Rename a file or a directory: mv

Syntax:

```
Mv [-if] oldname newname
```

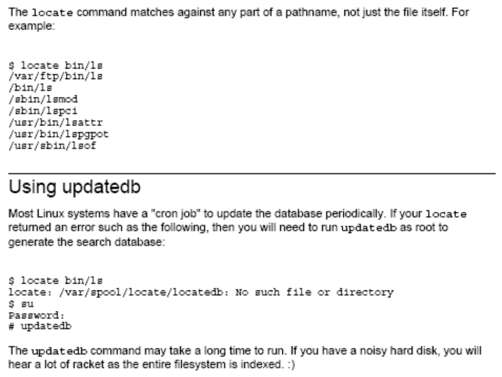
where oldname is the name of an existing file or directory, and newname is the new name.

Examples:

```
mv unimportant important
mv incomplete archive/complete
mv old new
```

Note

atlas.sheridanc.on.ca
does not have the locate command installed.



The `locate` command matches against any part of a pathname, not just the file itself. For example:

```
$ locate bin/ls
/var/ftp/bin/ls
/bin/ls
/sbin/lsmod
/sbin/lspci
/usr/bin/lsattr
/usr/bin/lspgpot
/usr/sbin/lsof
```

Using updatedb

Most Linux systems have a "cron job" to update the database periodically. If your `locate` returned an error such as the following, then you will need to run `updatedb` as root to generate the search database:

```
$ locate bin/ls
locate: /var/spool/locate/locatedb: No such file or directory
$ su
Password:
# updatedb
```

The `updatedb` command may take a long time to run. If you have a noisy hard disk, you will hear a lot of racket as the entire filesystem is indexed. :)

Finding files with find and locate

- ***Finding forgotten files with <u>find</u> command***
  Sometimes finding a file requires more than cursing at your computer or listing directory contents with ls. Instead, you can use the find command which lets you search in dozens of ways, including through the entire directory tree or through directories you specify.
- ***Locating lost files with <u>locate</u> command***
  If you're looking for a system file, a program or file that is part of the Unix system itself, rather than one of your own personal files in your home directory, try locate command to find it> You'll get more results than you can handle, but it's a quick and easy way to locate system files. It is not available on all Unix systems, but it is worth a try at any rate.
- Examine the following manual pages and note the description and what the options used are in these exercises: find and locate
- **find**: The find command allows you to search for files.
  ```
  find  . –type f
  ```
- **locate**: The locate command allows you to search for system files.
  ```
  locate useradd
  ```
- Find the file called music.data you created earlier:
  ```
  find  ~  -name  music.data
  ```
  to execute the following set of commands.
- Using find with xargs allows you to search a combination. By default xargs always tacks the list of filenames to the end of the specified command. (xargs may not be on every system.)
  ```
  find  .  –type  f  |  xargs  grep  –i  Ontario
  ```

Zip and unzip files

- For a quick help on **zip**, enter it without any options or arguments.
- To recursively zip all files from someDir:
  ```
  zip –r stuff.zip someDir
  ```
- To **unzip** a zip file:
  ```
  unzip stuff.zip
  ```
- The GNU's zip program is called **gzip**, which is very popular. The command **gunzip** is the uncompress utility with gzip.
- If you see a file with a tgz extension, you likely have a tar file that has been compressed with gzip (i.e., file.tar.gz). You have two choices to unzip the file:
- Stardard tar: use
  ```
  gunzip < file.tar.gz | tar xvf –
  ```
  or
  ```
  gunzip < file.tgz | tar xvf –
  ```
- GNU's tar: use
  ```
  tar xvzf file.tar.gz
  ```
  or
  ```
  tar xvzf file.tgz
  ```
- The **tar** (or tape archiver) packs files and directories together into a single file called a tar file. Although it refers to storage on magnetic tape, it is commonly used to archive file for either storage on most media or sharing over the internet. Often you can download applications as **tarballs**.

Contents