The User's Guide to Academic Computing and the Internet

RYERSON UNIVERSITY 2006-2007

Toronto, Ontario, Canada

RYERSON UNIVERSITY

UNIX Commands

Toronto, Ontario

Ryerson's central academic computing servers (Malthus, Turing, and Snapper) use the Solaris operating system, a variant of UNIX. This section explains how to work with the command line on all these systems.

WORKING WITH THE COMMAND LINE

After you log in to a UNIX system, you are prompted to type into a "command line". -- For example:

```
[/home/m3a/name]>sort -n mylist.txt > sortedlist.txt
```
[Press ENTER to make command take effect.]

runs the sort program. This command numerically sorts each line in the **mylist.txt** file and saves the sorted lines of text in the file **sortedlist.txt**. A command line consists of one or more words separated by spaces or tabs. The first word in the command line is the command. Any other words are arguments, also called parameters. Each word in this example is described below:

Word	Description
`sort`	The command of this line is a UNIX program called **sort**. The program exists on the UNIX machine as an executable file. Information on **sort** is available by **man sort** at the UNIX prompt.
`-n`	An option gives the command additional information for what to do. Without any options, each line is sorted in alphabetical order. The **-n** option instructs **sort** to sort the lines numerically. The man page for UNIX programs describes all the options available for each program.
`mylist.txt`	A file named **mylist.txt** will be sorted in this case. This is a text file. A text file contains only printable (or readable) characters. Since we are sorting numerically, we assume each line of the file starts with a number, although this may not always be the case. See the **sort** man page [the manual (help) page for the sort command] for what sort does in this case; or try it.
`>`	UNIX command lines may contain numerous special characters. The *greater than* symbol is used to redirect the standard output of the **sort** program to another file. Normally [*i.e.* without >], a program like **sort** would send its output to something called standard output which usually ends up on the terminal screen. Using > redirects this standard output to the file **sortedlist.txt**.
`sortedlist.txt`	The file that the sort list is being redirected to will be a text file also, because it contains only readable/printable characters. The file will be created if it doesn't exist. If it does exist, the results depend on how the environment is set up on your system. On most systems, existing contents will be deleted before the sort lines are written to the file. If the "noclobber" option is set, you may have to remove or rename your **sortedlist.txt** file before running this command line.

GENERAL UNIX CONVENTIONS

UNIX treats upper and lower case letters as distinct. For example: the file **FRED.TXT** is not the same as **fred.txt**.

UNIX commands are designed to be concise but some are rather cryptic. For example, **ls** is the command for listing files. Many DOS commands work in place of UNIX commands [are named the same in Unix].

Type all commands in lower case. For example: type **more filename** to display the contents of a file.

Like DOS, UNIX allows you to substitute wild card characters such as ***** and **?** when specifying the names of files and directories. The command **ls *.c** (for example) lists all the files in the current directory with the extension '**c**'. [A suffix which consists of a period followed by a letter (or letters) is called an extension.]

TYPES OF FILES

UNIX supports three different types of files: ordinary files, directories, and special files. Ordinary files are data and document files; executable files (such as a C program); or output from any application. Directory files are information files that help access other files. In most cases a directory helps organize other files in logical groups. A directory can contain all three types of files -- the same as DOS subdirectories do. Special files represent physical devices like keyboards, monitors, or printers [«in Unix, everything is a file»].

THERE ARE TWO TYPES OF ORDINARY FILES:

1. **Text Files (ASCII):** Contain printable characters such as letters, numbers or punctuation and are read, edited, displayed or printed easily.
2. **Binary Files:** Binary files in general are produced by compilers or by applications.

ATTRIBUTES AND FLAGS

The command explanations elsewhere in this guide use a number of foreign terms. These terms include "attributes " and "flags ". A file's attributes control who can use a file and if users can manipulate it. **Flags**, also called **Options**, are control characters added to [after] a command to modify its use [behavior].

SYSTEM PROMPT

In all examples, the system prompt is shown in the following format: [namely at Ryerson University]

```
[/home/m3a/username]>
```

- **username** represents your home directory and is usually your account name [log-in name].
- **m3a** (or similar designation) specifies the disk drive location within the directory hierarchy.

LISTING FILE INFORMATION

To list the contents of your home directory use the **ls** command.

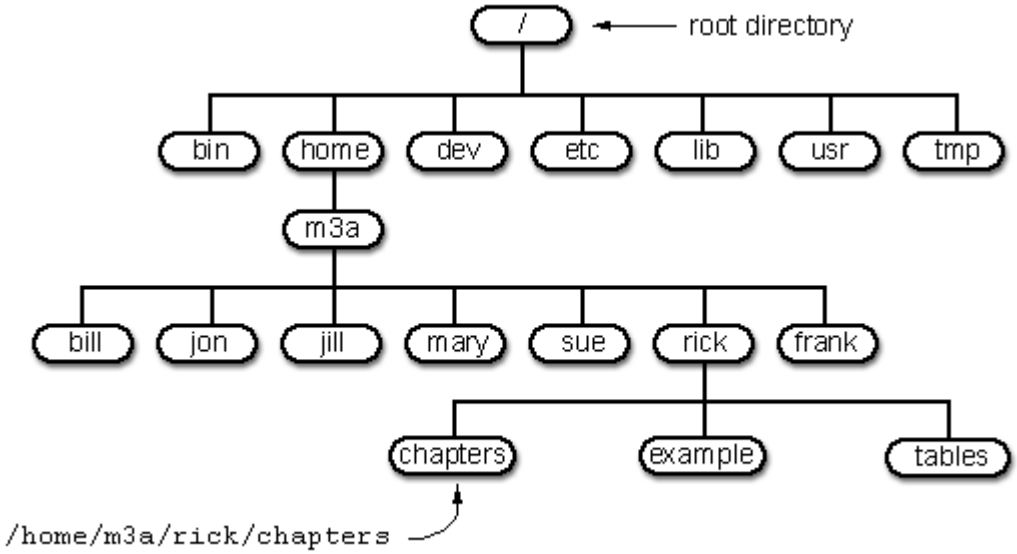
Cmd Name:	**ls**
Purpose:	Displays the contents of a directory
Description:	The **ls** command lists the files in the current directory location, or any specified directory location, in alphabetical order. The wild cards * (one or more characters) and ? (single character) are used. The default setting for **ls** is wide format, with files listed across the screen in four columns. Flags include **-l** (all file information); **-a** (show hidden files); **ls -lt** (last date modified) and others. More information is available using the command **man ls**.
Variations:	**ls -l, ls -la, ls -l *.txt, ls Lt, ls -m, ls -lr, ls -bt**

Examples:	`[/home/m3a/frank]> ls`
	
	(displays across the screen in four columns)
	
	`a.out  context.txt  log.fnk  notes`
	`aleck.f.c  files.list  mkk.txt  xalltheway`
	
	--- 	
	`[/home/m3a/frank]> ls -l`
	
	(displays full file information)
	
	`-rwxr-xr-x  1 frank whole  512 Mar  4 10:36 a.out`
	`-rw-r--r--  1 frank whole  5697 Apr 30 08:54 aleck.f.c`
	`-rw-r--r--  1 frank whole  3752 Mar 15 17:01 context.txt`
	`-rw-r--r--  1 frank whole  324 Jan  6 15:05 files.list`
	`-rwxr-xr-x  1 frank whole  3002 Jan  6 15:10 log.fnk`
	`-rw-r--r--  1 frank whole  987 APR 10 09:45 mkk.txt`
	`drwxr-xr-x  2 frank whole  512 Jan  6 08:01 notes`
	`drwx--x--x  3 frank whole  512 Jan  6 08:45 xalltheway`

See Also:	File permissions (**chmod**)

DIRECTORY STRUCTURE

UNIX directories follow a hierarchical tree structure similar to DOS. The diagram below demonstrates a simple UNIX directory structure. The symbol / represents the root or topmost directory. All other directories have names. This differs from DOS where a drive name is the root of the file system on that drive. It also differs from the more recent Window-centric operating systems, where the "**My Computer**" icon represents the top level of the directory tree and system drives are icons within the "My Computer" folder.



Tree structured systems refer to the root (/) directory as the ultimate **parent**, and all other directories as its **children**. In the above example, the directory **rick** is the parent of **chapters**, but **rick** is a child of **m3a**.

Absolute and Relative Paths

There are two types of paths within the tree structure: absolute paths and relative paths.

An absolute path is the full path name from the root (/) directory to the directory or file required. Using the tree diagram above, the absolute path to the directory **chapters** would be:

/home/m3a/rick/chapters ("/" for root , followed by "**home/**", "**m3a/**", "**rick/**" and finally **chapters**)

An absolute path must always contain the full path name from root (ultimate parent) to last **child** or **grandchild** directory.

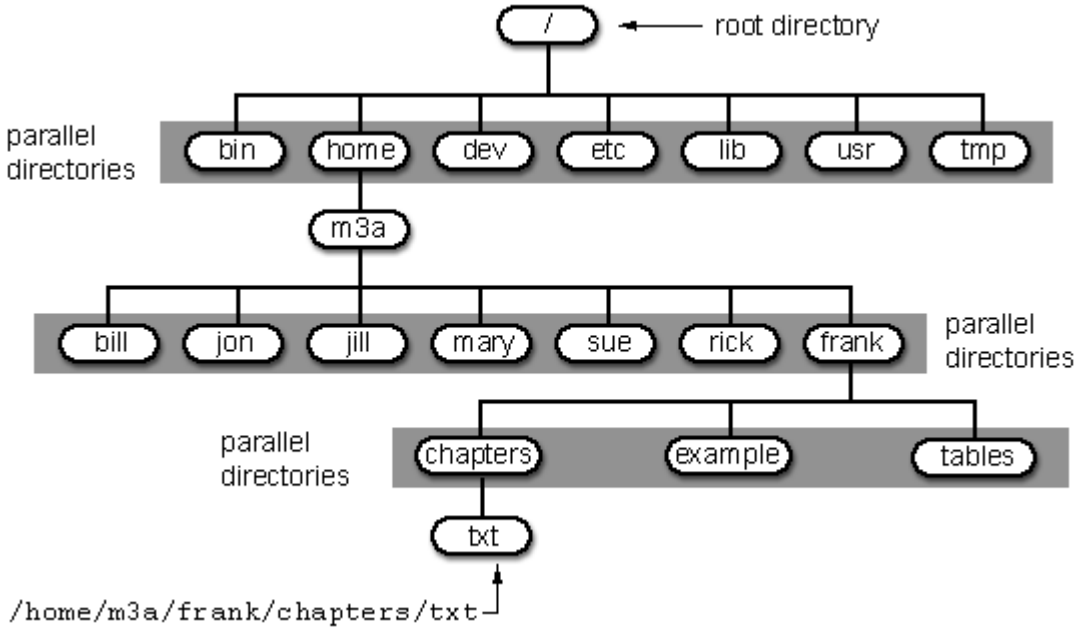
Relative paths are paths that start from your current or home directory. In the example above, the directory **rick** is the home directory of user **rick**. The path from **rick** to **chapters** would be **~/chapters** using the tilde substitution. [Tilde ~ represents (= a short-cut for) the home directory of the user who is logged in.]

A relative path from **rick** to the directory **jill** would be:

../jill (**..** used to go up one level from the current directory) [and then down to the **/jill** directory.]

The figure below shows how path designation specifies files within "parallel" level directories like the home directories of the users [who log in under the user-names of] **bill**, **jon**, **jill**, **mary**, **sue**, **rick** and **frank**.

~ [tilde] stands for your home directory and **..** stands for its parent directory [two periods or dots].



Home Directory with its Subdirectories

Your home directory is your personal file storage area. You are the only one with permission to create and execute files within your directory. When you log in , you start from your home directory. Within your own directory structure , you control how (or whether) other users can access your files. By default, only you can read and write files in your home directory. For more information, see section dealing with File Permissions.

Changing Directories

To change directories use the **cd** command. This command is similar to the DOS **cd** command.

Cmd Name:	**cd**
Purpose:	Changing your current location within the directory structure.
Description:	Using the **cd** (change directory) command to move from one directory location to another. You can use either absolute path names or relative path names to specify directories. Use the **cd** command by itself to return from any location to your home directory.
Variations:	**cd , cd .., cd /, cd name, cd ~/name, cd ../name,** **cd /dname1/dname2/dname3/dname4**
Examples:	`[/home/m3a/frank]>` *(result)* `[/home/m3a/frank]> cd ..` *(go up one level from home)* `[/home/m3a]>` *(result)* `[/home/m3a/frank]> cd /` *(go to root level)*

	`[/]>` *(result from cd /)* `[/home/m3a/frank]>` **`cd chapters`** *(go down to chapters directory)* `[/home/m3a/frank/chapters]>` *(result)* `[/home/m3a/frank]>` **`cd ~/chapters/txt`** *(go down from home to txt directory)* `[/home/m3a/frank/chapters/txt]>` *(result)* `[/home/m3a/frank]>` **`cd /usr/local/bin`** *(go to bin using absolute path)* `[/usr/local/bin]>` *(result)*
See Also:	Make Directory (**mkdir**), Remove Directory (**rmdir**)

Creating Directories

You can make directories within your home directory to any depth. You are responsible for managing your own directory structure, so keep it simple. To create a directory, use the **mkdir** command.

Cmd Name:	**mkdir**
Purpose:	Create a new directory.
Description:	The **mkdir** command creates a directory within your home directory and within any directory you have permission to modify. **-p** flag creates intermediate [see below] directories from your current directory. The **md** (DOS) command works without flags.
Variations:	**mkdir, mkdir -p, md**
Examples:	`[/home/m3a/Jill]>` **`mkdir chapters`** *(creates **/chapters** directory)* `[/home/m3a/Jill]>` **`mkdir ~/txt`** *(creates **/txt** directory from home directory)* `[/home/m3a/Jill]>` **`mkdir -p ~/snds/exec`** [**snds** is intermediate to **exec**.] *(creates [two] new directories **/snds** and directory **/exec** within **snds** directory)*

Removing Directories

To remove a directory, use the **rmdir** command.

Cmd Name:	**rmdir**
Purpose:	To remove [erase, delete] an existing directory.
Description:	The **rmdir** command removes a directory within your home directory. The directory must be empty before it can be removed. Use the **ls -la** (list all and hidden files) to check if a directory is empty. The DOS-style command **rd** removes a single directory.
Variations:	**rmdir, rd** (DOS-style command [same name *i.e.* in DOS])

Examples:	`[/home/m3a/Jill]> rmdir chapters`
	*(remove empty directory /**chapters**)*
	--- 	
	`[/home/m3a/Jill]> rmdir ~/txt`
	*(remove empty directory /**txt** from home directory)*

See Also:	**rm** (remove command)

Exploring System Directories

You are free to explore UNIX directories using absolute path names, if file permissions permit. Absolute path names mean that a directory location is being specified from the root (/). The only time a relative path name is used is when moving from a directory one level down (**cd dir_name**), or any number of levels up (**cd ..** -or- **cd ../../abc**). [Note use of ../../ for «grandparent directory.»] You may look at files using the **cat** command (or **cat <filename> | more**) if permissions allow (general read permissions for others).

MANIPULATING FILES

This section explains how to use UNIX commands to copy [below], move [p.9], delete [«remove,» p.10], and browse [read] files [p.11], and how to change file permissions [p.13].

Useful commands that are not documented in this section include: **cut** (cut information from file), **paste** (combine file information), **grep** (search files for character strings), **sort** (sort file information in specified order) and **uniq** (find duplicate lines in a file). -- For full information about these commands check the **man** pages.

Copying Files (cp)

You can copy files from any location within your home directory structure to any other location. You can also copy files within the same directory if you give the copy a new name. If permissions are granted, files can be copied from other users' directories (see Changing File Permissions). You cannot copy a file into another user's home directory. If you want to send a file to another user, use **email**. Once a file is copied from another user's directory (if you'd received read permission), you have full read and write privileges.

The syntax of the copy command reads:

cp [- flag] filename1 filename2

Directory and path are added as required [see examples]. Filename1 is the file being copied; filename2, the name of the copy. The following command summary information explains more about the **cp** command:

Cmd Name:	**cp**
Purpose:	Make a copy of an existing file
Description:	Use the **cp** command to copy a file. The source file and target file must be specified. Directory path names can be included as required. Absolute path name from root (/) must be used when copying files from other directories. Flags that are useful include **-i**, which prompts before overwriting existing files, and **-p,** which preserves original modification time of the source [original] file.
Variations:	**cp**, **cp -i**, **copy** (DOS command)

Examples:	`[/home/m3a/geoff]> cp sample.txt apple.txt`
	*(copy **sample.txt** to make **apple.txt**)*
	
	`[/home/m3a/geoff]> cp sample.txt ~/chapters/sample.txt`
	(copy file to another directory)
	
	`[/home/m3a/geoff]> cp /home/m3a/jenny/saslog saslog.jenny`
	(copy file from another directory to current directory)
	
	`[/home/m3a/geoff]> cp -i /home/m3a/jenny/saslog saslog.jenny`
	
	`cp: overwrite saslog.jenny?`
	(copy file with overwrite protection)
	
	`[/home/m3a/geoff]> cp ../jenny/letter .`
	(copy a file from another directory to the current directory, and give it the same name)

See Also:	File permissions (**chmod**), DOS command (**copy**), Link (**ln**), (**backup**) and (**restore**).

Moving and Renaming Files (mv)

Use the **mv** command to move or rename a file.

The syntax of the move command is:

mv [-flag] file1 directory/file2

To rename "file1" to "file2", type:

mv file1 file2

If the file permissions allow it, you can rename and freely move files anywhere within the system. You cannot move a file to another user's directory [to *i.e.* the home directory of another user], and it is impossible to move a file from another directory (or to rename a file from another directory) where you do not *i.e.* have read and write permission. The **mv** command has no effect on file permissions.

Cmd Name:	**mv**
Purpose:	To move a file to a new location, or to rename a file at the current location
Description:	The **mv** command moves files from one location to another, or renames files at their current location. You are free to move and rename files within your own home directory structure. The **-i** flag adds a check to the process: the system seeks confirmation that you do want to move (or rename) the file by asking for a **y** or **n** response.
Variations:	**mv, mv -i, move**

Examples:	`[/home/m3a/jenny]> mv books ~/chapters/books`
	*(move file to **/chapters** directory)*
	--- 	
	`[/home/m3a/jenny]> mv books ~/chapters/magazines`
	*(move file to **/chapters** directory and rename file)*
	--- 	
	`[/home/m3a/jenny]> mv -i books ~/chapters/books`
	`mv: overwrite /home/m3a/jenny/chapters/books (yes/no)?`
	(move with overwrite protection)
	--- 	
	`[/home/m3a/jenny]> mv notes.list footnotes`
	*(rename file **notes.list** to **footnotes**)*
	--- 	
	`[/home/m3a/jenny]> move last.dance first.dance`
	*(same as **mv** command)*

See Also:	Copying files (**cp**)

Removing Files (rm)

You can remove files from the system in a number of ways: by removing individual files, by removing a directory, or by using wild cards (* and ?) to remove a group of files. To remove a file, use the **rm** command and its associated flags and wild cards. You can also use the DOS-style **erase** command.

THE SYNTAX TO REMOVE A FILE IS:

rm [-flag] filename

The following command summary explains some of the flags available with the **rm** command:

Cmd Name:	**rm**
Purpose:	To remove files [delete them] from a file directory
Description:	The **rm** command erases [removes] files, directories and groups of files within your home directory structure. The **rm** command also removes links to a file [if there are any links] when the file is removed. You must be the owner of the file to remove it. You cannot remove files in any other user's home directory even when the files have full read- and write permissions for all users [=policy set-up at Ryerson University].

	Flags allied with the **rm** command include: **-e** displays message after file is removed; **-f** suppresses confirmation prompts on write protected files; **-i** prompts (asks for a **y** or **n** response) before removing them; and **-r** removes a directory and its contents. The DOS-type command **erase** also applies. You can't remove your home directory.
Variations:	**rm** , **rm -e** , **rm -f** , **rm -i**, **rm -ir**, **erase**

Examples:	`[/home/m3a/ralph]> rm sample.txt`
	`rm: remove sample.txt?`
	*(remove file **sample.txt**)*
	--- 	
	`[/home/m3a/Ralph]> rm -r ~/chapters`
	`rm: examine all files in directory /home/m3a/Ralph/chapters (yes/no)? y`
	`rm: remove books (yes/no)? y`
	`rm: remove rumble (yes/no)? y`
	`rm: remove chapters (yes/no)? y`
	(having removed [just now] *all files in the directory **/chapters** , now remove the dir?)*

See Also:	Removing Directories (**rmdir**), Moving Files (**mv**)

Browsing Files

Once read permissions are granted, you are free to look at any file, in any location in the system directory structure. The **cat** command lists a file's contents ("cat" stands for "concatenate"). The **more** and **page** commands allow you to browse files [read them] without all the options that the **cat** command has. Special features of the **cat** command include: writing a file to screen; creating a file based on input from the keyboard; connecting* one or more files into one file; and adding text to the end of an existing file (from keyboard input). Adding text to the end is called appending. For more information, type **man cat**.

***** [To «connect» one or more files into one file is to «concatenate» (or «catenate») them: hence name of the command.]

The following information summarizes the main uses of the **cat** command:

Cmd Name:	**cat** (concatenate)
Purpose:	Concatenates, appends or displays file contents
Description:	The **cat** command displays the contents of a file in various ways. The **cat** command writes information to the screen, a printer or another file. Type **cat** followed by the filename. If a file name isn't specified then the **cat** command reads from standard input (keyboard). Various flags produce different results: adding **-e** shows the end of each line with the symbol $; **-n** shows line numbers; **-t** displays tab characters as **^I;** **-v** displays all non-printing characters such as  ($); and **-** allows standard input to the **cat** command. By default the **cat** command produces a list on the screen. For files containing more than 23 lines pipe output through the **more** command.
Variations:	**cat**, **cat filename | more**, **cat -ev**, **cat -n**, **cat tv**, **cat -v**,

Examples:

```
[/home/m3a/froggy]> cat sample.txt
This is the file "sample.txt" listed in cat format. As this file is only
5 lines long, the complete file is shown before the system prompt returns.
If this file were longer than 23 lines it would scroll off the page.
[/home/m3a/froggy]>
```

Type **cat**, filename and control output with **more** option:

```
[/home/m3a/froggy]> cat books.list | more
This list of books is 200 lines long:
All About Birds, Macmillian, G. M., Temple Press
All The Way Home, Jackson, D.G., McGraw-Dale (etc)
Down The Up Ladder, Ronald, A.P. Simon & Pflug
[More 23%]
```

Pressing Enter displays the next line. Pressing the space bar displays the next screenful. The **more** captions shows the percent of file viewed. Pressing Ctrl C stops the displaying of the file. You may execute Ctrl-C under **more** at any time.

The **cat -ev** option displays hidden characters (**-v**). Note how the $ symbol represents Enter *i.e.* «carriage» return [from typewriter jargon]. This example shows the hidden characters **u** indicating that the word *file* is underlined when printed:

```
[/home/m3a/froggy]> cat -ev rumble
This is a short u\fileu\ to show the non-printing$
characters that show up using the cat "-ev" option.$
If this were a long u\fileu\ you would key in the following$
command sequence: "cat -ev rumble | more".$
[/home/m3a/froggy]>
```

The **cat -n** option displays line numbers in a file:

```
[/home/m3a/froggy]> cat -n apple.txt
  1 When one thinks of apple varieties of the past century
  2 you can't help being nostalgic. The varieties that we see
  3 today are the result of the need for high tree output, storage
  4 life, shipping durability and lastly, taste.
  5
  6 Today's supermarket, and even corner fruit and vegetable
  7 stand have no room for the Pippins and Talman Sweet's
  8 apples of yesterday
  9
 (etc.)
```

The **cat** command creates a short file by using the redirect symbol, the filename and Ctrl D, to end input from the keyboard:

```
[/home/m3a/froggy]> cat > new
This is a new file created through input from the keyboard. File is saved,
and ended, by pressing Ctrl-D when you have finished
Ctrl D
[/home/m3a/froggy]>
```

See Also: **more** and **page** commands, File listing command (**ls**)

More

Use the **more** command to display the contents of a file one screenful at a time. The syntax for the **more** command reads: [Note: on Linux systems, the command `less` is often used in place of `more`.]

more filename

Cmd Name:	**more**
Purpose:	Displays continuous text one screen at a time.
Description:	The **more** command displays text one screen at a time, or one line at a time. The percentage of the file already viewed as well as commands to continue or abort are displayed at the bottom of the screen. Press the space bar to display the next screen or press ↵Enter to show the next line. Previous display scrolls off the screen at top.
Variations:	**more**, **more -l**, **more -s**, **more -p**, **page**, **more +Number**

Examples:	`[/home/t3a/bfrank]>` **`more filename`**
	*(standard use of **more** command to view a file)*
	`This is a sample file that is printed to the screen showing the more command. It is printed until it reaches the screen limit, which varies`
	
	`between 20 and 22 lines. This is the 23rd line of this file so the`
	`-- More -- (51%) [Hit space to continue, Del to abort]`
	--- 	
	`[/home/t3a/bfrank]>` **`more -s filename`**
	*(**-s** option removes multiple blank lines)*

See Also:	Concatenate (**cat**)

File Permissions

Permissions are associated with every file on a UNIX system. These permissions control access to your files by you (the **u**ser), members of your group (**g**roup permissions) and any others with access to the same server (**o**thers). By default your home directory is protected from any member of your group and all others. You can change this if you wish to make your home directory and your files open to other users.

File permissions include read, write and execute permission. The **chmod** command [p.15] changes file permissions. The **ls -l** command [p.4] lists files so as to display permission information.

If a file is a directory, a **d** appears at the start of permissions listing. If a file links to another location, an **l** is shown at the start of its permission information. -- The following file list shows permission information with underlines [inserted here as highlights] showing **u**ser, **g**roup and **o**ther permissions :

```
Permissn Links Usr      Group    FileSize   Date      FileName
```

User permissions underlined:

```
-rw-r--r--   3 peggy    staff      3276  Mar  4 10:36  apple.txt
-rwxr-x--x   1 peggy    staff     45031  Jan 19 12:56  a.out
```

Group permissions underlined:

```
-rw-r--r--   1 peggy    staff       366  May  3 03:46  books.list
-rw-r-----   1 peggy    staff      4578  May 15 06:06  marks.old
```

All others permissions underlined:

```
-r-xr-xr-x   1 peggy    staff     35101  Apr 21 10:56  sortlnk
-r-x--x--x   2 peggy    staff       781  Feb 29 15:23  sort3
```

Directory Notation underlined:

```
drwx-x--x    3 peggy    staff       512  Jan  8 08:56  peggy
```

Link notation:

```
lrwx-x-x    17 donny    staff     45031  Jan 15 08:15  update.marks
```

Reading File Permission Information

As shown above, file permission information displays at the beginning of the file listing if you are using **ls -l** or **dir** . File permission codes read as follows:

- **-** No permissions set: permission level locked
- **r** Read permissions: file may be read or copied
- **w** Write permissions: file may be edited, read or copied
- **x** File is an executable file; compiled code; or an interpreted script file
- **d** Directory
- **l** Link

FILE PERMISSION INFORMATION BREAKS DOWN IN THE FOLLOWING MANNER:

a directory	user	group	others
-	rwx	rwx	rwx

THE FOLLOWING LIST SHOWS TYPICAL FILE PERMISSIONS AND THE LEVEL OF ACCESS:

-rw-r--r--	Default permissions: read and write by user, read by the group and read by all others. This means that if the directory is unprotected, anyone may look at this file or copy it.
-rwxr-xr-x	Default executable file permissions: read, write and executable by user; read and executable by group; and read and executable by all others. This means anyone can copy or execute the file.
-rw-r-----	Read and write by user; read only by members of your group. Members of your group may read and copy this file.
-rwx--x---	Read, write and executable by user, executable by members of your group: a good way to share an executable file within your group while maintaining complete control of it.
-r-x--x---	Read and executable by user, and executable by group. -- A Simple way to protect an interpreted script file from accidental writes. User is free to change write permissions to allow changes to the file.
drwx------	Default home directory permissions. Only the user gets read and write permission.
drwxr-x--x	Directory has read and write permissions for user, and read permission for group.
drwxr-xr-x	Directory has read permission for user, group and others. User has write permission.

Changing File Permissions (chmod)

Use the **chmod** command to change the file permission. Do this to any file within your home directory structure , and to your home directory. **chmod** allows you to add or subtract permissions at any level (user, group or others). Flags specify each level. These flags are: **u** (user or owner); **g** (group); **o** (others); and **a** (all). The + and **-** symbols are used to add or subtract permissions. Permission levels are: **r** (read); **w** (write); and **x** (executable).

The syntax for the **chmod** command reads as follows:

chmod [level (**u**, **g**, **o** or **a**)][+/-][permission (**r**, **w** or **x**)] **filename**

Summary of **chmod** commands are as follows:

Cmd Name:	**chmod**
Purpose:	Modify file permissions
Description:	The **chmod** command changes the **r**ead, **w**rite and e**x**ecutable permissions of any file for the following levels: **u**ser, **g**roup and **a**ll **o**thers. All levels (**u**,**g** and **o**) of file permissions may be changed at the same time by using the **a**ll option (a). By using more than one flag, like **ug** for user and group, you can change file permissions for more than one level. The + and - flags add or subtract permissions.
Variations:	consult the `chmod` manual page via **man chmod**
Examples:	`[/home/m3a/dreads]>` **`chmod g+xr filename`** *(add read and executable permissions for the group)* --- `[/home/m3a/dreads]>` **`chmod go-r filename`** *(remove read permissions for group and others)* --- `[/home/m3a/dreads]>` **`chmod u+x filename`** *(add executable permissions for user)* --- `[/home/m3a/dreads]>` **`chmod a+x filename`** *(add executable permissions to all users)* --- A special case - changing home directory permissions: [/home/m3a/dreads]> **cd ..** *(move to parent of /dreads directory)* `[/home/m3a]>` **`chmod g+xr dreads`** *(add read and execute permissions for group)* `[/home/m3a]>` **`cd`** *(return to home directory /dreads)* `[/home/m3a/dreads]>` --- `[/home/m3a/dreads]>` **`chmod a+r file1 file2`** *(add read permissions for a group of files)*
See Also:	Set default file permissions (**umask**)

SECURITY ISSUES

File permissions and file security are important issues. Keep in mind the following guidelines:

Passwords:

- Do not tell anyone your password.
- Change your password often -- at least once a month.
- Do not use your name, or any other person's name as a password. --
  Programs exist which can guess at words and proper names in a number of languages.

Security, File and Directory Permissions

By default, only you have read and write permissions on your directory -- which means that others cannot list the contents of your directory to find out what is there.

When you create a new file, the default permissions are set at read and write for you and read for all others. Although all others may read the file, the default directory permissions do not allow anyone to know the file exists. [Others are able to read the file only if they know it exists (know what the name of the file is).]

If you would like others in your group [first bullet below], or all others [second bullet below], to have read access to your files, the following permissions are recommended:

- Your home directory would have read permissions set for your group:
  e.g. **drwxr-x--x**
- Your home directory would have read permissions set for all others:
  e.g. **drwxr-xr-x**
- Subdirectories* should have read permissions for no one (except you):
  e.g. **drwx--x--x** * [*i.e.* subdirectories of («below») your (main, original, topmost) home directory.]
- Store files you wish to remain private in a subdirectory; store files you wish to share with others in your home directory. Others, or your group, can access only the files you wish to be public. Subdirectories fully protect the remaining files you want to have exclusive read and write access to.
- If you wish to share compiled executable files, turn off all read permissions so anyone can execute the executable files but yet not be able to read or to change them:
  e.g. **---x--x--x**

If you do not wish to share any file information, leave the file permissions set the same as the default [the default set-up at Ryerson University]. -- Your home directory will be fully protected: **drwx------**

NOTE: *Another user can copy a file that has read permissions set for group or others. When they get a copy of file in their own home directory they become the user (owner) of the copy of that file.*

Restore Operations

[Policy in effect at Ryerson University:] This service protects your work in case of power failure or other un-foreseen events. -- If you make a mistake and erase a file from your home directory, you can ask an advisor in room KHW71 to have a file restored. The minimum restore time is 24 hours during a working week. Files deleted more than two months ago are not recoverable. Restored files overwrite any files in your account which have the same name. The Systems Account Coordinator will need your Matrix ID; Student Number; the complete path and filename; and the restore date. The restore date is the day before the file was erased. For example: if *myfile1* was erased on May 29, its restore date is May 28.

Exchanging Files Between Matrix Computers

If you have more than one Matrix computer account , you may need to transfer files from one server to another. Use **ftp** to exchange files between Matrix computers. For more information see the Copying Files Between Computers section of this guide. [A «Matrix» account is internal to Ryerson University.]

CONNECTING FROM ONE SERVER TO ANOTHER

Using SSH

SSH [«**s**ecure **sh**ell»] provides secure log-in to remote systems. It encrypts information, including your user name and password. You are urged to use this client [meaning here: this program] where available.

To connect to an account with the same username on an alternate server, type **ssh servername** at the system prompt and press 

. For example:

```
[demo01] /ufs4/demo01> ssh stw .ryerson.ca
```

In this case, the user demo01 is logged in to *Turing* and is connecting to the demo01 account the Student Web Server (stw), in order to check his|her STW disk quota or to change their STW password. The login ID is demo01 for both accounts. Once the connection is successful, the user will be prompted to enter the password for his|her account (demo01) on STW.

To log in in to an account on another server with a different log-in ID , type **ssh -l loginID servername** and press 

. For example:

```
[demo01] /ufs4/demo01> ssh -l test02 stw .ryerson.ca
```

The user demo01 is connecting to STW from Turing. However, the log-in ID is not the same for both accounts. The log-in ID for the account on STW is test02. Once the connection is successful, the user is prompted to enter the password for the test02 account.

Typing **exit** ends the remote session [or Ctrl-D, below]. You will be returned to your Matrix home prompt.

The **hostname** command displays the name of the current server you are logged onto. These [log-in procedures coupled with the double-check made possible by the **hostname** command] are a good way to ensure that you are logged on to the correct server before you begin to work.

Pressing 

aborts an SSH session. [On many systems, Ctrl-D is equivalent to typing **exit**.]

□

[*For all chapters of this document please refer to http://www.ryerson.ca/acs/usersguide/.*]