

# Federal University Dutse



## CCS 206 - Introduction to Linux for Security and Forensics

Lecture Note 4  
By  
Muhammad S. Ali



# Manipulating Files & Directories

- The following commands are used for manipulating files and directories in Linux.
- `cp` – copy files and directories
- `mv` – move/rename files and directories
- `mkdir` – create directories
- `rm` – remove files and directories
- `rmdir` - remove the directory (ies), if they are empty.



- While it is easy to perform simple file manipulations with a GUI manager, complicated tasks can be easier with the command-line programs. For example, how could we copy all the HTML files one directory to another – but those that do not exist in the destination directory or are newer than the versions in the destination directory?

## Wildcards

- Wildcards are shell features that make these commands powerful. Because the shell uses file names so much, it provides special characters to help you rapidly specify groups of files.
- Using wildcards aka globbing allows you to select filenames based on patterns of characters. The following table list wildcards and what they select:



Wildcard	Matches
*	Any characters
?	Any single character
[ <i>characters</i> ]	Any character that is a member of the set <i>characters</i>
[! <i>characters</i> ]	Any character that is not a member of the set <i>characters</i>
[[: <i>class</i> :]]	Any character that is a member of the specified <i>class</i>

## Table: Wildcards

### Common Wildcard character classes

- The following table list the mostly command used wildcard character classes. Using wildcards makes it possible to construct very sophisticated selection criteria for filenames.



Character Class	Matches
<code>[ :alnum:]</code>	Any alphanumeric character
<code>[ :alpha:]</code>	Any alphabetic character
<code>[ :digit:]</code>	Any numeral
<code>[ :lower:]</code>	Any lowercase letter
<code>[ :upper:]</code>	Any uppercase letter

## Common Wildcard character classes

- Wildcards can be used with **any** command that accepts filenames as arguments



Pattern	Matches
*	All files
g*	Any file beginning with g
b*.txt	Any file beginning with b followed by any characters and ending with .txt
Data???	Any file beginning with Data followed by exactly three characters
[abc]*	Any file beginning with either a, b, or c
BACKUP.[0-9][0-9][0-9]	Any file beginning with BACKUP. followed by exactly three numerals
[:upper:]*	Any file beginning with an uppercase letter
[![:digit:]]*	Any file not beginning with a numeral
*[[:lower:]123]	Any file ending with a lowercase letter or the numerals 1, 2, or 3



# mkdir – Create Directories

- The mkdir command is used to create directories. It works like this:

mkdir directory...

- The dots in the syntax means that the argument can be repeated to create multiple directories as shown below:

```
$ mkdir dir1 # creates one directory (dir1)
```

```
$ mkdir dir1 dir2 dir3 # creates three directories (dir1, dir2 and dir3)
```

When given the option -p, then mkdir will create parent directories as needed.

```
$ mkdir -p mydir2/mysubdir2/threedirsdeep # creates two  
subdirectories in mydir2
```





# cp – Copy Files and Directories

- The cp command copies files or directories. It can be used in two different ways:

- To copy single file or directory item1 to file or directory item2

cp item1 item2

- To copy multiple items (either files or directories) into a directory

cp item... directory

- The following table list cp commonly used options:





Option	Meaning
-a, --archive	Copy the files and directories and all of their attributes, including ownerships and permissions. Normally, copies take on the default attributes of the user performing the copy.
-i, --interactive	Before overwriting an existing file, prompt the user for confirmation. <b>If this option is not specified, cp will silently overwrite files.</b>
-r, --recursive	Recursively copy directories and their contents. This option (or the -a option) is required when copying directories.
-u, --update	When copying files from one directory to another, copy only files that either don't exist or are newer than the existing corresponding files in the destination directory.
-v, --verbose	Display informative messages as the copy is performed.



## cp – examples

Command	Results
<code>cp file1 file2</code>	Copy <i>file1</i> to <i>file2</i> . If <i>file2</i> exists, it is <b>overwritten with the contents of <i>file1</i></b> . If <i>file2</i> does not exist, it is created.
<code>cp -i file1 file2</code>	Same as above, except that if <i>file2</i> exists, the user is prompted before it is overwritten.
<code>cp file1 file2 dir1</code>	Copy <i>file1</i> and <i>file2</i> into directory <i>dir1</i> . <i>dir1</i> must already exist.
<code>cp dir1/* dir2</code>	Using a wildcard, all the files in <i>dir1</i> are copied into <i>dir2</i> . <i>dir2</i> must already exist.
<code>cp -r dir1 dir2</code>	Copy directory <i>dir1</i> (and its contents) to directory <i>dir2</i> . If directory <i>dir2</i> does not exist, it is created and will contain the same contents as directory <i>dir1</i> .



# mv – Move and Rename Files

- The mv command performs both file moving and file renaming, depending on how it is used. In either case, the original filename no longer exists after the operation. mv is used in much the same way as cp:

- To move or rename file or directory item1 to item2

`mv item1 item`

- To move one or items from one directory to another

`mv item... directory`

- mv shares many of the options as cp as shown in table below:



# mv – options

Option	Meaning
-i, --interactive	Before overwriting an existing file, prompt the user for confirmation. <b>If this option is not specified, mv will silently overwrite files.</b>
-u, --update	When moving files from one directory to another, move only files that either don't exist in the destination directory or are newer than the existing corresponding files in the destination directory.
-v, --verbose	Display informative messages as the move is performed.

See the table below for example how to work with mv command:



# mv – examples

Command	Results
<code>mv file1 file2</code>	Move <i>file1</i> to <i>file2</i> . If <i>file2</i> exists, it is <b>overwritten with the contents of <i>file1</i></b> . If <i>file2</i> does not exist, it is created. In <b>either case, <i>file1</i> ceases to exist</b> .
<code>mv -i file1 file2</code>	Same as above, except that if <i>file2</i> exists, the user is prompted before it is overwritten.
<code>mv file1 file2 dir1</code>	Move <i>file1</i> and <i>file2</i> into directory <i>dir1</i> . <i>dir1</i> must already exist.
<code>mv dir1 dir2</code>	Move directory <i>dir1</i> (and its contents) into directory <i>dir2</i> . If directory <i>dir2</i> does not exist, create directory <i>dir2</i> , move the contents of directory <i>dir1</i> into <i>dir2</i> , and delete directory <i>dir1</i> .



# rm – Remove Files and Directories

- The rm command is used to remove (delete) files and directories, like this:

rm item...

- Where *item* is the name of one or more files or directories.
- Be careful using rm command particularly with wildcard characters, once you delete something it is gone, and you cannot undo it.
- Suppose you want delete just the HTML files in a directory. To do this, you type: **rm \*.html** which is correct, but if you accidentally place a space between the \* and the .html as: **rm \* .html** the rm command will delete all the files in the directory and then complain that there is not file called .html
- The table below lists some options used with rm command:





# rm – options

Option	Meaning
-i, --interactive	Before deleting an existing file, prompt the user for confirmation. <b>If this option is not specified, rm will silently delete files.</b>
-r, --recursive	Recursively delete directories. This means that if a directory being deleted has subdirectories, delete them too. To delete a directory, this option must be specified.
-f, --force	Ignore nonexistent files and do not prompt. This overrides the --interactive option.
-v, --verbose	Display informative messages as the deletion is performed.





# rm – examples

Command	Results
<code>rm file1</code>	Delete <i>file1</i> silently.
<code>rm -i file1</code>	Before deleting <i>file1</i> , prompt the user for confirmation.
<code>rm -r file1 dir1</code>	Delete <i>file1</i> and <i>dir1</i> and its contents.
<code>rm -rf file1 dir1</code>	Same as above, except that if either <i>file1</i> or <i>dir1</i> does not exist, <code>rm</code> will continue silently.



# Working with Commands

- A command can be one of four things:
- **An executable program** – like all those files we saw in `/usr/bin`. Within this category, programs can be *compiled binaries*, such as programs written in C or C++, or programs written in *scripting languages*, such as the shell, Perl, Python, Ruby, etc.
- **A command built into the shell** – bash supports a number of commands internally called *shell builtins*. The `cd` command, for instance, is a shell builtin.
- **A shell function** – Shell functions are miniature shell scripts incorporated into the environment.
- **An alias** – An alias is a command that we can define ourselves, build from other commands.



# Identifying commands

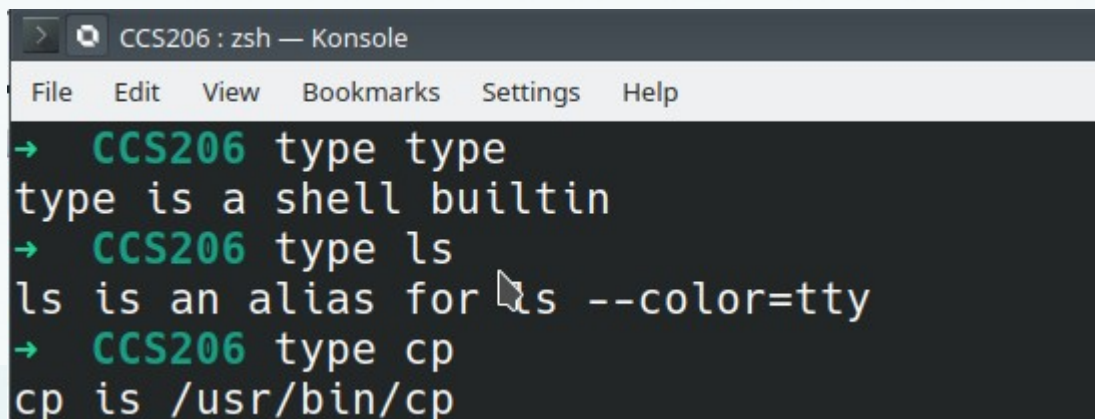
- It is often useful to know exactly which of the four kinds of commands is being used, and Linux provides a couple of ways to find out.

## **type – display a command's type**

- The type command is a shell builtin that displays the kind of command the shell will execute, given a particular command name. It works like this:

*type command*

- Where command is the name of the command you want examine. Examples are:



```
CCS206 : zsh — Konsole
File Edit View Bookmarks Settings Help
→ CCS206 type type
type is a shell builtin
→ CCS206 type ls
ls is an alias for ls --color=tty
→ CCS206 type cp
cp is /usr/bin/cp
```



# which – Display an Executable's Location

- Sometimes more than version of an executable program is installed on a system. While this is not very common on desktop systems, it is not unusual on large servers. To determine the exact location of a given executable, `which` command is used as follows:

*which command*

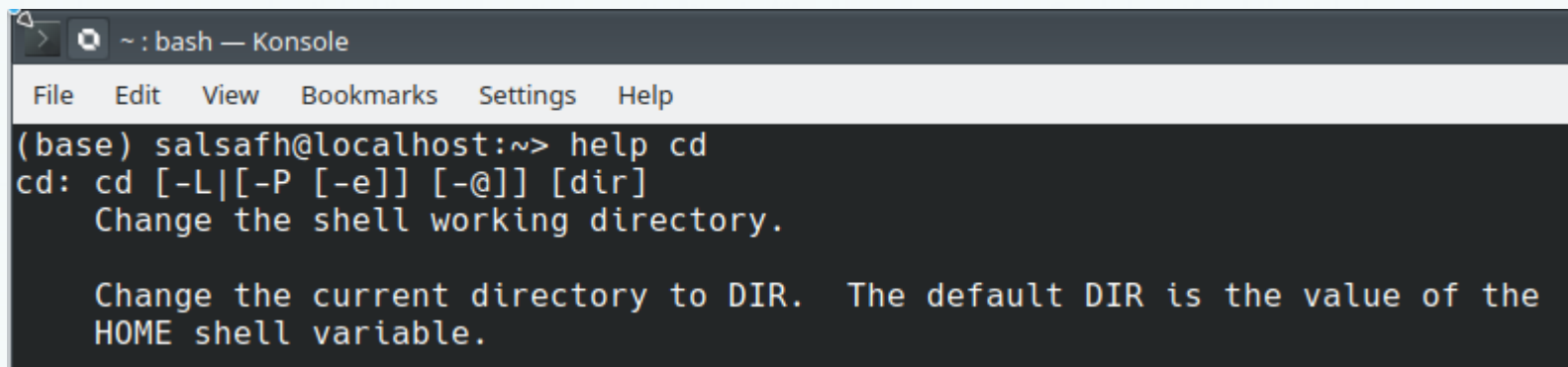
- Where *command* is usually an executable programs. Some Linux machines may display an error or no response when you used *which* on builtin or alias programs.

```
CCS206 : zsh — Konsole
File Edit View Bookmarks Settings Help
→ CCS206 which ls
ls: aliased to ls --color=tty
→ CCS206 which cd
cd: shell built-in command
→ CCS206 which cp
/usr/bin/cp
→ CCS206 which cd
cd: shell built-in command
```



# help – Get Help for Shell Builtins

- Bash has a builtin help facility for each of the shell builtins. To use it, type *help command*
- When a square brackets appear in the description of a command's syntax, they indicate optional items.
- A vertical bar character indicates mutually exclusive items. For example, *help cd* display something like `cd [-L |-P] [dir]`.
- This notation says that the command `cd` may be followed optionally by either a `-L` or a `-P` and further, optionally followed by the the argument `dir`.



```
~ : bash — Konsole
File Edit View Bookmarks Settings Help
(base) salsafh@localhost:~> help cd
cd: cd [-L|[-P [-e]] [-@]] [dir]
    Change the shell working directory.

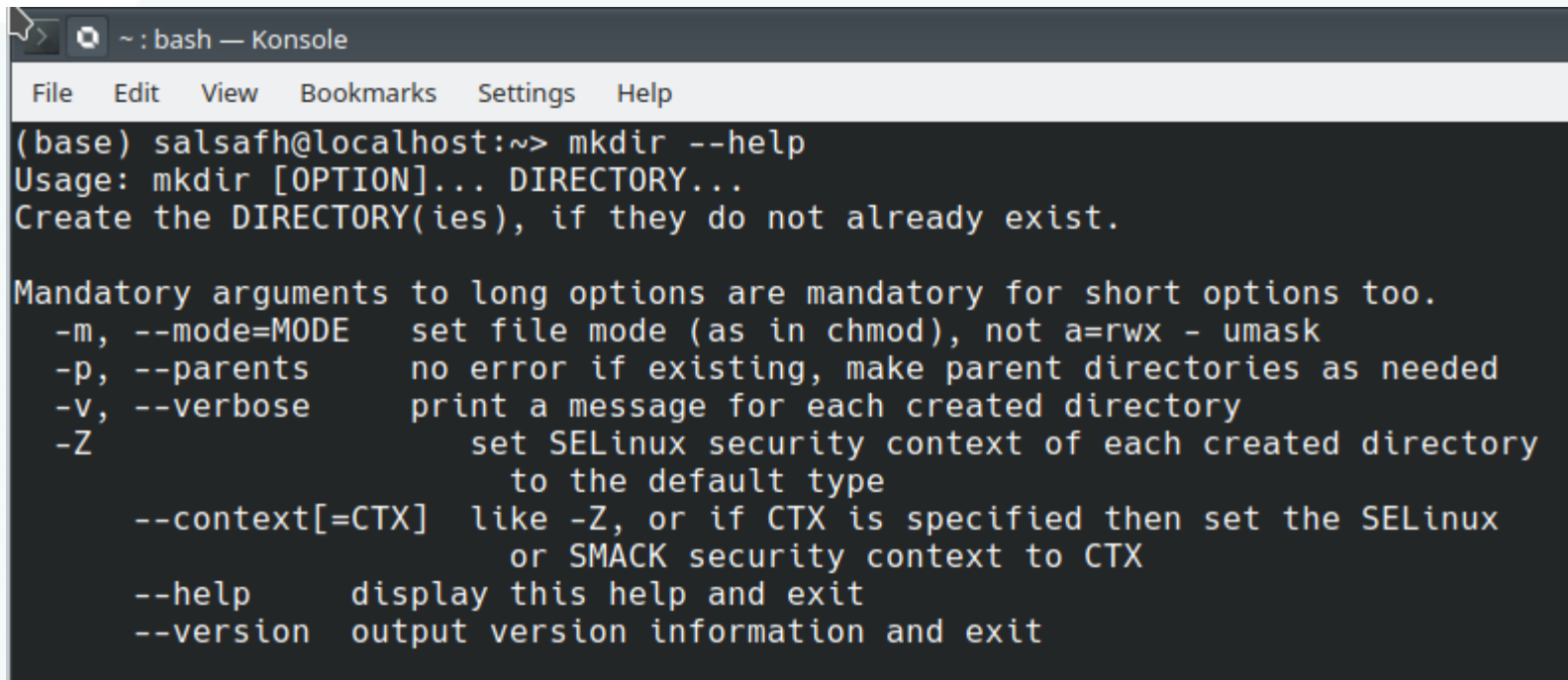
    Change the current directory to DIR.  The default DIR is the value of the
    HOME shell variable.
```





# --help – Display Usage Information

- Many executable programs supports a `--help` option that displays a description of the command's supported syntax and options. For example:

A terminal window titled '~: bash — Konsole' with a menu bar (File, Edit, View, Bookmarks, Settings, Help). The terminal shows the command '(base) salsa fh@localhost:~> mkdir --help' and its output. The output includes the usage 'Usage: mkdir [OPTION]... DIRECTORY...', a description 'Create the DIRECTORY(ies), if they do not already exist.', and a list of options: -m, --mode=MODE; -p, --parents; -v, --verbose; -Z; --context[=CTX]; --help; and --version.

```
(base) salsa fh@localhost:~> mkdir --help
Usage: mkdir [OPTION]... DIRECTORY...
Create the DIRECTORY(ies), if they do not already exist.

Mandatory arguments to long options are mandatory for short options too.
-m, --mode=MODE      set file mode (as in chmod), not a=rwx - umask
-p, --parents        no error if existing, make parent directories as needed
-v, --verbose        print a message for each created directory
-Z                  set SELinux security context of each created directory
                    to the default type
--context[=CTX]     like -Z, or if CTX is specified then set the SELinux
                    or SMACK security context to CTX
--help              display this help and exit
--version           output version information and exit
```

- Some programs may not support the `--help` option. It often results in an error message that reveals the same usage information.



# man – Display a Program's Manual Page

- Most executable programs intended for command-line use provide a formal piece of documentation called a *manual* or *man page*. A special paging program called `man` is used to view them like this:

*man program*

- Where `program` is the name of the command to view.
- Man pages may generally contain a title, a synopsis of the command's syntax, a description of the command's purpose, and a listing and description of each of the command's options.
- Man pages do not usually include examples, and they are intended as a reference, not a tutorial.
- On most Linux systems, `man` uses `less` to display manual page, so all of the familiar `less` commands work while displaying the page.
- The following table describes the layout of the manual page.





# man page organization

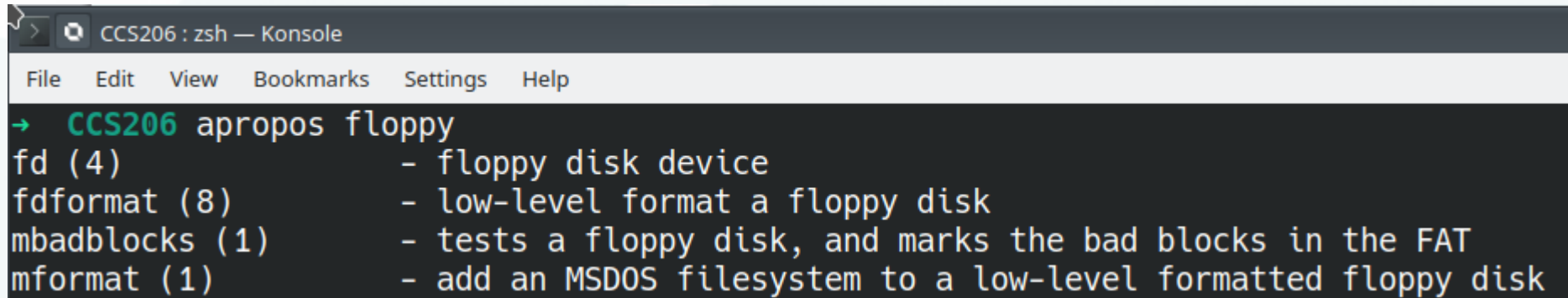
Section	Contents
1	User commands
2	Programming interfaces for kernel system calls
3	Programming interfaces to the C library
4	Special files such as device nodes and drivers
5	File formats
6	Games and amusements such as screensavers
7	Miscellaneous
8	System administration commands

Sometimes we need to look in a specific section of the manual to find what we are looking for. This is particularly true if we are looking for a file format that is also the name of a command. To specify a section number we use: **man 5 passwd**



# apropos – Display Appropriate Commands

- It is not possible to search the list of man pages for possible matches based on a search term. Though crude, this approach is sometimes helpful.



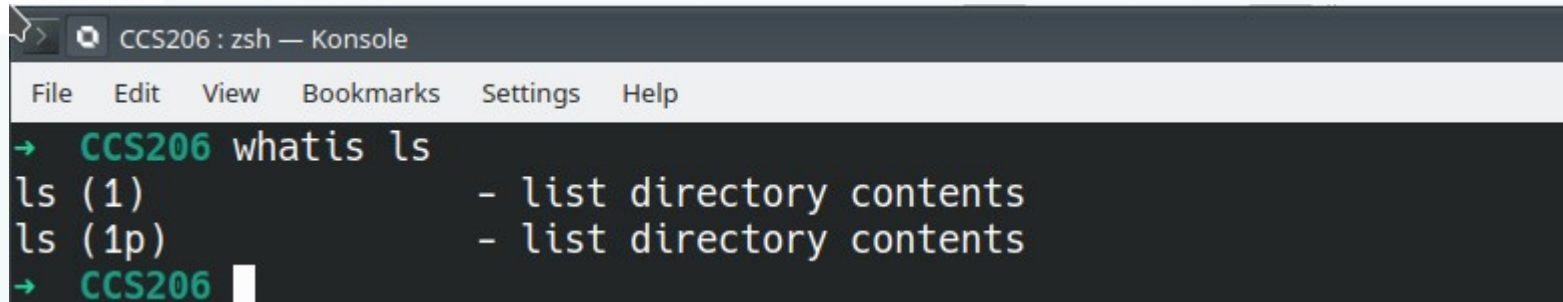
```
CCS206 : zsh — Konsole
File Edit View Bookmarks Settings Help
→ CCS206 apropos floppy
fd (4)                - floppy disk device
fdformat (8)          - low-level format a floppy disk
mbadblocks (1)        - tests a floppy disk, and marks the bad blocks in the FAT
mformat (1)           - add an MSDOS filesystem to a low-level formatted floppy disk
```

- The first field in each line of output is the name of the man page, and the second field shows the section. So, now you can use the command and the section number with man to get manual page for the section. For example –  
\$ man 8 fdformat



# whatis – Display a Very Brief Description of a Command

- The *whatis* program displays the name and a one-line description of a man page matching a specified keyword:



```
CCS206 : zsh — Konsole
File Edit View Bookmarks Settings Help
→ CCS206 whatis ls
ls (1)          - list directory contents
ls (1p)         - list directory contents
→ CCS206
```

- Each manual page has short description available within it. It searches the manual pages names and displays the manual page descriptions of any name matched.
- name may contain wildcards (-w) or be a regular expression (-r). Using these options, it may be necessary to quote the name or escape (\) the special characters to stop the shell from interpreting them.



## info – Display a Program's Info Entry

- The GNU Project provides an alternative to man pages called *info pages*. Info pages are displayed with a reader program named, appropriately enough, *info*. Info pages are hyperlinked much like web pages. info is used as follows:

```
info [OPTION]... [MENU-ITEM...]
```

- The first non-option argument, if present, is the menu entry to start from; it is searched for in all 'dir' files along INFOPATH. If is not present, info merges all 'dir' files and shows the result.
- Any remaining arguments are treated as the names of menu items relative to the initial node visited.
- The info program reads info files, which are tree-structured into individual nodes, each containing a single topic. Info files contain hyperlinks that can move you from node to node. A hyperlink can be identified by its leading asterisk and is activated by placing the cursor upon it and pressing the ENTER key.



A hyperlink can be identified by its leading asterisk and is activated by placing the cursor upon it and pressing the ENTER key.

## Some info commands

Command	Action
?	Display command help.
PAGE UP or BACKSPACE	Display previous page.
PAGE DOWN or Spacebar	Display next page.
n	Next—Display the next node.
p	Previous—Display the previous node.
u	Up—Display the parent node of the currently displayed node, usually a menu.
ENTER	Follow the hyperlink at the cursor location.
q	Quit.