GNU/Linux Command−Line Tools Summary

Gareth Anderson

<somecsstudent(at)gmail.com>

Chris Karakas − Conversion from LyX to DocBook SGML, Index generation

Revision History

Revision 1.2	15th April 2006	Revised by: GA

Corrected typing errors, generated new, much smaller index (more accurate in my opinion). Updated errors in document for TLDP.

Revision 1.1	28th February 2006	Revised by: CK

Corrected typos, generated new index (9000 index entries!).

Revision 1.0	6th February 2006	Revised by: GA

Major restructuring, now in a docbook book format. Removed large chunks of content and revised other parts (removed chapters and sectioned some areas more). This is likely the final release by the author, I hope that someone finds this guide useful as I do not intend to continue work on this guide.

Revision 0.7.1	25th February 2005	Revised by: CK

Set special characters in math mode, produced PDF and PS with Computer Modern fonts in OT1 encoding and created correct SGML for key combinations.

Revision 0.7	5th December 2004	Revised by: GA

Updated document with new grammatical review. Re−ordered the entire Text section. Removed a fair amount of content.

Revision v0.6	20th April 2004	Revised by: GA

Attempted to fix document according to TLDP criticisms. Added notes and tips more sectioning. Now complying to the open group standards for the UNIX
system trademark. Document should be ready for TLDP site.

Revision v0.5	6th October 2003	Revised by: GA

Fixed a variety of errors as according to the review and made some consistency improvements to the document.

Revision v0.4	15th July 2003	Revised by: GA

Made small improvements to the document as suggested (so far) by the thorough TLDP review, improved consistency of document and made small content additions.

Revision v0.3	26th June 2003	Revised by: GA

Minor errors fixed, updated the appendix with information for finding where a tool is from. Fixed referencing/citation problems and improved further reading and intro sections, added an audio section.

Revision v0.2	20th April 2003	Revised by: GA

This is the initial public release. Added more code−style then before, broke text−section into more subsections. Improved consistency of document and fixed various index entries.

Revision v0.1	27th March 2003	Revised by: GA

This is the initial draft release (the first release to be converted from LyX to DocBook SGML).

This document is an attempt to provide a summary of useful command−line tools available to a GNU/Linux based operating system, the tools listed are designed to benefit the majority of users and have being chosen at the authors discretion. This document is not a comprehensive list of *every* existent tool available to a GNU/Linux based system, nor does it have in−depth explanations of how things work. It is a summary which can be used to learn about and how to use many of the tools available to a GNU/Linux based operating system.

Table of Contents

Table of Contents

Table of Contents

Chapter 1. Introduction

This document is an attempt to summarise the many command−line based tools available to a GNU/Linux based operating system. This guide is not a complete listing (I doubt it's possible to document all available programs), this document lists many tools which are available to GNU/Linux systems and which are, or can be useful to the majority of users.

Each tool description provides a quick overview of it's function and some useful options for that individual tool.

The tools listed that require a GUI, usually the X windowing system, are those listed in the Graphics Tools section. All other tools are completely command−line−based and do not require a GUI to run.

If you are looking for information on GUI based tools you will need to look elsewhere.

Also note that a few of the tools in this guide are bash (the Bourne−Again−SHell) specific, tools specific to other shells are not listed in this document.

For some of the tools that are harder to use, or perform a more complex task, there are several mini−tutorials (or mini−guides; Chapter 20) within this document.

Where a mini−guide was considered unncessary, detailed descriptions that explain in detail how a particular tool works, and some examples of how to use it are provided.

Please note that the word "tool" is used interchangeably with the word "command", both have the same meaning (at least in this guide). For a more detailed explanation, read about the UNIX Tools Philosophy here: Chapter 3 or visit the links in the appendix, Section A.2.2.1.

To find out which tools are bash specific

To find out which tools are bash specific you can type:

```
enable −a
```

1.1. Who would want to read this guide?

Anyone who is interested in learning about the tools (also known as commands) available to them when using their GNU/Linux based operating system.

Why would you want to learn how to use the command−line (and available tools)? The *C*ommand *L*ine−*I*nterface (CLI), while difficult to learn, is the quickest and most efficient way to use a computer for many different tasks. The CLI is the normal method of use for most UNIX system administrators, programmers and some power users. While a GUI is better suited to some tasks, many operations are best suited to the CLI.

The major motivation behind learning the GNU/Linux CLI is the authors idea that, with software in general, the more time spent learning something equals less time spent performing that particular task *(authors opinion only)*.

This guide is aimed at beginners to intermediate users who want to learn about the command−line tools available to them. Advanced users may wish to use it as a command reference, however this document aims to list commands of interest, as judged by the authors opinion, it is not designed to be completely comprehensive, see the appendix, Section A.2.1 for further information. Or if you are not looking for a command reference guide, but a more gentle introduction to GNU/Linux you may be interested in the Introduction to Linux guide authored by Machtelt Garrels.

This guide could also be considered a summarised version of the Linux Cookbook. If you are looking for a book with more detailed descriptions of each tool have a look at the Linux Cookbook Homepage, also check out the command list from "Linux in a Nutshell 3rd Edition" for an index of 300+ commands and their explanations.

1.2. Who would not want to read this guide?

Anyone who is not interested in the command−line, or anyone looking for a detailed reference to all available GNU/Linux tools should look elsewhere. This is only a summary, while it does list many commands, it's not a complete listing (I don't think it's possible to make a complete listing anyway).

This document would not be of interest to those who already have an expert knowledge of the command−line interface and do require any reference information. Or those readers who require detailed lists of options for each command, the man pages are better suited to this purpose.

1.3. Availability of sources

The modifiable sources of the original book (in english), are available in LyX format (LyX Document Processor) or Machine−translated SGML (SGML markup language).

LyX is a completely free document processor based on LaTeX, downloadable from the LyX homepage..

See for the modifiable sources of this document. These are the official versions. We (the translators and current maintainers) plan to continue work on this document and add new chapters and enhancements. If you want to see the version we are currently working on (the "bleeding edge" version), check the GNU/Linux Command−Line Tools Summary Homepage from time to time (kindly hosted by Chris Karakas).

1.4. Conventions used in this guide

The following conventions are used within this guide:

italic
: Anything appearing in italic, *like this* is either an executable command or emphasized text. Tools (executable commands) are in italics to prevent confusion. Some tools have names which are real english words, such as the "locate" tool.

key combinations
: Are represented by using a '−' (dash sign) in−between the key(s), which must be used in combination. All combinations are also printed in italics to improve clarity. For example **CTRL−Z** means hold down the *Control key* and press the *z key.*

admonitions
: Admonitions are little pictures used to emphasize something of importance to the reader.

The five types used are:

This is a note

Notes often give important information about a tool.

This is a tip

This will offer a useful switch or useful way to use a tool.

This is something important

This is something that is considered very important. Consider it like a note with extra importance, they are usually there to save the reader time.

This is a caution

This will inform you of something that you be careful about (because it could be harmful to your system).

This is a warning

This will inform you of something that you shouldn't do (because it probably will break something within your system).

code examples

Code examples are shown for most commands.

Below is an example of what code looks like:

```
Hello World, I'm a code example. :)
```

command syntax

(or a similar phrase) simply shows how you would normally use the command. Often real examples are used instead of explaining the command syntax.

The phrase " Command syntax" is always followed by the way you would type a command in a shell.

The standard syntax for any tool is usually:

```
command −options file
```

Note

Note that some tools do not accept options.

wildcards

Also note that most commands, even when not explicitly stated, will work with standard wildcards (or globbing patterns) such as *, [A−Z] and various other standard wildcards. Refer to Section 20.4.1 for further information.

access keys

Access keys enable navigation through the document, without relying on a mouse. The following keys have been given special meaning in this document:

P

Previous page.

N
: Next page.

H
: Home of the document (Table of Contents).

U
: Up (takes you one level up the section hierarchy).

If you also happen to be reading the document from its original location, then the following access keys can also be used:

S
: Start (takes you to the author's start page).

T
: The current ("This") page, without the Sitemenu on the left.

M
: The current page in a frameset, where the left frame contains a Menu.

To use the access keys, you have to simultaneously press a modifier key, which may vary from browser to browser. For example in NN6+/Mozilla, the modifier key is **ALT**, so you have to use **ALT−N** to go to the next page, and **ALT−P** to come back. In other browsers such as IE6, the access keys just give focus to the associated link, so the sequence becomes **ALT−N Enter** . Try it, you'll like it! Inline graphic

1.5. Resources used to create this document

To create the GNU/Linux Command−Line Tools Summary, I used LyX, the document processor. To convert the LyX files to DocBook SGML I used the lyxtox Scripts created by Chris Karakas.

You may also want to check out the db2lyx package, created by Dr. B Guillion, which can be used to convert LyX files to XML DocBook and XML DocBook back to LyX.

I also had assistance from various The Linux Documentation Project volunteers (see the contributors section Section 1.7 for specific details).

1.6. Feedback

Feedback is necessary for the advancement of this guide. Positive, constructive criticism is encouraged. If you have ideas, suggestions, advice, or problems with this guide, please send an email to the author Gareth Anderson.

Contributions

If you wish to make contributions it is recommended (if possible) to read the LyX file(s) for this document. They contain various notes which you can't see in the other versions.

These notes highlight the areas that need contributions, certain tools which I cannot understand, tools which have not been added, or tools which were removed. These notes also explain some of the structure of this document.

1.7. Contributors

As you may be able to see, parts of this guide are based off various advice columns on GNU/Linux, anything that has being directly quoted from an article can be found in the references, *Bibliography*, section of this document.

The following is a list of people who have made a significant contribution to this document, in a rough chronological order.

Chris Karakas:
: Chris allowed the use of his lyxtox scripts to convert the LyX file of the document to working DocBook SGML output (to learn how to use the lyxtox scripts yourself, see Document processing with LyX and SGML).

◊ Chris provided useful suggestions and advice, and added an index listing for many of the commands.
◊ Chris is also responsible for the great looking HTML file for this document (the CSS file and HTML customisations are completely his work).
◊ Chris has also helped fix up problems in the document (many times), especially with docbook/sgml, and LyX related issues.
◊ Chris has also improved the structure of the document by adding labels and fixing minor errors.

William West:
: William provided a thorough review of the document as required by the Linux Documentation Project. He is responsible for a variety of improvements to the quality of this document.

His contributions include:

◊ Improvements to the readability of this document.
◊ Improvements to the structure and consistency of this document.
◊ Various grammar improvements throughout the document.
◊ Repair of some minor technical errors.

Tabatha Persad/Marshall:
: Tabatha, as the Linux Documentation Project Review Coordinator (at the time) also gave a brief review of this document. Her general advice was used to improve the structure, language and grammar of the document.

Rahul Sundaram:
: Rahul provided a brief review of this document for the Linux Documentation Project. Advice from his brief review was integrated into this document to improve readability and structure, several references were added as recommended by Rahul.

David Lawyer:
: David's criticism of the document (via the TLDP discuss list) were listened to, and attempts to improve the document were made. A number of his criticisms were addressed and improved.

George Harmon:
: George provided a second language review. His detailed review of the material allowed me to improve the general grammar of the document and some minor errors.

Machtelt Garrels (tille):
: Machtelt provided tips in regard to referencing the correct LDP documents from this guide. As well as general advice on improvements to the guide.

Michael Kerrisk:

Michael pointed out a number of technical errors in the document after his brief review on behalf of the TLDP during posts to the discussion list.

Chapter 2. Legal

The legal chapter provides information about the disclaimer that applies to the entire document and the licensing information.

2.1. Disclaimer

2.2. License

Chapter 3. The Unix Tools Philosophy

A tool is a simple program, usually designed for a specific purpose, it is sometimes referred to (at least throughout this document) as a command.

The " Unix tools philosophy" emerged during the creation of the UNIX operating system, after the breakthrough invention of the pipe '|' (refer to Chapter 6 for information on using the pipe).

The pipe allowed the output of one program to be sent to the input of another. The tools philosophy was to have small programs to accomplish a particular task instead of trying to develop large monolithic programs to do a large number of tasks. To accomplish more complex tasks, tools would simply be connected together, using pipes.

All the core UNIX system tools were designed so that they could operate together. The original text−based editors (and even TeX and LaTeX) use ASCII (the American text encoding standard; an open standard) and you can use tools such as; *sed*, *awk*, *vi*, *grep*, *cat*, *more*, *tr* and various other text−based tools in conjunction with these editors.

Using this philosophy programmers avoided writing a program (within their larger program) that had already been written by someone else (this could be considered a form of code recycling). For example, command−line spell checkers are used by a number of different applications instead of having each application create its own own spell checker.

This philosophy lives on today in GNU/Linux and various other UNIX system−based operating systems (FreeBSD, NetBSD, OpenBSD, etc.).

For further information (articles) on the UNIX tools philosophy please see the further reading section, here: Section A.2.2.1

Chapter 4. Shell Tips

The shell tips chapter provides handy tricks that you may wish to use when you are using a GNU/Linux shell (the command−line interface). This information includes handy shortcut key combinations, the shell's command history and information on virtual terminals.

If you can't boot into your system

If your having problems booting into your system you may like to use a shell so you can boot into your system and attempt to fix things up again.

To do this you need to pass the "init=/bin/sh" to your system before you boot up.

If you don't know how to do this please see Chapter 14, the technique is the same except this time you pass "init=bin/sh" rather than "single".

4.1. General Shell Tips

Automatic Command Completion

Use the TAB key and bash will attempt to complete the command for you automatically. You can use it to complete command (tool) names. You can also use it when working with the file−system, when changing directories, copying files et cetera.

There are also other lesser known ways to use automatic command completion (for example completing user names):[1]

ESC−Y (Y: special character)

testing autoindexing Will attempt to complete the command name for you. If it fails it will either list the possible completions (if they exist). If there are none it will simply beep (and/or) flash the screen.

CTRL−X−Y (Y: special character)

Lists the possible completions (it won't attempt to complete it for you) or beep if there are no possible completions.

Special−characters:

Use the following special characters combined with either **ESC−Y** or **CTRL−X−Y** , where Y is some special characters. For example **ESC−$** or **CTRL−X−$** to complete an environment variable name.

◊ ~ (tilde) complete a user name
◊ @ (at sign) complete a machine name
◊ $ (dollars sign) complete an environment variable name
◊ ! (exclamation mark) a magic character for completing a command name or a file name. The ! special character has the same function as the TAB key. It works in some other situations; for example when completing man page names.

alias

The *alias* command will list your current aliases. You can use *unalias* to remove the alias (to disable it just for one command add a "\" (back−slash) before the command)...

An alias allows one command to be substituted for another. This is used to make a command do something else or to automatically add certain options. This can be either be done during one session using the alias command (see below) or the information can be added to the *.bashrc* file (found in the users home directory).

Below is an example of what an alias section (within your *.bashrc* file) might look like:

```
# my personal aliases
alias cp='cp -vi' #to prompt when copying if you want to overwrite and will tell you where
alias rm='rm -i' #Prompts you if you really want to remove it.
alias mv='mv -i' #Prompts you if you are going to overwrite something
```

On any Mandriva GNU/Linux system the global aliases (for all users) are all in /etc/profile.d/alias.sh. The above listed commands already have aliases, as well as several other commonly used commands.

set −x

set is one of bash's inbuilt commands, try looking in the bash manual for its many usage options.

Using *set* with the *−x* option will make bash print out each command it is going to run before it runs it.

This can be useful to find out what is happening with certain commands such as things being quoted that contain wildcards or special symbols that could cause problems, or complex aliases. Use *set +x* to turn this back off.

Examples

After using *set −x* you can run the command:

```
ls
```

The output printed before the command runs (for example):

```
+ ls -F --color=auto
```

Which means that the command is really an alias to run *ls* with the *−F* and *−−color=auto* options. Use a "\" (backslash) before the command to run it without the alias.

\ (backslash)

The backslash escape character can be used before a shell command to override any aliases.

For example if *rm* was made into an alias for *rm −i* then typing "rm" would actually run *rm −i*.

However, typing *\rm* lets the shell ignore the alias and just run *rm* (its runs exactly what you type), this way it won't confirm if you want to delete things.

Using rm

Please note that the alias for the remove command is there for a reason. Using it incorrectly could remove files which you don't want removed.

Only use *\rm* if you know exactly what you are doing (recovering files is not easy, *rm* does not send things to a recycle bin).

The "\" character can be used before special characters (such as a space or a wildcard), to stop bash from trying to expand them. You can make a directory name with a space in it using a backslash before the space. For example you could type *cd My\ Directory\ With\ Spaces* which normally wouldn't work.

The "\" character can also be used to stop bash from expanding certain symbols (as an alternative you could use single quotation marks, although you may need to use both).

The TAB Key

Please note that using the TAB key (automatic−command−completion) will automatically use escapes for spaces (so you don't have to type them manually).

script

The "*script*" command creates a typescript, or "capture log" of a shell session − it writes a copy of your session to a file, including commands you type and their output.

~ (tilde character)

The tilde character is used as an alias to a users home directory.

For example, if your user−name was "fred", instead of typing *cd /home/fred* you could simply type *cd ~*. Or to get to fred's tmp directory (under his home directory) you could type *cd ~/tmp*.

Home directory shortcut

~ (tilde) can also be used as a shortcut to other users home directories, simply type: ~*user_name* and it will take you to the users home directory. Note that you need to spell the username exactly correct, no wildcards.

set bell−style none

This particular *set* command will turn off the system bell from the command−line (use xset −b for X windows). If you want the bell to stay off pernamently (no audible bell) then you can add this command to your ".bashrc" or ".bash_profile" (just add it to the same one you have your alises in...).

reset

The *reset* command re−initializes your current terminal. This can be useful when the text from your terminal becomes garbled, simply type "reset" and this will fix your terminal.

exit

Closes your current terminal (with x−terminals) or logs−out. Also try **CTRL−D** .

logout

Logs out of a terminal, also try **CTRL−D** .

echo

A little command that repeats anything you type.

Example:

```
echo "hello world"
```

Simply displays " hello world".

Example:

```
echo rm −R *
```

This will output what will be passed to the *rm* command (and therefore what would be deleted), putting echo before a command renders it harmless (it just expands wildcards so you know what it will do).

Also try using the *−e* option with echo. This will allow you to use the escape character sequences to format the output of a line. Such as '\t' for tab, '\n' for newline etc.

Using echo to prevent accidents

Typing: *echo command(s)* could save you the trouble of accidentally doing something you didn't expect.

Using *echo* allows you to expand the wildcards to understand what will happen before you actually run the command.

4.2. The command−line history

Using the command history

Use the up and down key's to scroll through previously typed commands. Press [Enter] to execute them or use the left and right arrow keys to edit the command first. Also see *history* (below).

The history command

The *history* command can be used to list Bash's log of the commands you have typed:

This log is called the "history". To access it type:

```
history n
```

This will only list the last *n* commands. Type "history" (without options) to see the the entire history list.

You can also type *!n* to execute command number n. Use *!!* to execute the last command you typed.

!−n will execute the command n times before (in other words *!−1* is equivalent to *!!*).

!string will execute the last command starting with that "string" and *!?string?* will execute the last command containing the word "string". For example:

```
!cd
```

Will re−run the command that you last typed starting with "cd".

" commandName !"* will execute the "commandName" with any arguments you used on your last command. This maybe useful if you make a spelling mistake, for example. If you typed:

```
emasc /home/fred/mywork.java /tmp/testme.java
```

In an attempt to execute emacs on the above two files this will obviously fail. So what you can do is type:

```
emacs !*
```

This will execute emacs with the arguments that you last typed on the command−line. In other words this is equivalent to typing:

```
emacs /home/fred/mywork.java /tmp/testme.java
```

Searching through the Command History (**CTRL−R**)

Use the CTRL−R key to perform a "reverse−i−search". For example, if you wanted to use the command you used the last time you used *snort*, you would type:

CTRL−R then type "snort".

What you will see in the console window is:

```
(reverse−i−search)`':
```

After you have typed what you are looking for, use the **CTRL−R** key combination to scroll backward through the history.

Use **CTRL−R** repeatedly to find every reference to the string you've entered. Once you've found the command you're looking for, use [Enter] to execute it.

Alternatively, using the right or left arrow keys will place the command on an actual command−line so you can edit it.

4.3. Other Key combinations

GNU/Linux shells have many shortcut keys which you can use to speed up your work, below is a rough list of some (also see **CTRL−R** in the history section of the commands, over here, Section 4.2).

CTRL−D
: the "end−of−file" (EOF) key combination can be used to quickly log out of any terminal. **CTRL−D** is also used in programs such as *"at"* to signal that you have finished typing your commands (the EOF command).

CTRL−Z
: key combination is used to stop a process. It can be used to put something in the background temporarily.

  For example, if you were editing a file with *vim* or *emacs* just press **CTRL−Z** to regain control of the terminal do what you want and then type *fg* to bring it back.

  For further information please see Section 9.3.

  If *fg* doesn't work

  If *fg* doesn't work you may need to type *jobs* and then *fg job_name or fg job_number*

CTRL−A and **CTRL−E**
: These key combinations are used for going to the start and end of the line on the command line. Use **CTRL−A** to jump to the start of the line, and **CTRL−E** to jump to the end of the line.

CTRL−K
: This key combination can be used to cut or delete what is currently in front of the cursor.

CTRL−Y
: This key combination can be used to paste the last thing you deleted (using **CTRL−K** or **CTRL−W**).

CTRL−W
: This key combination can be used to cut or delete the entire line that has being typed.

4.4. Virtual Terminals and screen

Using the key combination **ALT−F*** keys you may change to different virtual terminals. You will have several (usually 6) virtual terminals setup with shells. Number 7 is usually setup with X you need to use **CTRL−ALT−F*** to change to a terminal from within X (X as in the X windowing system).

screen

is a great program that allows you to switch between multiple virtual terminals on the one physical terminal that you are using. Its a command−line based window manager, clearly this isn't that useful if you do have virtual terminals, but its amazingly useful when you log into machines remotely, using ssh and similar, see Section 13.3. It works on key−combinations, you type

```
screen
```

On the command−line to begin. Now you start with one virtual terminal by default, but using the key combination **CTRL−A** and then hitting "C" you can create another virtual terminal to use.

Use **CTRL−N** to go to the next virtual terminal and **CTRL−P** to go to the previous virtual terminal. Also try hitting **CTRL−A** to go backwards and forwards between two particular terminals.

screen also has various other abilities that you can test out. The documentation and guides are well written so please feel free to read the manual page or try searching the internet.

Chapter 5. Help

The help chapter provides information on how you may access the documentation of the GNU/Linux system. There is normally a document describing every single tool you have installed, even if its only brief...

man

This command displays summary information on a program from an online manual. For example typing *man man* will bring up the manual page for man (the manual page viewer). Note: q is the quit key.

Command syntax:

```
man program_name
```

Also try

Specifying the section of the manual page, sometimes the man page is different for the same tool in different sections, note sections are numbered 1 to 9. Use apropos to find which section number to look in.

The syntax to look at a different section is:

```
man section_number tool_name
```

For example:

```
man 2 time
```

This will show you the man page called time in section 2, the equivalent page in section 1 is completely different

man −K keyword

Search the manual pages for a string, as in it will search all manual pages for a particular string within each individual man page, it will then prompt whether you would like to view each page it will find. Use double quotes " and " if there are spaces in the string you are typing.

Speed issue

Please be warned that this method is going to be really, really slow. You are searching *all* man pages for a string

man −f command

This will list details associated with the command. The root user must run *makewhatis* (see below) before this command will work.

Equivalent to *whatis*

This command is the same as running *whatis*

info

Provides a more detailed hyper−text manual on a particular command, this only works for some commands.

Command syntax:

```
info program_name
```

whatis

Displays a one−line description of what a program does. The string needs to be an exact match, otherwise *whatis* won't output anything. Relies on the whatis database (see below).

Command syntax:

```
whatis program_name
```

makewhatis

Make the whatis database for *apropos*, *whatis* and *man −f*.

Root Privileges

This takes some time and you require root privileges to do this.

apropos

Searches the whatis database for strings, similar to *whatis* except it finds and prints anything matching the string (or any part of the string). Also relies on the whatis database (see above).

Command syntax:

```
apropos string
```

Equivalent to...

apropos is the same as doing *man −k* (lowercase k).

Please note

You need to run *makewhatis* (as root) so *whatis*, *man −f* and *apropos* will work.

Also try

Using a program with the *−?*, *−−h*, *−−help*, and the *−h* options, they will display very short summary information on the command usage options.

Chapter 6. Directing Input/Output

The directing input/output chapter explains how you can use a program and send its output to a file or to another command that you wish to use. This technique is very powerful and there are a number of ways of doing this.

6.1. Concept Definitions

All three of the following definitions are called " File Streams." They hold information that is either received from somewhere or sent to somewhere. In a UNIX system, the keyboard input (standard input), information printed to the screen (standard output) and error output (also printed to the screen) are treated as separate File Streams.

Standard output
: Standard output is the output from the program printed to the screen, not including error output (see below).

Standard input
: Standard input is the input from the user. Normally the keyboard is used as the standard input device in a UNIX system.

Standard error
: Standard error is error output from programs. This output is also sent to the screen and will normally be seen mixed in with standard output. The difference between standard output and standard error is that standard error is unbuffered (it appears immediately on the screen) and standard error is only printed when something goes wrong (it will give you details of what went wrong).

6.2. Usage

>

The greater than symbol is used to send information somewhere (for example a text file)

Example:

```
cat file1 file2 > file1_and_2.txt
```

This will concatenate the files together into one big file named "file1_and_2.txt". Note that this will overwrite any existing file.

<

The less than symbol will insert information from somewhere (a text file) as if you typed it yourself. Often used with commands that are designed to get information from standard input only.

For example (using tr):

```
tr '[A-Z]' '[a-z]' < fileName.txt > fileNameNew.txt
```

The example above would insert the contents of "fileName.txt" into the input of *tr* and output the results to "fileNameNew.txt".

>>

The >> symbol appends (adds) information to the end of a file or creates one if the file doesn't exist.

<<

The << symbol is sometimes used with commands that use standard input to take information. You simply type << *word* (where word can be any string) at the end of the command. However its main use is in shell scripting.

The command takes your input until you type "word", which causes the command to terminate and process the input.

Using << is similar to using **CTRL−D** (EOF key), except it uses a string to perform the end−of−file function. This design allows it to be used in shell scripts.

For example type "cat" (with no options...) and it will work on standard input.

To stop entering standard input you would normally hit **CTRL−D** .

As an alternative you can type "cat << FINISHED", then type what you want.

When you are finished, instead of hitting **CTRL−D** you could type "FINISHED" and it will end (the word FINISHED will not be recorded).

2>

Redirects error output. For example, to redirect the error output to /dev/null, so you do not see it, simply append this to the end of another command...

For example:

```
make some_file 2> /dev/null
```

This will run make on a file and send all error output to /dev/null

|

The "pipe" command allows the output of one command to be sent to the input of another.

For example:

```
cat file1.txt file2.txt | less
```

Concatenates the files together, then runs *less* on them. If you are only going to look at a single file, you would simply use *less* on the file...

tee

Sends output of a program to a file and to standard output. Think of it as a T intersection...it goes two ways.

For example:

```
ls /home/user | tee my_directories.txt
```

Lists the files (displays the output on the screen) and sends the output to a file: "my_directories.txt".

&>

Redirects standard output and error output to a specific location.

For example:

```
make &> /dev/null
```

Sends both error output and standard output to /dev/null so you won't see anything...

6.3. Command Substitution

Command substitution is basically another way to do a pipe, you can use pipes and command substitution interchangeably, it's up to you which one you find easier...

Command substitution can be done in two distinct ways.

Method One (back−quotes)

Simply type:

```
command_1 `command_2 −options`
```

This will execute "command_2" and it's output will become the input to "command_1".

Backquote key

The back−quote key is usually located at the same place as the tilde, above the [Tab] key.

Method Two (dollars sign)

Simply type:

```
command_1 $(command_2)
```

This will execute "command_2" and it's output will become the input to "command_1".

Using the pipe instead

You can of course use pipes to do the same thing, if you don't know what a pipe is, please see Section 6.2. For example instead of doing:

```
less $cat file1.txt file2.txt
```

You could do:

```
cat file1.txt file2.txt | less
```

And end up with exactly the same result, it's up to you which way you find easier.

6.4. Performing more than one command

Executing the second command only if the first is successful

To do this you would type:

```
command1 && command2
```

command2 will be executed if command1 successfully completes (if command1 fails command2 won't be run). This is called a logical AND.

Executing the second command only if the first fails

To do this you would type:

```
command1 || command2
```

command2 will be executed if command1 does not successfully complete (if command1 is successful command2 won't be run). This is called a logical OR.

Executing commands sequentially

To execute commands sequentially regardless of the success/failure of the previous you simply type:

```
command1; command2
```

command2 will execute once command1 has completed.

More than two commands

You can continue to use ';' (semicolon) characters to do more and more commands on the one line.

Chapter 7. Working with the file−system

The working with the file−system chapter explains a number of commands that you use to move around the file system hierarchy and manipulate the files. Also explained are finding files and how to mass−rename files.

7.1. Moving around the filesystem

cd

Change directory. Use *" cd .."* to go up one directory.

One dot '.' represents the current directory while two dots '..' represent the parent directory.

" cd −" will return you to the previous directory (a bit like an "undo").

You can also use *cd absolute path* or *cd relative path* (see below):

Absolute paths

An " absolute path" is easily recognised from the leading forward slash, /. The / means that you start at the top level directory and continue down.

For example to get to /boot/grub you would type:

```
cd /boot/grub
```

This is an absolute path because you start at the top of the hierarchy and go downwards from there (it doesn't matter where in the filesystem you were when you typed the command).

Relative paths

A " relative path" doesn't have a preceding slash. Use a relative path when you start from a directory below the top level directory structure. This is dependent on where you are in the filesystem.

For example if you are in root's home directory and want to get to /root/music, you type:

```
cd music
```

Please note that there is no / using the above *cd* command. Using a / would cause this to be an absolute path, working from the top of the hierarchy downward.

ls

List files and directories. Typing "ls" will list files and directories, but will not list hidden files or directories that start with a leading full stop ".".

Example options:

◊ *ls −l* −−− long style, this lists permissions, file size, modification date, ownership.
◊ *ls −a* −−− this means "show all", this shows hidden files, by default any file or directory starting with a '.' will not be shown.
◊ *ls −d* −−− list directory entires rather than contents (see example below)
◊ *ls −F* −−− append symbols to particular files, such as * (asterisk) for executable files.
◊ *ls −S* −−− sort the output of the command in decending order sorted by size.
◊ *ls −R* −−− (recursive) to list everything in the directories below as well as the current directory.

Command syntax, either:

```
ls -options
```

This simply lists everything in the current directory, the options are not required (options such as *−l*, *−a* et cetera).

```
ls -options string
```

This lists files using a certain string. The string can contain standard wildcards to list multiple files, to learn more about standard wildcards please read Section 20.4.1

You can use *ls −d* to show directories that match an exact string, or use standard wildcards. Type " ls −d */" to list all subdirectories of the current directory. Depending on the setup of your aliases (see Chapter 4) you may simply be able to type *lsd* as the equivalent to *ls −d */.*

Examples for *ls −d*:

```
ls -d */
```

Lists all subdirectories of current directory.

```
ls -d string*
```

Lists directories that start with "string".

```
ls -d /usr/*/*/doc
```

Lists all directories that are two levels below the /usr/ directory and have a directory called "doc", this trick can come in quite handy sometimes.

You can also use

Depending on how your aliases (see Chapter 4) are setup you can also use *l*, *la* (list all) and *ll* (list long) to perform the above commands

pwd

Print working directory. Print the absolute (complete) path to the directory the user is currently in.

Command syntax:

```
pwd
```

This will tell you the full path to the directory you are in, for example it may output "/usr/local/bin" if you are currently in that directory.

tree

Outputs an ASCII text tree/graph starting at a given directory (by default the current directory). This command recursively lists all files and all directories.

In other words, it will list files within the directories below the current one, as well as all files in the current directory.

tree has a large number of options, refer to the manual page for details.

Command syntax:

```
tree
```

or

```
tree −option(s) /optional/directory/to/list
```

7.1.1. Finding files

find

find is a tool which looks for files on a filesystem. *find* has a large number of options which can be used to customise the search (refer to the manual/info pages).

Note that find works with standard wildcards,Section 20.4.1, and can work with regular expressions, Section 20.4.2.

Basic example:

```
find / −name file
```

This would look for a file named "file" and start at the root directory (it will search all directories including those that are mounted filesystems).

The `*−name'* option is case sensitive you can use the `*−iname'* option to find something regardless of case.

Use the *'−regex'* and *'−iregex'* to find something according to a regular expression (either case sensitive or case insensitive respectively).

The *'−exec'* option is one of the more advanced find operations. It executes a command on the files it finds (such as moving or removing it or anything else...).

To use the *−exec* option: use find to find something, then add the *−exec* option to the end, then:

```
command_to_be_executed ❶  then '{}' (curly brackets) ❷ then the arguments (for example a
```

See below for an example of use this command.

❶
This is the tool you want to execute on the files find locates. For example if you wanted to remove everything it finds then you would use *−exec rm −f*

❷
The curly brackets are used in find to represent the current file which has been found. ie. If it found the file shopping.doc then {} would be substituted with shopping.doc. It would then continue to substitute {} for each file it finds. The brackets are normally protected by backslashes (\) or single−quotation marks ('), to stop bash expanding them (trying to interpret them as a special command eg. a wildcard).

❸
This is the symbol used by find to signal the end of the commands. It's usually protected by a backslash (\) or quotes to stop bash from trying to expand it.

```
find / −name '*.doc' −exec cp '{}' /tmp/ ';'
```

The above command would find any files with the extension '.doc' and copy them to your /tmp directory, obviously this command is quite useless, it's just an example of what find can do. Note that the quotation marks are there to stop bash from trying to interpret the other characters as something.

Excluding particular folders with *find* can be quite confusing, but it may be necessary if you want to search your main disk (without searching every mounted filesystem). Use the *−path* option to exclude the particular folder (note, you cannot have a '/' (forward slash) on the end) and the *−prune* option to exclude the subdirectories. An example is below:

```
find / -path '/mnt/win_c' -prune -o -name "string" -print
```

This example will search your entire directory tree (everything that is mounted under it) excluding /mnt/win_c and all of the subdirectories under /mnt/win_c. When using the *−path* option you can use wildcards.

Note that you could add more *−path '/directory'* statements on if you wanted.

find has many, many different options, refer to the manual (and info) page for more details.

slocate

slocate outputs a list of all files on the system that match the pattern, giving their full path name (it doesn't have to be an exact match, anything which contains the word is shown).

Replaces *locate*

Secure locate is a replacement for *locate*, both have identical syntax. On most distributions locate is an alias to *slocate*.

Commmand syntax:

```
slocate string
```

This won't work unless

You need to run either *updatedb* (as root) or *slocate −u* (as root) for slocate to work.

whereis

whereis locates the binary, source, and manual page for a particular program, it uses exact matches only, if you only know part of the name use *slocate*.

Command syntax:

```
whereis program_name
```

which

Virtually the same as whereis, except it only finds the executable (the physical program). It only looks in the PATH (environment variable) of a users shell.

Use the *−a* option to list all occurances of the particular program_name in your path (so if theres more than one you can see it).

Command syntax:

```
which program_name
```

7.2. Working with files and folders

mkdir

Make a directory. Use *mkdir −p* to create subdirectories automatically.

Directories are Folders

Directories are sometimes called folders in other operating systems (such as Microsoft Windows)

Examples:

```
mkdir -p /home/matt/work/maths
```

This would create the directories "work" and "maths" under matt's home directory (if matt's home directory didn't exist it would create that too).

```
mkdir foo
```

This would create a directory in the current path named "foo".

rm

Remove/delete a file(s) or directories(s). You can use standard wildcards with this command Section 20.4.1.

Command syntax:

```
rm -options file_or_folder
```

You can of course use standard wildcards to delete multiple files or multiple directories and files.

Use the *−R* or *−r* option to remove recursively, this removes everything within subdirectories. Also try the *−f* option to force removal (useful when you don't want to be prompted).

Disabling Aliases (per execution)

On some systems such as Mandrake an alias will send *rm* to *rm −i* (prompting you for every file you wish to delete). To override this use: *\rm −R directory* (using the \ disables the alias for this run only)

rmdir

Remove an empty directory. If you want to remove a directory with files in it type " rm −R directory", read above for information on *rm −R*

Command syntax:

```
rmdir directory
```

This will only remove directory if it's empty otherwise it will exit with an error message.

mv

Move a file or a directory to a new location or rename a file/directory.

Rename example:

```
mv filename1 filename2
```

Renames filename1 to filename2.

To move a file or directory, simply type:

```
mv original_file_or_folder new_location
```

Note that this command can use standard wildcards Section 20.4.1 to move files (not for renaming).

Move and rename

Note that you can also move and rename a file in a single command. The difference is with the destination (right hand side) you change the filename to the new name of the file.

For example typing:

```
mv /etc/configuration.txt /home/joe/backupconfig
```

This would move the file "configuration.txt" to /home/joe/ and rename it "backupconfig"

cp

Copy a file. Has a number of useful options, such as *−R* (or *−r*) which recursively copies directories and subdirectories.

Command syntax:

```
cp −options file_or_files new_location
```

Examples:

```
cp file1 file2
```

Simply copy file1 to file2 (in the same directory).

```
cp /tmp/file1 ~/file2 /mnt/win_c
```

Where the last option is the directory to be copied to. The above example copies two files from different areas of the file system to /mnt/win_c

```
cp −R directory_and_or_files new_location
```

This command will copy directories (and all subdirectories) and/or files t*o new_location*

Note that this command can use standard wildcards Section 20.4.1 to copy multiple files.

You may also like to try the "−u" when moving large directories around, this copies only if the source file is newer than the destination to where you are copying to, or if the destination file does not exist at all.

ln

Create a link to a file. There are two types of links:

Hard links

Hard links are considered pointers to a file (the number is listed by typing *ls −l*). Each hard−link is a reference to a file.

The file itself only goes away when all hard−links are deleted. If you delete the original file and there are hard links to it the original file will remain.

Example:

```
ln target_name link_name
```

Will create a "hard link" to target_name called link_name, you need to delete both of these to remove the file.

Symbolic links

Symbolic links are created by typing "ln −s". When you remove the original file the symbolic link becomes broken, a symbolic link is similar to a windows "short−cut".

The advantage of symbolic links is that the target can be to something on another file−system, while hard−links can only exist on the same file−system.

For example:

```
ln −s target_name link_name
```

This creates a symbolic link to "target_name" called "link_name", if you delete the original file the symbolic link won't work (it becomes a broken link).

shred

Securely remove a file by overwriting it first. Prevents the data from being recovered by software (and even by most hardware), please be very careful when using shred as you may never be able to retrieve the data you have run the application on.

For example:

```
shred −n 2 −z −v /dev/hda1
```

"What this tells shred, is to overwrite the partition 2 times with random data (− n 2) then finish it up by writing over it with zeroes (−z) and show you its progress (−v). Of course, change /dev/hda1 to the correct partition . Each pass can take some time, which is why I set it to only do 2 random passes instead of the default 25. You can adjust this number, of course, to your particular level of paranoia and the amount of time you have.

Since shred writes on such a low−level, it doesn't actually matter what kind of filesystem is on the partition−−everything will be unrecoverable. Once shred is finished, you can shutdown the machine and sell or throw away the drive with peace of mind.

...However, even shre dding devices is not always completely reliable. For example, most disks map out bad sectors invisibly to the application; if the bad sectors contain sensitive data, `shred' won't be able to destroy it. [shred info page]."[2]

Shredding files doesn't work with all filesystems

Please note that as mentioned in the shred manual page (please see the manual and preferably info pages for more information). *shred* does not work correctly on log−structured or journaled filesystems, such as JFS, ReiserFS, XFS, Ext3 and many other modern filesystems

Alternatives to using shred

shred has its disadvantages when run on a filesystem. First of all since it has to be installed you cannot run shred on your operating systems filesystem, you also cannot use shred on a windows machine easily since you cannot install *shred* on this machine.

You may like to try alternatives such as the DBAN project that create self−booting floppy disks that can completely erase a machines hard disk.

You may also like to see how *chattr* can assist you in shredding files once they are removed (it has similar problems to shred, only ext2 and ext3 style filesystems...), please see Section 14.2.

du

Displays information about file size. Use *du filename* to display the size of a particular file. If you use it on directories it will display the information on the size of the files in the directory and each subdirectory.

Options for du (use *du −option(s)*):

- ◊ *−c* −− this will make *du* print a grand total after all arguments have being processed.
- ◊ *−s* −− summarises for each argument (prints the total).
- ◊ *−h* −− prints things in " human readable" mode; for example printing 1M (megabyte) rather than 1,024,000 (bytes).

Using the *−hs* options on a directory will display the total size of the directory and all subdirectories.

Command syntax:

```
du -options file_directory_or_files
```

Example:

```
du -hs *
```

This command will list the size of all files in the current directory and it will list the size of subdirectories, it will list things in human−readable sizes using 1024 Kb is a Megabyte, M for megabyte, K for kilobyte etc.

file

Attempts to find out what type of file it is, for example it may say it's: binary, an image file (well it will say jpeg, bmp et cetera), ASCII text, C header file and many other kinds of files, it's a very useful utility.

Command syntax:

```
file file_name
```

stat

Tells you detailed information about a file, including inode number creation/access date. Also has many advanced options and uses.

For simple use type:

```
stat file
```

dd

Copies data on a very low level and can be used to create copies of disks Section 20.3 and many other things (for example CD image files).

dd can also perform conversions on files and vary the block size used when writing the file.

Command syntax, note the block size and count are optional and you can use files instead of devices...

Please note

dd is an advanced and difficult to use command. Its also very powerful, so be careful what you do with it

Command syntax:

```
dd if=/dev/xxx of=/dev/xxx bs=xxxx count=x
```

Warning

The command *dd* is used to work on a very low level. It can be used to overwrite important information such as your master−boot record or various important sections of your hard−disk. Please be careful when using it (especially when working with devices instead of files).

touch

This command is used to create empty files, simply do *touch file_name*. It is also used to update the timestamps on files.

touch can be used to change the time and/or date of a file:

```
touch −t 05070915 my_report.txt[3]
```

This command would change the timestamp on my_report.txt so that it would look like you created it at 9:15. The first four digits stand for May 7th (0507), in MM−DD (American style), and the last four (0915) the time, 9:15 in the morning.

Instead of using plain numbers to change the time, you can use options similar to that of the *date* tool. For example:

```
touch −d '5 May 2000' some_file.txt
```

You can also use *−−date=* instead of *−d.* Also have a look at the date command under Section 8.1 for examples on using *−d* and *−−date=* (the syntax for the date part is exactly the same when using *−d* or *−−date*).

split

Splits files into several smaller files.

Use the *−b xx* option to split into *xx* bytes, also try *−k* for kilobytes, and *−m* for megabytes. You can use it to split text files and any other files... you can use *cat* to re−combine the files.

This may be useful if you have to transfer something to floppy disks or you wish to divide text files into certain sizes.

Command syntax:

```
split −options file
```

This will split the input file into 1000 lines of input each (thats the default...), and output (using the above example), with the input name file, "fileaa" (1st part of file), "fileab" (2nd part of file), "fileac" (3rd part of file) etc. until the there is no more of the file left to split.

7.3. Mass Rename/copy/link Tools

There are a few different ways to perform mass renaming of files in GNU/Linux (yes, mass renaming is possible!). There is also a perl script that renames the extentions on files, see Chapter 19.

Below are three ways to perform mass renaming of files, using the commands *mmv*, *rename* (a perl script) or some bash shell scripting.

mmv

mmv is a mass move/copy/renaming tool that uses standard wildcards to perform its functions.

mmv's manual page is quite difficult to understand, I have only a limited understanding of this tool. However *mmv* supports some standard wildcards.

According to the manual the ";" wildcard is useful for matching files at any depth in the directory tree (ie it will go below the current directory, recursively).

An example of how to use *mmv* is shown below:

```
mmv \*.JPG \#1.jpg
```

The first pattern matches anything with a ".JPG" and renames each file (the "#1" matches the first wildcard) to ".jpg".

Each time you use a \(wildcard) you can use a #x to get that wildcard. Where x is a positive number starting at 1.

mmv Homepage

You can find *mmv* on the web here.

Also be aware that certain options used with *mmv* are also applicable to other tools in the suite, these include *mcp* (mass copy), *mad* (mass append contents of source file to target name), *mln* (mass link to a source file).

Tip:

A Java alternative to *mmv* which runs on both GNU/Linux and Windows is available, Esomaniac

rename

rename is a perl script which can be used to mass rename files according to a regular expression.

An example for renaming all ".JPG" files to ".jpg" is:

```
rename 's/\.JPG$/.jpg/' *.JPG
```

Finding rename

You can get rename from various places. I would recommend trying CPAN Search Site, I found the script here Rename Script Version 1.4

Bash scripting

Bash scripting is one way to rename files. You can develop a set of instructions (a script) to rename files. Scripts are useful if you don't have *mmv* or *rename*...

One way to this is shown below:

```
for i in *.JPG;
do mv $i `basename $i JPG`jpg;
```

```
done
```

Note that the above script came from a usenet post. Unfortunately I do not know the author's name.

The first line says find everything with the ".JPG" extension (capitals only, because the UNIX system is case sensitive).

The second line uses *basename* (type man basename for more details) with the '$i' argument. The '$i' is a string containing the name of the file that matches. The next portion of the line removes the JPG extension from the end and adds the jpg extention to each file. The command *mv* is run on the output.

An alternative is:

```
for i in *.JPG;
do mv $i ${i%%.JPG}.jpg;
done
```

The above script renames files using a built−in bash function. For more information on bash scripting you may like to see the advanced bash scripting guide, authored by Mendel Cooper.

Chapter 8. Finding information about the system

time

If you are looking for how to change the time please refer to *date* here: Section 8.1.

time is a utility to measure the amount of time it takes a program to execute. It also measures CPU usage and displays statistics.

Use *time −v* (verbose mode) to display even more detailed statistics about the particular program.

Example usage:

```
time program_name options
```

/proc

The files under the /proc (process information pseudo file−system) show various information about the system. Consider it a window to the information that the kernel uses.

For example:

```
cat /proc/cpuinfo
```

Displays information about the CPU.

```
less /proc/modules
```

Use the above command to view information about what kernel−modules are loaded on your system.

dmesg

dmesg can be used to print (or control) the " kernel ring buffer". *dmesg* is generally used to print the contents of your bootup messages displayed by the kernel. This is often useful when debugging problems.

Simply type:

```
dmesg
```

df

Displays information about the space on mounted file−systems. Use the *−h* option to have *df* list the space in a 'human readable' format. ie. if there are 1024 kilobytes left (approximately) then *df* will say there is 1MB left.

Command syntax:

```
df -options /dev/hdx
```

The latter part is optional, you can simply use *df* with or without options to list space on all file−systems.

who

Displays information on which users are logged into the system including the time they logged in.

Command syntax:

```
who
```

w

Displays information on who is logged into the system and what they are doing (ie. the processes they are running). It's similar to *who* but displays slightly different information.

Command syntax:

```
w
```

users

Very similar to *who* except it only prints out the user names who are currently logged in. (Doesn't need or take any options).

Command syntax:

```
users
```

last

Displays records of when various users have logged in or out. This includes information on when the computer was rebooted.

To execute this simply type:

```
last
```

lastlog

Displays a list of users and what day/time they logged into the system.

Simply type:

```
lastlog
```

whoami

Tells the user who they are currently logged in as, this is normally the usename they logged in with but can be changed with commands like su). *whoami* does not need or take any options.

Simply type:

```
whoami
```

free

Displays memory statistics (total, free, used, cached, swap). Use the *−t* option to display totals of everything and use the *−m* to display memory in megabytes.

Example:

```
free −tm
```

This will display the memory usage including totals in megabytes.

uptime

Print how long the computer has been "up", how long the computer has been running. It also displays the number of users and the processor load (how hard the CPU has been working...).

The w command

The *w* command displays the output of the uptime command when you run this command. You could use the *w* command instead of uptime.

uname

uname is used to print information on the system such as OS type, kernel version et cetera.

Some *uname* options:

◊ *−a* −−− print all the available information.
◊ *−m* −−− print only information related to the machine itself.
◊ *−n* −−− print only the machine hostname.
◊ *−r* −−− print the release number of the current kernel.
◊ *−s* −−− print the operating system name
◊ *−p* −−− print the processor type.

Command syntax:

```
uname -options
```

xargs

Note that *xargs* is an advanced, confusing, yet powerful command. *xargs* is a command used to run other commands as many times as necessary, this way it prevents any kind of overload... When you run a command then add a *"| xargs command2*". The results of command1 will be passed to command2, possibly on a line−by−line basis or something similar.

Understanding *xargs* tends to be very difficult and my explanation is not the best. Refer to the examples below or try [6] of the *Bibliography* for another *xargs* tutorial.

Alternatives to using xargs

Please note that the below explanation of *xargs* is not the strongest (at the time of writing I could not find anything better :()).

Alternatives may include writing a simple bash script to do the job which is not the most difficult task in the world.

Examples:

```
ls | xargs grep work
```

The first command is obvious, it will list the files in the current directory. For each line of output of *ls*, *xargs* will run *grep* on that particular line and look for the string "work". The output have the each time *grep* is executed on a new line, the output would look like:

```
file_name: results_of_grep
```

If *grep* didn't find the word then there would be no output if it had an error then it will output the error. Obviously this isn't very useful (you could just do:

```
grep 'word' *
```

This is just a simple example...

xargs also takes various options:

◊ *−nx* −−− will group the first x commands together
◊ *−lx* −−− xargs will execute the command for every x number of lines of input
◊ *−p* −−− prompt whether or not to execute this particular string
◊ *−t* −−− (tell) be verbose, echo each command before performing it
◊ *−i* −−− will use substitution similar to find's −exec option, it will execute certain commands on something.

Example:

```
ls dir1 | xargs -i mv dir1/'{}' dir2/'{}'
```

The {} would be substituted for the current input (in this example the current file/directory) listed within the directory. The above command would move every file listed in dir1 to dir2. Obviously this command won't be too useful, it would be easier to go to dir1 and type *mv * ../dir2*

Here is a more useful example:

```
\ls *.wav | xargs -i lame -h '{}' '{}'.mp3
```

This would find all wave files within the current directory and convert them to mp3 files (encoded with lame) and append a ".mp3" to the end of the filename, unfortunately it doesn't remove the .wav and so its not too useful...but it works.

8.1. Date/Time/Calendars

There is one command to change both the date and time on a UNIX like system, *date*, there is also a simple calendar utility, *cal*. If you are looking to change the timestamps on files please see Chapter 8

date

Tells you the date (and the time) and is also used to set the date/time.

To set the date, type *date MM:DD:YYYY* (American style date) where MM is month, DD is the number of days within the month and YYYY is the year.

For example to set the date to the 1st January 2000 you would type:

```
date 01:01:2000
```

To set the time (where the *−s* option is to set a new time), type:

```
date -s hh:mm:ss
```

Another useful option you can use is *−−date="string"* (or *−d "string"*) option to display a date from x days ago or in x days (or x weeks, months, years et cetera). See the examples below.

Examples:

```
date --date="3 months 1 day ago"
```

Will print the date 3 months and 1 day ago from the current date. Note that *−−date="x month x day ago"* and *−d "x month x day ago"* are equivalent.

```
date -d "3 days"
```

The above command will print the date 3 days in the future from now.

cal

Typing *cal* will give you the calendar of the present month on your screen, in the nice standard calendar format. There are various options to customise the calendar, refer to the info/man page.

Example:

```
cal -y year
```

Will display a calendar for a specific year, simply use cal −y to print the calendar for the current year.

```
cal 2 2004
```

This will display the calendar for February 2004

8.2. Finding information about partitions

There are a number of ways to find out information on your hard disk drives, for information on mounted partitions also try *df* in Chapter 8

Using the proc filesystem

You can look through the information in the relevant area of the proc filesystem, under the directory of either /proc/ide/ or /proc/ide?/hd? where the first question mark is a number and the second is a letter (starting with 'a').

For example:

```
cd /proc/ide0/hda
```

Under this directory there will be various information on the hard drive or cdrom connected.

Using fdisk

Using *fdisk* with the *−l* option will output information on any hard drives connected to the system and information on their partitions (for example, the type of partition).

Information relating to using *fdisk* to partition hard disks can be found in your distributions documentation, the *fdisk* manual page or online.

Root Access Required

This command needs root access to work

Chapter 9. Controlling the system

The controlling the system chapter details commands that you may wish to use to interact with devices on your system and then details how to control processes and services/daemons.

eject
: eject simply tells a device to open (eject) the drive. Useful for cdrom/DVD drives.

  For example the command below would eject the cdrom−drive (if your cdrom is linked to /dev/cdrom):

```
eject /dev/cdrom
```

  This won't work unless

  This will only work if the user has permission to mount the partition. Please see the tip in Section 9.1 for more information.

9.1. Mounting and Unmounting (Floppy/CDROM/Hard−drive Partitions)

Allowing Users to mount partitions

By default a UNIX system will allow normal users to unmount partitions. However unless given permission by the superuser, users will not be allowed to mount partitions.

The commands listed below will not work for normal users unless users have permission to mount that device.

If your particular distribution is setup not to allow users to mount partitions its not very hard to change this, simply edit the */etc/fstab* file (as root) and:

```
Replace the word "defaults" with "user" or
Add "user" to the end of the options list for the particular partition(s).
```

mount
: Mount a device. Attach the device to the file−system hierarchy (the tree (/)). This needs to be done so you can access the drive (see below, Section 9.1 for an example).

umount
: 'Unmount' a device. The command *umount* (no 'n') unmount's a device. It removes it from the file−system hierarchy (the tree (/)). This needs to be done before you remove a floppy/CDROM or any other removable device (see below, Section 9.1 for an example).

smbmount //wincomp/c /mnt/win
: Where "win" would be the place you want it mounted and "wincomp" is the IP address or name of your windows computer.

  Please note

  Using ping/smbmount/ssh or other UNIX system programs with a computer name rather than IP address will only work if you have the computer listed in your /etc/hosts file. Here is an example:

```
192.168.1.100 new
```

This line says that their is a computer called "new" with IP address 192.168.1.100. Now that it exists in the /etc/hosts file I don't have to type the IP address anymore, just the name "new".

smbmount is a tool from the samba package, it can mount a remote windows file−system onto your current computer.

Un−mounting uses the same syntax as 'umount', as listed above, or you may like to use:

```
smbumount /mountpoint
```

Here are some more examples of how to mount a file−system:

```
mount −t ext2 /dev/fd0 /mnt/floppy ❶
mount −t iso9660 /dev/hdb /mnt/cdrom ❷
mount −t iso /tmp/image_file /mnt/iso_file/ −o loop ❸
```

❶ The windows filesystem is known as vfat (standard on Windows 9x) or NFTS (standard on Windows 2000 and XP).

❷ for CDROM's

❸ This will mount an image file (usually a CD image file) so you can view/change the files (it will appear to be like any other device).

The −t option

On any system running a newer version of the Linux kernel the *−t* option is not always necessary and can be left out.

Examples of how to unmount a file−system (necessary before you eject/remove disk):

```
umount /mount_point
```

An example unmount point could be "/mnt/floppy" or "/mnt/cdrom"

9.2. Shutting Down/Rebooting the System

shutdown now
: Shutdown the computer immediately (don't power down). Note that in UNIX systems this kind of shutdown means to go to " single−user mode". Single−user mode is a mode where only the administrator (root) has access to the computer, this mode is designed for maintenance and is often used for repairs.

  For example this would take you to single user mode

```
shutdown now
```

shutdown −h now
: Shutdown *(−h = halt)* the computer immediately. It begins the shutdown procedure, press **CTRL−C** (break−key) to stop it. After the end of the command you can also leave a message in quotation marks

which will be broad−casted to all users, for example:

```
shutdown −h now "Warning system malfunction, self−destruct imminent"
```

This would halt the system and send the message to anyone who is currently logged in.

Shutting down at a particular time

You can also put a time that the system should shutdown instead of "now". Typing "+x minutes" (any number of minutes is appropriate) or you can even set an exact time. For example to shutdown at 11:50 type:

```
shutdown −h 11:50
```

Shutdown −h vs poweroff

On some systems, *shutdown −h* and *halt* do not actually turn the system's power off. On systems that do not power off with these commands use the *poweroff* command

halt
: The same as *shutdown −h now* doesn't take any options, this command simply shuts down immediately.

shutdown −r now
: Shutdown (*−r = reboot*) the computer immediately. It begins the reboot procedure, press **CTRL−C** (break−key) to stop it. After the end of the command you can also leave a message in quotation marks which will be broad−casted to all users, for example:

```
shutdown −r now "Warning system rebooting, all files will be destroyed"
```

This would reboot the system and send the message to anyone who was logged in.

Rebooting at a particular time

You can also put a time that the system should reboot instead of "now". Typing "+x minutes" (any number of minutes is appropriate) or you can even set an exact time. For example to reboot at 11:50 type:

```
shutdown −r 11:50
```

reboot
: The same as *shutdown −r now*, doesn't take any options, simply reboots the computer immediately.

CTRL−ALT−DEL
: (key−combination) May be used from a terminal to reboot or shutdown, it depends on your system configuration. Note that this doesn't work from an xterminal. **CTRL−ALT−DEL** begins the reboot/shutdown immediately, the user does not have to be logged in.

You can change the behaviour of CTRL−ALT−DEL from rebooting

To disable **CTRL−ALT−DEL** from rebooting your computer (or to have it do something different), you can edit the /etc/inittab file (as root).

Here is how it looks on a Mandrake/Mandriva Linux system:

```
# Trap
CTRL-ALT-DEL

ca::ctrlaltdel:/sbin/shutdown -t3 -r now
```

Note that the # means a comment (and is not used). If you simply put a # (hash) before the command it would disable it (it would become a comment).

You could also change the command it runs for example if you changed the *−r* to a *−h* the computer would turn off instead of rebooting, or you could have it do anything you want. It's up to your creativity to make it do something interesting.

9.3. Controlling Processes

ps

Will give you a list of the processes running on your system. With no options, *ps* will list processes that belong to the current user and have a controlling terminal.

Example options include:

- ◊ *−aux* −−− list all running processes (by all users with some information).
- ◊ *−a* −−− list all processes from all users.
- ◊ *−u* −−− list more information including user names, %cpu usage, and %mem usage et cetera.
- ◊ *−x* −−− list processes without controlling terminals.
- ◊ *−l* −−− display different information including UID and nice value.
- ◊ *−−forest* −−− this makes it easier to see the process hierarchy, which will give you an indication of how the various processes on your system interrelate (although you should also try *pstree*).

For example to list all running processes with additional information, simply type:

```
ps -aux
```

pstree

Displays the processes in the form of a tree structure (similar to how *tree* does it for directories).

Use the *−p* option to show process id's.

Example:

```
pstree -p
```

This would list all processes and their id's.

pgrep

This command is useful for finding the process id of a particular process when you know part of its name.

Use the *−l* option to list the name of the process as well and the *−u* option to search via a particular user(s).

Normally *pgrep* will only return the pid number; this way you can use it with other commands.

Examples:

```
kill $(pgrep mozilla)
```

This would kill any process name that starts with mozilla. Note that this is the same as using *pkill* (see below).

If you are unfamiliar with the *$( )* part of this command, please refer to Section 6.4.

To list processes id's and names type:

```
pgrep −l process_name
```

top

Displays the 'top' (as in CPU usage) processes, provides more detail than *ps*.

top also provides an updated display, it has many options that make it fully customisable, refer to the manual or info page for details.

kill

To kill processes on your system, you will need their pid's or id's . Use *ps* or *pstree* to find out the process id's (pid's), or use *jobs* to find out id's.

killall and pkill − kill a process by name

pkill and *killall* can be a lot easier to use than *kill*. *pkill* allows you to type part of the name of a process to kill it, while *killall* requires the full process name. See below for more information.

Examples:

```
kill pid
```

Simply kill a process (allow it time to save it's files and exit)

```
kill %id
```

Same as above, except it uses an id instead of a pid, you need to use a % (percent) when using an id to kill.

```
kill −kill pid
```

Force a process to be killed (won't allow files to be saved or updated); only use when necessary because all data that the program had will be lost.

There are also many other kill options such as kill −HUP (hangup)... refer to the manual/info pages for more information.

killall

Kill a process by it's name, uses names instead of process id's (pid's). Use *−v* to have *killall* report whether the kill was successful or not and *−i* for interactive mode (will prompt you before attempting to kill).

pkill − a little like a killall with regular expressions

pkill is another command that allows processes to be killed but does so using regular expressions. See below for more information.

For example:

```
killall −iv mozilla
```

Would kill anything named "mozilla" and prompt you before each kill and report whether the kill was successful or not. Unfortunately you need to get the name exactly right for *killall* to work, you would need to use "mozilla−bin" to kill the mozilla browser. If you want something where you don't need to know the exact name try *pkill* (below).

pkill

pkill is used to kill processes according to an extended regular expression. Use the *−u* option to kill using a user name(s) and process name (for example to only kill a process of a certain user). *pkill* can also send specific signals to processes.

For normal usage simply type:

```
pkill process_name
```

Note that the "process_name" doesn't have to be an exact match...

Or to kill the "process_name" of only the users "fred" and "anon" type:

```
pkill −u fred anon process_name
```

skill

skill is used to send a command/username/tty a particular signal.

skill has a number of options available to ensure correct interpretation (otherwise it just guesses what it is), simply type *skill −option(s)*

- ◊ *−L* −−− list the various signals that can be sent
- ◊ *−u* −−− specify a username; this is obviously followed by the user name or a space−seperated list of usernames.
- ◊ *−p* −−− process id (followed by the process id)
- ◊ *−c* −−− command name (this is the same as *killall*)
- ◊ *−t* −−− (tty number)
- ◊ *−v* −−− verbose mode
- ◊ *−i* −−− interactive mode.

skill can be used to stop, continue, or kill processes using the username, command name or process id (or send them any variety of signals you like).

Useful example:

```
skill −STOP abusive_user_name
```

The above command will stop all of that users processes, this will cause his screen to freeze until you type:

```
skill −CONT abusive_user_name
```

This would tell that all processes may continue as before. Note that this would only work if you are root. Also note you can list more than one user name with the command so it will apply to multiple users.

CTRL−C

The break key, will kill (break, stop) something that's running on your terminal.

jobs

Prints currently running jobs, as in processes you have executed within the shell.

bg

Backgrounds a process. To start a program in the background (so it doesn't take over the terminal) use an "&" (ampersand) sign at the end of the command. You usually use **CTRL−Z** to suspend something you are currently using. You can simply use *bg* to resume in the background the last job suspended...

Command syntax:

```
bg job_number
```

or

```
bg job_name
```

fg

Bring a process to the foreground, so you can interact with it. The process will use your current terminal. Note simply use *fg* to foreground the last job number suspended...

You can bring jobs to the foreground by name or by number (use *jobs* to find the number).

Command syntax:

```
fg job_number
```

or

```
fg job_name
```

nice

Sets the priority for a process. *nice −20* is the maximum priority (only administrative users can assign negative priorities), *nice 20* is the minimum priority. You must be root to give a process a higher priority, but you can always lower the priority of your own processes...

Example:

```
nice −20 make
```

Would execute *make* and it would run at maximum priority.

renice

Changes the priority of an existing command. You may use the options *−u* to change the priorities of all processes for a particular user name and *−g* to change priorities for all processes of a particular group. The default is to change via the process id number.

Example:

```
renice +20 2222
```

This would change the priority of process 2222 to +20 (minimum priority).

snice

snice works very similarly to *skill*, only it changes the priority of the process(es). Its function is similar to that of *renice*.

To use options (to ensure correct interpretation) you simply type *snice −option(s):*

- ◊ *−u* −−− specify a username; this is obviously followed by the user name or a space−seperated list of usernames.
- ◊ *−p* −−− process id (followed by the process id)
- ◊ *−c* −−− command name (this is the same as *killall*)
- ◊ *−t* −−− tty number
- ◊ *−v* −−− verbose mode
- ◊ *−i* −−− interactive mode.

Example:

```
snice −10 −u root
```

This would increase the priority of all root's processes.

9.4. Controlling services

Concept Definitions

UNIX systems use scripts to control "daemons" which provide "services" (for example your sound output) to run a UNIX system. UNIX systems consist of a variety of services (daemons).

A "daemon" is a system process which runs in the background (zero interaction) performing a particular task.

Daemons normally have a "d" on the end of their name and either listen for certain events or perform a system task, for example *sshd* listens for secure shell requests to the particular machine and handles them when they occur.

Daemons usually perform critical system tasks such as control swap−space, memory management and various other tasks.

service

service is a shell script available on Mandrake/Mandriva and Redhat systems which allows you to perform various tasks on services.

◊ Use the *−s* option to print the status of all services available
◊ Use the *−f* option followed by a service name to restart that particular service.
◊ Use the *−R* option to restart all services (note that this will kill any current services running, including the X windows system).

For example to restart the daemon *sshd* you would type:

```
service −f sshd
```

Using the script directly

You may also execute the shell script directly from */etc/init.d*. Simply go to that directory then type *./script_name*.

Executing the script should return the options it can take, by default they will be:

◊ restart −−− this will make the service stop and then start again.
◊ start −−− this option will start a service (assuming its not running).
◊ stop −−− this option will stop a service (assuming its running).
◊ status −−− this option will tell you about the service

Chapter 10. Managing users

su username
: (Switch User), change to a different user.

  Use *su −* to switch to root or *su username*, to switch to a different username.

  Using sudo

  Its often considered better practice to use the *sudo* command rather than switch to the root user

  The *sudo* command allows you to perform actions as root but logs the actions you take (so you can trace anything that was done to the system by yourself or others). *sudo* has a very good manual page which provides plenty of information about it.

  You use sudo similar to how you execute a normal command with sudo prepended to it, for example:

```
sudo rpm −U myrpm.i386.rpm
```

  This would allow you to install a rpm even if you have the correct sudo access

  Note that if you want to return to your original user you don't use *su* again, type *exit* or press **CTRL−D** .

  Simply typing *su* will give you some root privileges, but there are minor complications relating to environment variables. It's generally considered better practice to use *su −* because it has no restrictions.

root
: The superuser. This user has power over everything and all, and can do anything with the system (including destroy it, and of course fix it :)). This user is used to perform most administration functions on the system.

10.1. Users/Groups

All user information is normally listed in the "/etc/passwd" file and the group information in the "/etc/groups" file.

If you need to edit either file it is recommended that you use *vipw* to edit the password file and *vigr* to edit the group file. These particular commands take care of any processing and locking of the files before and after editing them.

There is a lot of information about adding/removing/controlling users and groups, this information is only the minimal information required.

chsh
: Used to change your login shell.

  To list the shells available type:

```
chsh --list-shells
```

Simply type *chsh* then [Enter], then type the name of the shell you would like to use every time you login.

chfn

Change finger information.

The information this command changes is reflected in the /etc/passwd file, use this utility to update your real name, office and home phone numbers (if they exist).

Use the *−f* option to change a users full name. Use this tool as either *chfn* or *chfn user_name* (usable by root only).

Command syntax:

```
chfn user_name
```

passwd

Changes the password of a user. You will need to be root if you want to change other users passwords.

Simply type *passwd* to change your own password or to change another users password type:

```
passwd username
```

Chapter 11. Text Related Tools

The text related tools chapter is the largest in this guide, most of the time on a GNU/Linux machine you will spend time interacting with text. This chapter briefly covers text editors and goes into more depth on viewing text, using tools to manipulate text, finding text within files and changing text formats between windows based systems and GNU/Linux based systems.

11.1. Text Editors

vi

A traditional UNIX system text editor, should be on any UNIX system. It requires learning a few key combinations, but is very powerful, and it is also quite small. vi is well known for its minimal use of resources.

vim

vim − vi improved. A newer version of the vulnerable *vi* editor. Many systems use *vim* rather than *vi*.

emacs

More than just a text editor. This text editor has a steep learning curve but is also very powerful, it is both advanced and quite large. *emacs* can do anything, surf the internet, chat, play games and many other tasks.

Others

There are too many different text editors to list here. Have a look on the internet, either search for them using any search engine or you will find many of them at Sourceforge or Freshmeat.

11.2. Text Viewing Tools

head

With no options it shows the first ten lines of a text file.

Use *head −n x* (where "x" is a number) to display the first x lines.

Try *head −F* to use a continually updated version of *head* (if the file changes it will be reloaded and displayed), please note that using this option will run *head* is a continuous loop so you'll need to use **CTRL−C** to exit.

For example:

```
head −n 20 somelog.txt
```

Will display the top 20 entries of the file "somelog.txt".

tail

With no options it shows the last ten lines of a file.

Use *tail −n x* (where "x" is a number) to display the last *x* lines.

Try *tail −F* to use a continually updated version of *tail* (if the file changes it will be reloaded and displayed), please note that using this option will run *tail* is a continuous loop so you'll need to use **CTRL−C** to exit.

For example:

```
tail −n 20 somelog.txt
```

Will display the last 20 entries of the file "somelog.txt".

less

Views text, can scroll backwards and forwards. Has many different options which are all described in the manual page.

When *less* is already running, use :n and :p (type a colon then the character) to move to the next and previous files (when there are multiple open files).

Command syntax:

```
less filename.txt
```

Or using a tool (in this example *cat*):

```
cat file.txt | less
```

more

Displays text, one page full at a time, more limited than *less*. In this case *less* is better than *more*.

```
more filename.txt
```

Or using a tool (is this example cat):

```
cat file.txt | more
```

cat

Combines (concatenates) multiple documents into one document. Can be used on individual files as well.

Some useful options:

- ◊ *−b* −−− number all non−blank lines
- ◊ *−n* −−− number all lines.

Also try using *nl* to number lines (it can do more complex numbering), you will find it under under this section, Section 11.4

Example:

```
cat filepart1 filepart2 filepart3 > wholefile.txt
```

This will combine (concatenate) filepart1, filepart2 and filepart3 into the single file "wholefile.txt".

tac

Combines (concatenates) multiple documents into one document and outputs them in reverse order. Can also be used on individual files. Notice that *tac* is *cat* written backwards.

Example:

```
tac filepart1 filepart2 filepart3 > wholefile.txt
```

This will combine (concatenate) filepart1, filepart2 and filepart3 into the single file but have each of the files written in reverse.

z* commands

Many commands can be prefixed with a "z" to read/work within a gzip compressed file.

Some examples are *zcat, zless, zmore, zgrep, zcmp, zdiff.*

There are many utilities for working with text within compressed files without trying to manually de−compress them somewhere first...most begin with a "z". You will find some of them mentioned over here, Section 15.3.

bz* commands
: There are also a few commands that prefixed with a "bz" to read/work within a file compressed with bzip2.

  The tools are *bzcat, bzless, bzgrep.* You will find some of them mentioned over here, Section 15.3.

11.3. Text Information Tools

wc
: Word count, count how many words you have in a text document. Can also be used to count the lines or bytes within the file.

  Use the options *−w* for words, *−l* for lines and *−c* for bytes. Or simply run *wc* with no options to get all three.

  Command syntax:

```
wc −option file.txt
```

style
: To run various readability tests on a particular text file. Will output scores on a number of different readability tests (with no options).

  Command syntax:

```
style −options text_file
```

Find style in the diction package

This command is part of the diction package and does not appear to be used too often these days

cmp
: Determines whether or not two files differ, works on any type of file. Very similar to *diff* only it compares on the binary level instead of just the text.

diff
: Compares two text files and output a difference report (sometimes called a "diff") containing the text that differs between two files.

  Can be used to create a 'patch' file (which can be used by *patch*).

  Example:

```
diff file1.txt file2.txt
```

diff will output a '>' (followed by the line) for each line that isn't in the first file but is in the second file, and it will output a '<' (followed by the line) for each line that is in the first file but not in the second file.

sdiff
: Instead of giving a difference report, it outputs the files in two columns, side by side, separated by spaces.

diff3
: Same as *diff* except for three files.

comm
: Compares two files, line−by−line and prints lines that are unique to file1 (1st column), unique to file2 (2nd column) and common to both files (3rd column).

  Use *comm* with the −1, −2, or −3 to suppress the printing of those particular lines. Simply run *comm* to have all three listed (ie. unique to files 1 and 2 and common to both).

  Command syntax:

```
comm file1 file2
```

look
: To output a list of words in the system dictionary that begin with a given string −− this is useful for finding words that begin with a particular phrase or prefix.

  Give the string as an argument; it is not case sensitive.

  Command syntax:

```
look string
```

11.4. Text manipulation tools

Also see

Also see *tac*, and *cat* over in this section, Section 11.2, as they can perform text manipulation too

sort
: Sorting text with no options the sort is alphabetical. Can be run on text files to sort them alphabetically (note it also concatenates files), can also be used with a pipe '|' to sort the output of a command.

  Use *sort −r* to reverse the sort output, use the *−g* option to sort 'numerically' (ie read the entire number, not just the first digit).

  Examples:

```
cat shoppinglist.txt | sort
```

  The above command would run *cat* on the shopping list then sort the results and display them in alphabetical order.

```
sort −r shoppinglist.txt
```

  The above command would run *sort* on a file and *sort* the file in reverse alphabetical order.

  Advanced sort commands:

sort is a powerful utility, here are some of the more hard to learn (and lesser used) commands. Use the *−t* option to use a particular symbol as the separator then use the *−k* option to specify which column you would like to sort by, where column 1 is the first column *before* the separator. Also use the *−g* option if numeric sorting is not working correctly (without the −g option sort just looks at the first digit of the number). Here is a complex example:

```
sort −t : −k 4 −k 1 −g /etc/passwd | more
```

This will sort the "/etc/passwd" file, using the colon ':' as the separator. It will sort via the 4th column (GID section, in the file) and then sort within that sort using the first (name) to resolve any ties. The *−g* is there so it sorts via full numbers, otherwise it will have 4000 before 50 (it will just look at the first digit...).

join

Will put two lines together assuming they share at least one common value on the relevant line. It won't print lines if they don't have a common value.

Command syntax:

```
join file1 file2
```

cut

Prints selected parts of lines (of a text file), or, in other words, removes certain sections of a line. You may wish to remove things according to tabs or commas, or anything else you can think of...

Options for *cut:*

◊ −d −−− allows you to specify another delimiter, for example ':' is often used with /etc/passwd:
```
cut −d ':' (and probably some more options here) /etc/passwd
```
◊ *−f* −−− this option works with the text by columns, separated according to the delimiter. For example if your file had lines like "result,somethingelse,somethingelse" and you only wanted result you would use:
```
cut −d ',' −f 1 /etc/passwd
```
This would get you only the usernames in /etc/passwd

◊ "," (commas) −−− used to separate numbers, these allow you to cut particular columns. For example:
```
cut −d ':' −f 1,7 /etc/passwd
```
This would only show the username and the shell that each person is setup for in /etc/passwd.

◊ "−" (hyphen) −−− used to show from line x to line y, for example 1−4, (would be from lines 1 to line 4).
```
cut −c 1−50 file1.txt
```
This would cut (display) characters (columns) 1 to 50 of each line (and anything else on that line is ignored)

◊ −x −−− where x is a number, to cut from line 1 to "x"

◊ x− −−− where x is a number, to cut from "x" to the end.
```
cut −5, 20−, 8 file2.txt
```
This would display ("cut") characters (columns) 1 to 5, 8 and from 20 to the end.

ispell/aspell

To spell check a file interactively, prompts for you to replace word or continue. *aspell* is said to be better at suggesting replacement words, but its probably best to find out for yourself.

aspell example:

```
aspell −c FILE.txt
```

This will run *aspell* on a particular file called "FILE.txt", *aspell* will run interactively and prompt for user input.

ispell example:

```
ispell FILE.txt
```

This will run *ispell* on a particular file called "FILE.txt" *ispell* will run interactively and prompt for user input.

chcase

Is used to change the uppercase letters in a file name to lowercase (or vice versa).

You could also use *tr* to do the same thing...

```
cat fileName.txt | tr '[A−Z]' '[a−z]'  > newFileName.txt
```

The above would convert uppercase to lowercase using the the file "fileName.txt" as input and outputting the results to "newFileName.txt".

```
cat fileName.txt | tr '[a−z]' '[A−Z]' > newFileName.txt
```

The above would convert lowercase to uppercase using the the file "fileName.txt" as input and outputting the results to "newFileName.txt".

chcase (a perl script) can be found at the chcase homepage.

fmt

(format) a simple text formatter. Use *fmt* with the *−u* option to output text with "uniform spacing", where the space between words is reduced to one space character and the space between sentences is reduced to two space characters.

Example:

```
fmt −u myessay.txt
```

Will make sure the amount of space between sentences is two spaces and the amount of space between words is one space.

paste

Puts lines from two files together, either lines of each file side by side (normally separated by a tab−stop but you can have any symbols(s) you like...) or it can have words from each file (the first file then the second file) side by side.

To obtain a list of lines side by side, the first lines from the first file on the left side separated by a tab−stop then the first lines from the second file. You would type:

```
paste file1.txt file2.txt
```

To have the list displayed in serial, first line from first file, [Tab], second line from first file, then third and fourth until the end of the first file type:

```
paste −−serial file1.txt file2.txt
```

This command is very simple to understand if you make yourself an example

Its much easier if you create an example for yourself. With just a couple of lines, I used "first line first file" and "first line second file" et cetera for a quick example.

expand

Will convert tabs to spaces and output it. Use the option *−t num* to specify the size of a "tapstop", the number of characters between each tab.

Command syntax:

```
expand file_name.txt
```

unexpand

Will convert spaces to tabs and output it.

Command syntax:

```
unexpand file_name.txt
```

uniq

Eliminates duplicate entries from a file and it sometimes greatly simplifies the display.

uniq options:

◊ *−c* −−− count the number of occurances of each duplicate
◊ *−u* −−− list only unique entries
◊ *−d* −−− list only duplicate entries

For example:

```
uniq −cd phone_list.txt
```

This would display any duplicate entries only and a count of the number of times that entry has appeared.

tr

(translation). A filter useful to replace all instances of characters in a text file or "squeeze" the whitespace.

Example:

```
cat some_file | tr '3' '5' > new_file
```

This will run the *cat* program on some file, the output of this command will be sent to the *tr* command, *tr* will replace all the instances of 3 with 5, like a search and replace. You can also do other things such as:

```
cat some_file | tr '[A−Z]' '[a−z]' > new_file
```

This will run *cat* on some_file and convert any capital letters to lowercase letters (you could use this to change the case of file names too...).

Alternatives

You can also do a search and replace with a one line Perl command, read about it at the end of this section.

nl

The number lines tool, it's default action is to write it's input (either the file names given as an argument, or the standard input) to the standard output.

Line numbers are added to every line and the text is indented.

This command can do take some more advanced numbering options, simply read the info page on it.

These advanced options mainly relate to customisation of the numbering, including different forms of separation for sections/pages/footers etc.

Also try *cat −n* (number all lines) or *cat −b* (number all non−blank lines). For more info on *cat* check under this section: Section 11.2

There are two ways you can use *nl*:

```
nl some_text_file.txt
```

The above command would add numbers to each line of some_text_file. You could use *nl* to number the output of something as shown in the example below;

```
grep some_string some_file | nl
```

Perl search and replace text

To search and replace text in a file is to use the following one−line Perl command[4]:

```
$ perl -pi -e "s/oldstring/newstring/g;" filespec [RET]
```

In this example, "oldstring" is the string to search, "newstring*"* is the string to replace it with, and "filespe*c*" is the name of the file or files to work on. You can use this for more than one file.

Example: To replace the string "helpless" with the string "helpful" in all files in the current directory, type:

```
$ perl -pi -e "s/helpless/helpful/g;" * [RET]
```

Also try using *tr* to do the same thing (see further above in this section).

If these tools are too primitive

If these text tools are too simple for your purposes then you are probably looking at doing some programming or scripting.

If you would like more information on bash scripting then please see the advanced bash scripting guide, authored by Mendel Cooper.

sed and awk are traditional UNIX system tools for working with text, this guide does not provide an explanation of them. sed works on a line−by−line basis performing substitution and awk can perform a similar task or assist by working on a file and printing out certain information (its a programming language).

You will normally find them installed on your GNU/Linux system and will find many tutorials all over the internet, feel free to look them up if you ever have to perform many similar operations on a text file.

11.5. Text Conversion/Filter Tools

Filters (UNIX System/dos formats)

The following filters allow you to change text from Dos−style to UNIX system style and vice−versa, or convert a file to other formats. Also note that many modern text editors can do this for you...

Why use filters?

Because UNIX systems and Microsoft use two different standards to represent the end−of−line in an ASCII text file.

This can sometimes causes problems in editors or viewers which aren't familiar with the other operating systems end−of−line style. The following tools allow you to get around this difference.

Whats the difference?

The difference is very simple, on a Windows text file, a newline is signalled by a carriage return followed by a newline, '\r\n' in ASCII.

On a UNIX system a newline is simply a newline, '\n' in ASCII.

dos2unix

This converts Microsoft−style end−of−line characters to UNIX system style end−of−line characters.

Simply type:

```
dos2unix file.txt
```

fromdos

This does the same as *dos2unix* (above).

Simply type:

```
fromdos file.txt
```

fromdos can be obtained from the from/to dos website.

unix2dos

This converts UNIX system style end−of−line characters to Microsoft−style end−of−line characters.

Simply type:

```
unix2dos file.txt
```

todos

This does the same as *unix2dos* (above).

Simply type:

```
todos file.txt
```

todos can be obtained from the from/to dos website.

antiword

This filter converts Microsoft word documents into plain ASCII text documents.

Simply type:

```
antiword file.doc
```

You can get *antiword* from the antiword homepage.

recode

Converts text files between various formats including HTML and dozens of different forms of text encodings.

Use *recode −l* for a full listing. It can also be used to convert text to and from Windows and UNIX system formats (so you don't get the weird symbols).

Warning

By default recode overwrites the input file, use '<' to use recode as a filter only (and to not overwrite the file).

Examples:

UNIX system text to Windows text:

```
recode ..pc file_name
```

Windows text to UNIX system text:

```
recode ..pc/ file_name
```

UNIX system text to Windows text without overwriting the original file (and creating a new output file):

```
recode ..pc < file_name > recoded_file
```

tr

(Windows to UNIX system style conversion only). While *tr* is not specifically designed to convert files from Windows−format to UNIX system format by doing:

```
tr -d '\r' < inputFile.txt > outputFile.txt
```

The −d switch means to simply delete any occurances of the string. Since we are looking for '\r', carriage returns it will remove any it finds, making the file a UNIX system text file. You can read more about *tr* over here, Section 11.4.

11.5.1. Conversion tools

enscript

Converts text files to postscript, rtf, HTML (use *ghostview* to view the postscript file). *enscript* has a large number of options which can be used to customise the output.

Examples:[5]

```
enscript --language=html input_file.txt -o output_file.html
```

This will take some file and output it as a html file.

```
enscript --help-highlight
```

Display help on using the highlight feature (list all different types of highlighting available)

```
enscript --help-highlight
```

Highlight (pretty print), example:

```
enscript -E --color --language=html --toc --output=foo.html *.h *.c
```

Add all the files with a .h and a .c (C source and header files) into a file called foo.html, use colour and add a table of contents

For further options refer to the well written manual page of enscript.

figlet

Used to create ASCII "art". Figlet can create several different forms (fonts) of ASCII art, its one of the more unusual programs around.

11.6. Finding Text Within Files

grep

Looks for text within files. For example:

```
grep this_word this_file.txt
```

Example options:

◊ *−v* −−− this option is used to display lines which do not contain the string.
◊ *−n* −−− this option displays the line numbers
◊ *−w* −−− this option makes grep match the whole word
◊ *−A x* or *−B* x (where x is a number) −−− display "x" lines After or Before the section where the particular word is found.
◊ *−r* or *rgrep* −−− search for text within files recursively.

This command uses regular expressions, for more information please see, Section 20.4.2.

For example, this command would look in the file "rpmlist.txt" for anything starting with "rpm":

```
grep rpm rpmlist.txt
```

Or you could use it like this, to search through the output of another file:

```
rpm −qa | grep ogg
```

The first command lists all RPM's installed on your system, the second finds any containing the string "ogg" and outputs them.

rgrep

A "recursive" version of *grep* (this is a different program to *grep*). This will search all the files in the current directory and all it's subdirectories and print the names of the files and the matching line. Follows similar syntax to *grep* (see above). You could also use *grep* with the *−r* option to achieve the same affect.

fgrep

This version of *grep* calls *grep* with the *−F* option. This will look for literal strings only, it won't use or expand any kind of regular expression.

For example you could type:

```
fgrep 'a$*b?' file.txt
```

And *fgrep* would look for the string "a$*b?" in the file "file.txt".

Other Versions

There are various versions of grep which are designed to do different things try searching for them on the internet or within your distribution.

Chapter 12. Mathematical tools

num−utils homepage

The "num−utils" homepage, Num Utils, contains a variety of command line programs that could be useful when performing maths on your GNU/Linux machine.

units

Convert units of measurement between different scales. For example, centimeters to inches, litres to gallons.

Simply run the program, I recommend running it as follows:

```
units --verbose
```

This will run the program and it will tell you exactly what it is doing.

Example: you enter "60 meters" then you want it worked out in "kilometers". The first line will tell you what this evaluates to.

If you wanted the conversion rate for "meters" to "kilometers" read the second line of the output (which will tell you meters/1000).

To exit

Press **CTRL−D** (end−of−file key) when you are finished using *units*.

python

Python is a very powerful, easy to learn, general purpose, interpreted programming language. And it makes a great calculator! If you don't have a calculator installed then simply type *python*, then hit [Enter].

This will execute the Python interpreter in interactive mode. Type your sums just like you would use a calculator. Note that if you want to work out fractions make sure you use a decimal point and a zero to obtain the correct answer (otherwise it will use integer division).

To start python in interactive mode, simply type:

```
python
```

Once python is started you can use it to add up sums or maybe do some python programming.

Use **CTRL−D** (end−of−file key) to exit the Python interpreter.

numgrep

A little bit like grep only this is designed for numbers only.

Use '/' (forward slashes) to contain each expression.

Use m<n> to find multiples of the number n and use f<n> to find factors of the number n.

Use commas to seperate expressions and .. (two dots) to represent a range.

For example, to input from standard input you could simply type:

```
numgrep
```

To input from a file and look for numbers between 1 and 1000 you could type:

```
numgrep /1..1000/ file_name
```

This tool comes from the num−utils package

Please note that this tool is part of the num−utils package.

Chapter 13. Network Commands

The network commands chapter explains various tools which can be useful when networking with other computers both within the network and accross the internet, obtaining more information about other computers. This chapter also includes information on tools for network configuration, file transfer and working with remote machines.

netstat
: Displays contents of /proc/net files. It works with the Linux Network Subsystem, it will tell you what the status of ports are ie. open, closed, waiting, masquerade connections. It will also display various other things. It has many different options.

tcpdump
: This is a sniffer, a program that captures packets off a network interface and interprets them for you. It understands all basic internet protocols, and can be used to save entire packets for later inspection.

ping
: The ping command (named after the sound of an active sonar system) sends echo requests to the host you specify on the command line, and lists the responses received their round trip time.

  You simply use ping as:

```
ping ip_or_host_name
```

Note to stop ping (otherwise it goes forever) use **CTRL−C** (break).

Please note

Using ping/smbmount/ssh or other UNIX system programs with a computer name rather than IP address will only work if you have the computer listed in your /etc/hosts file. Here is an example:

```
192.168.1.100 new
```

This line says that their is a computer called "new" with IP address 192.168.1.100. Now that it exists in the /etc/hosts file I don't have to type the IP address anymore, just the name "new".

hostname
: Tells the user the host name of the computer they are logged into. Note: may be called *host.*

traceroute
: *traceroute* will show the route of a packet. It attempts to list the series of hosts through which your packets travel on their way to a given destination. Also have a look at *xtraceroute* (one of several graphical equivalents of this program).

  Command syntax:

```
traceroute machine_name_or_ip
```

tracepath
: *tracepath* performs a very simlar function to *traceroute* the main difference is that *tracepath* doesn't take complicated options.

  Command syntax:

```
tracepath machine_name_or_ip
```

findsmb

findsmb is used to list info about machines that respond to SMB name queries (for example windows based machines sharing their hard disk's).

Command syntax:

```
findsmb
```

This would find all machines possible, you may need to specify a particular subnet to query those machines only...

nmap

" network exploration tool and security scanner". *nmap* is a very advanced network tool used to query machines (local or remote) as to whether they are up and what ports are open on these machines.

A simple usage example:

```
nmap machine_name
```

This would query your own machine as to what ports it keeps open. *nmap* is a very powerful tool, documentation is available on the nmap site as well as the information in the manual page.

13.1. Network Configuration

ifconfig

This command is used to configure network interfaces, or to display their current configuration. In addition to activating and deactivating interfaces with the "up" and "down" settings, this command is necessary for setting an interface's address information if you don't have the *ifcfg* script.

Use *ifconfig* as either:

```
ifconfig
```

This will simply list all information on all network devices currently up.

```
ifconfig eth0 down
```

This will take eth0 (assuming the device exists) down, it won't be able to receive or send anything until you put the device back "up" again.

Clearly there are a lot more options for this tool, you will need to read the manual/info page to learn more about them.

ifup

Use *ifup device−name* to bring an interface up by following a script (which will contain your default networking settings). Simply type *ifup* and you will get help on using the script.

For example typing:

```
ifup eth0
```

Will bring eth0 up if it is currently down.

ifdown

Use *ifdown device−name* to bring an interface down using a script (which will contain your default network settings). Simply type *ifdown* and you will get help on using the script.

For example typing:

```
ifdown eth0
```

Will bring eth0 down if it is currently up.

ifcfg

Use *ifcfg* to configure a particular interface. Simply type ifcfg to get help on using this script.

For example, to change eth0 from 192.168.0.1 to 192.168.0.2 you could do:

```
ifcfg eth0 del 192.168.0.1
ifcfg eth0 add 192.168.0.2
```

The first command takes eth0 down and removes that stored IP address and the second one brings it back up with the new address.

route

The *route* command is the tool used to display or modify the routing table. To add a gateway as the default you would type:

```
route add default gw some_computer
```

13.2. Internet Specific Commands

Note that should DNS not be configured correctly on your machine, you need to edit "/etc/resolv.conf" to make things work...

host

Performs a simple lookup of an internet address (using the Domain Name System, DNS). Simply type:

```
host ip_address
```

or

```
host domain_name
```

dig

The "domain information groper" tool. More advanced then *host*... If you give a hostname as an argument to output information about that host, including it's IP address, hostname and various other information.

For example, to look up information about "www.amazon.com" type:

```
dig www.amazon.com
```

To find the host name for a given IP address (ie a reverse lookup), use *dig* with the *`−x'* option.

```
dig −x 100.42.30.95
```

This will look up the address (which may or may not exist) and returns the address of the host, for example if that was the address of "http://slashdot.org" then it would return "http://slashdot.org".

dig takes a huge number of options (at the point of being too many), refer to the manual page for more information.

whois

(now BW whois) is used to look up the contact information from the "whois" databases, the servers are only likely to hold major sites. Note that contact information is likely to be hidden or restricted as it is often abused by crackers and others looking for a way to cause malicious damage to organisation's.

wget

(GNU Web get) used to download files from the World Wide Web.

To archive a single web−site, use the *−m* or *−−mirror* (mirror) option.

Use the *−nc* (no clobber) option to stop *wget* from overwriting a file if you already have it.

Use the *−c* or *−−continue* option to continue a file that was unfinished by wget or another program.

Simple usage example:

```
wget url_for_file
```

This would simply get a file from a site.

wget can also retrieve multiple files using standard wildcards, the same as the type used in bash, like *, [ ], ?. Simply use *wget* as per normal but use single quotation marks (' ') on the URL to prevent bash from expanding the wildcards. There are complications if you are retrieving from a http site (see below...).

Advanced usage example, (used from *wget* manual page):

```
wget --spider --force-html -i bookmarks.html
```

This will parse the file bookmarks.html and check that all the links exist.

Advanced usage: this is how you can download multiple files using http (using a wildcard...).

Notes: http doesn't support downloading using standard wildcards, ftp does so you may use wildcards with ftp and it will work fine. A work−around for this http limitation is shown below:

```
wget -r -l1 --no-parent -A.gif http://www.website.com[6]
```

This will download (recursively), to a depth of one, in other words in the current directory and not below that. This command will ignore references to the parent directory, and downloads anything that ends in ".gif". If you wanted to download say, anything that ends with ".pdf" as well than add a *−A.pdf* before the website address. Simply change the website address and the type of file being downloaded to download something else. Note that doing *−A.gif* is the same as doing *−A "*.gif"* (double quotes only, single quotes will not work).

wget has many more options refer to the examples section of the manual page, this tool is very well documented.

Alternative website downloaders

You may like to try alternatives like httrack. A full GUI website downloader written in python and available for GNU/Linux

curl

curl is another remote downloader. This remote downloader is designed to work without user interaction and supports a variety of protocols, can upload/download and has a large number of tricks/work−arounds for various things. It can access dictionary servers (dict), ldap servers, ftp, http, gopher, see the manual page for full details.

To access the full manual (which is huge) for this command type:

```
curl −M
```

For general usage you can use it like *wget*. You can also login using a user name by using the *−u* option and typing your username and password like this:

```
curl −u username:password http://www.placetodownload/file
```

To upload using ftp you the *−T* option:

```
curl −T file_name ftp://ftp.uploadsite.com
```

To continue a file use the *−C* option:

```
curl −C − −o file http://www.site.com
```

13.3. Remote Administration Related

ssh

Secure shell, remotely login on a machine running the *sshd* daemon. Once you are logged in you have a secure shell and are able to execute various commands on that computer such as copy files, reboot the computer, just like it was your own GNU/Linux PC.

Or you can use *ssh* with a full hostname to connect to a remote machine (as in across the internet).

Examples:

```
ssh hostname
```

Connect to a remote system with your current username, you will obviously need the password of the user on the other machine.

```
ssh username@hostname
```

Connect to a remote system with your a different username, you will obviously need the password of the user on the other machine.

scp

Secure copy, part of the ssh package. Allows you to copy files from one computer to another computer, use *−r* to copy recursively (copy entire directories and subdirectories).

scp's syntax is always

```
scp machineToBeCopiedFrom machineToBeCopiedTo
```

Where either machine can be a local directory (on the current filesystem /) or a remote machine. Remote machines are usually *machinesFullName:/directory* (if you omit the directory part it will just assume the home directory of the username you are logging in with).

The example below copies all files from the current directory (not including any directories), the command will login to "new" using the username of the person currently logged in on the local computer, the files will be copied to the root directory of the remote computer called "new" (which is probably on the LAN):

```
scp * new:/
```

You could also copy files from another computer to another computer. Let's say you are on a computer called "p100". And you want to copy files (and directories) from "hp166" (in the /tmp directory and anything below that) to "new" and put the files in new's temporary directory. You could do:

```
scp -r hp166:/tmp new:/tmp
```

Assuming you were logged in as "fred" you would need passwords for user "fred" on the computers hp166 and new. Add an *user_name@* before the computer name to login under a different user name.

For example to perform the above command with user "root" on hp166 and "anon" on new you would type:

```
scp -r root@hp166:/tmp anon@new:/tmp
```

To copy from a remote machine to a local computer you simply do things in reverse:

```
scp remoteMachine:/mystuff/* .
```

This will copy files on the remote machine in the directory "mystuff" to your local computer.

Remote Machines

Please note that when working with a remote machine you need to have a : (colon) after the machine name even if you want the files in their home directory. Otherwise the command will fail.

sftp

Secure ftp, another part of the ssh package. This command is similar to ftp but uses an encrypted tunnel to connect to an ftp server and is therefore more secure than just plain *ftp*.

The command usage is very similar to *ftp* (the command−line tool), *sftp* (once running) uses commands such as *help* (for help), *put* (send files to the server), *get* (download files from the server) and various others, refer to the manual page and internal documentation for further details.

Graphical programs

Sometimes its easier to manage files with a GUI, many of these programs do have good GUI equivalents, try searching the internet or sites like Sourceforge or Freshmeat.

Chapter 14. Security

The security chapter is designed to give the user a very basic level of understanding of security within the GNU/Linux operating system. This chapter also has information on the UNIX system style file permissions used on most GNU/Linux machines.

More comprehensive guides can be found at the Linux Documentation Project, such as the Linux Security howto authored by Kevin Fenzi and Dave Wreski.

There are also sites such as Linux Security. If your looking for a program to assist in locking down your operating system you may want to check Bastille Linux that runs on RPM based distributions (Redhat/Mandriva/SuSE).

Changing root's password

This trick works well if you have forgotten your superuser password, type *linux single* at a LILO/Grub prompt. Then *passwd* once the system has started and you are at a console.

Grub:

If you are using grub go to the relevant line (the one with the kernel and various options) then press 'e' for edit and add "single" on to the end of the lines that boot the kernel. Then hit [Enter] and press 'b' (to boot).

Lilo:

If you are using lilo press escape and type " linux single" and then hit [Enter] to boot.

Security Warning

This is also a basic security hazard if you have others using your computer and security is a concern, you may like to add a password to your LILO or Grub prompt to stop this from being done.

umask

The umask is a value set by the shell. It controls the default permissions of any file created during that shell session. This information is inherited from the shell's parent and is normally set in some configuration file by the root user (in my case /etc/profile).

umask has an unusual way of doing things ...to set the umask you must describe file permissions by saying what will be disabled.

You can do this by doing 777 minus the file permissions you want. Note that *umask* works with numbers only, for an explanation please see, Section 14.2

For example:

You want the default during a particular shell session to be equivalent to *chmod 750* (user has r/w/x, group has r/x and other has no permissions), then the command you would use would be:

```
umask 027
```

14.1. Some basic Security Tools

md5sum

Compute an md5 checksum (128−bit) for file "file_name" to verify it's integrity. You normally use the " md5sum −c" option to check against a given file (often with a ".asc" extention) to check whether the various files are correct, this comes in handy when downloading isos as the checking is automated for you.

Command syntax:

```
md5sum file_name
```

mkpasswd −l 10

This command will make a random password of length ten characters. This password generator creates passwords that are designed to be hard to guess. There are similar alternatives to this program scattered around the internet.

14.2. File Permissions

Use *ls −l* to see the permissions of files (list−long). They will appear like this, note that I have added spaces between permissions to make it easier to read:

Where: r = read, w = write, x = execute

```
  -  rwx   rw-   r--  1 ❶ newuser newuser
type❷owner❸group❹others❺
```

❶

This number is the number of hard links (pointers) to this file. You can use *ln* to create another hard−link to the file.

❷

This is the type of file. '−' means a regular file, 'd' would mean a directory, 'l' would mean a link. There are also other types such as 'c' for character device and 'b' for block device (found in the /dev/ directory).

❸

These are the permissions for the owner of the file (the user who created the file).

❹

These are the permissions for the group, any users who belong is the same group as the user who created the file will have these permissions.

❺

These are the permissions for everyone else. Any user who is outside the group will have these permissions to the file.

The two names at the end are the username and group respectively.

chmod

Change file access permissions for a file(s).

There are two methods to change permissions using *chmod*; letters or numbers.

Letters Method:

use a + or − (plus or minus sign) to add or remove permissions for a file respectively. Use an equals sign =, to specify new permissions and remove the old ones for the particular type of user(s).

You can use *chmod letter* where the letters are:

a (all (everyone)), *u* (user), *g* (group) and *o* (other).

Examples:

```
chmod u+rw somefile
```

This would give the user read and write permission.

```
chmod o−rwx somefile
```

This will remove read/write/execute permissions from other users (doesn't include users within your group).

```
chmod a+r somefile
```

This will give everyone read permission for the file.

```
chmod a=rx somefile
```

This would give everyone execute and read permission to the file, if anyone had write permission it would be removed.

Numbers Method:

you can also use numbers (instead of letters) to change file permissions. Where:

r (read) = 4 w (write) = 2 x (execute) = 1

Numbers can be added together so you can specify read/write/execute permissions; read+write = 6, read+execute = 5, read+write+execute = 7

Examples:

```
chmod 777 somefile
```

This would give everyone read/write/execute permission on "this_file". The first number is user, second is group and third is everyone else (other).

```
chmod 521 somefile
```

This would give the user read and execute permission, and the group write permission (but not read permission!) and everyone else execute permission. (Note that it's just an example, settings like that don't really make sense...).

chown

Changes the ownership rights of a file (hence the name 'chown' − change owner). This program can only be used by root.

Use the *−R* option to change things recursively, in other words, all matching files including those in subdirectories.

Command syntax:

```
chown owner:group the_file_name
```

sticky bit

Only the person who created the file within a directory may delete it, even if other people have write permission. You can turn it on by typing:

```
chmod 1700 somedirectory (where 1 = sticky bit)
```

or (where *t* represents the sticky bit)

```
chmod +t somedirectory
```

To turn it off you would need to type:

```
chmod 0700 somefile (where the zero would mean no sticky bit)
```

or (where *t* represents the sticky bit)

```
chmod −t somefile
```

Note that the permissions aren't relevant in the numbers example, only the first number (1 = on, 0 = off).

An example of a sticky directory is usually /tmp

suid

Allow SUID/SGID (switch user ID/switch group ID) access. You would normally use *chmod* to turn this on or off for a particular file, suid is generally considered a security hazard so be careful when using this.

Example:

```
chmod u+s file_name
```

This will give everyone permission to execute the file with the permissions of the user who set the +s switch.

Security Hazard

This is obviously a security hazard. You should avoid using the suid flag unless necessary.

chattr

Change file system attributes (works on ext2fs and possibly others...). Use the *−R* option to change files recursively, *chattr* has a large number of attributes which can be set on a file, read the manual page for further information.

Example:

```
chattr +i /sbin/lilo.conf[7]
```

This sets the 'immutable' flag on a file. Use a '+' to add attributes and a '−' to take them away. The +i will prevent any changes (accidental or otherwise) to the "lilo.conf" file. If you wish to modify the lilo.conf file you will need to unset the immutable flag: *chattr −i*. Note some flags can only be used by root; *−i*, *−a* and probably many others.

Note there are many different attributes that chattr can change, here are a few more which may be useful:

◊ A (no Access time) −−− if a file or directory has this attribute set, whenever it is accessed, either for reading of for writing, it's last access time will not be updated. This can be useful, for example, on files or directories which are very often accessed for reading, especially since this parameter is the only one which changes on an inode when it's opened.
◊ a (append only) −−− if a file has this attribute set and is open for writing, the only operation possible will be to append data to it's previous contents. For a directory, this means that you can only add files to it, but not rename or delete any existing file. Only root can set or clear this attribute.
◊ s (secure deletion) −−− when such a file or directory with this attribute set is deleted, the blocks it was occupying on disk are written back with zeroes (similar to using *shred*). Note that this does work on the ext2, and ext3 filesystems but is unlikely to work on others (please see the documentation for the filesystem you are using). You may also like to see *shred*, please see Chapter 7

lsattr

(list attributes). This will list if whether a file has any special attributes (as set by chattr). Use the *−R* option to list recursively and try using the *−d* option to list directories like other files rather than listing their contents.

Command syntax:

```
lsattr
```

This will list files in the current directory, you may also like to specify a directory or a file:

```
lsattr /directory/or/file
```

Chapter 15. Archiving Files

The archiving files chapter provides some basic information on the simple programs that you can use to archive files. You will often see these programs used when you try to install programs without using a package management tool.

This is not a backup guide

Please note that while *tar* is useful for regular purposes, and possibly combined with bash sciprting or similar it can become useful, it is not a great program for performing real backups of data.

You should try searching the internet if you are looking for backup programs on GNU/Linux or try Sourceforge or Freshmeat for programs that you can use. You may also like to see rsync, Section 15.2.

15.1. tar (tape archiver)

Type *tar* then *−option(s)*

Options list:

- −c −−− create.
- −v −−− verbose, give more output, show what files are being worked with (extracted or added).
- −f −−− file (create or extract from file) − should always be the last option otherwise the command will not work.
- −z −−− put the file though gzip or use gunzip on the file first.
- −x −−− extract the files from the tarball.
- −p −−− preserves dates, permissions of the original files.
- −j −−− send archive through bzip2.
- −−exclude=pattern −−− this will stop certain files from being archived (using a standard wild−card pattern) or a single file name.

tar examples

```
tar −cvpf name_of_file.tar files_to_be_backed_up
```

This would create a tape archive (no compressing).

```
tar −zxvpf my_tar_file.tar.gz
```

This would extract files (verbosely) from a gzipped tape archive.

15.2. rsync

rsync

rsync is a replacement for the old *rcp* (remote−copy) command. It can use *ssh* for encryption and is a very flexible tool, it can copy from local machine to local machine, from local to remote (and vice−versa), and to and from rsync servers.

rsync uses an advanced differencing algorithm, so when to copies or syncs something it will (a) only copy new/changed files and (b) if the files have being changed it will copy the differences between the files (not the entire file). Using this method *rsync* saves time and bandwidth.

rsync also has advanced exclusion options similar to GNU tar. *rsync* has a well written manual page, for further information read the *rsync* documentation online or type:

```
man rsync
```

If you wish to visit the rsync site you will find it over here

15.3. Compression

There are two main compression utilities used in GNU/Linux. It's normal to first "tar" a bunch of files (using the *tar* program of course) and then compress them with either *bzip2* or *gzip*. Of course either of these tools could be used without tar, although they are not designed to work on more than one file (they use the UNIX tools philosophy, let *tar* group the files, they will do the compression...this simplifies their program). It's normal to use *tar* and then use these tools on them, or use *tar* with the correct options to use these compression programs.

GNU zip (gzip)

gzip is the GNU zip compression program and probably the most common compression format on UNIX−like operating systems.

```
gzip your_tar_file.tar
```

This will compress a tar archive with GNU zip, usually with a .gz extension. Gzip can compress any type of file, it doesn't have to be a tar archive.

```
gunzip your_file.gz
```

This will decompress a gzipped file, and leave the contents in the current directory.

bzip2

bzip2 is a newer compression program taht offers superior compression to gzip at the cost of more processor time.

```
bzip2 your_tar_file.tar
```

This will compress a tar archive with the bzip2 compression program, usually with a .bz extension. bzip2 can compress any type of file, it doesn't have to be a tar archive.

```
bunzip2 your_file.tar.bz2
```

This will decompress a file compressed by bzip2, and leave the contents in the current directory.

zipinfo

Use *zipinfo* to find detailed information about a zip archive (the ones usually generally used by ms−dos and windows, for example winzip).

Command syntax:

```
zipinfo zip_file.zip
```

zipgrep

Will run *grep* to look for files within a zip file (ms−dos style, for example winzip) without manually decompressing the file first.

Command syntax:

```
zipgrep pattern zip_file.zip
```

bzip2recover

Used to recover files from a damaged bzip2 archive. It simply extracts out all the working blocks as there own bzip2 archives, you can than use *bzip2 −t* on each file to test the integrity of them and extract the working files.

bzme

Will convert a file that is zipped or gzipped to a file compressed using *bzip2*.

Command syntax:

```
bzme filename
```

Tip

Both gzip and bzip2 supply tools to work within compressed files for example listing the files within the archive, running *less* on them, using *grep* to find files within the archive et cetera.

For gzip the commands are prefixed with z, *zcat, zless, zgrep*.

For bzip2 the commands are prefixed with bz, *bzcat, bzless, bzgrep*.

Chapter 16. Graphics tools (command line based)

The graphics tools chapter explains some image programs that can be called from the command−line. While I have found image programs that can be used from the command−line, zgv is the only one I've ever heard of, I did not find them very useful. All the tools listed use the X windowing system to work and simply run from the command line (so they can be scripted/automated if necessary).

montage

Creates a 'montage', an image created of many other images, arranged in a random fashion.

Command syntax:

```
montage r34.jpg r32.jpg skylines* skyline_images.miff
```

The above would create a "montage" of images (it would tile a certain number of images) into a composite image called "skyline_images.miff", you could always use *display* to view the image.

Note

Note that the images are converted to the same size (scaled) so they can be tiled together.

convert

To convert the file format of an image to another image format. *convert* is used to change a files format, for example from a jpeg to a bitmap or one of many other formats. *convert* can also manipulate the images as well (see the man page or the ImageMagick site).

Example from Jpeg to PNG format:

```
convert JPEG: thisfile.jpg PNG: thisfile.png
```

import

Captures screen−shots from the X server and saves them to a file. A screen−dump of what X is doing.

Command syntax:

```
import file_name
```

display

display is used to display (output) images on the screen. Once open you are can also perform editing functions and are able to read/write images. It has various interesting options such as the ability to display images as a slide show and the ability to capture screenshots of a single window on−screen.

Command syntax (for displaying an image):

```
display image_name
```

To display a slide show of images, open the images you want possibly using a wildcard, for example:

```
display *.jpg
```

And then click on the image to bring up the menu and then look under the miscellaneous menu for the slide show option.

Speed Warning

Be careful when opening multiple large sized images (especially on a slow machine) and putting the slide show on a small delay between image changes. Your processor will be overloaded and it will take a significant amount of time to be able to close ImageMagick.

identify

Will identify the type of image as well as it's size, colour depth and various other information. Use the *−verbose* option to show detailed information on the particular file(s).

Command syntax:

```
identify image_name
```

mogrify

mogrify is another ImageMagick command which is used to transform images in a number of different ways, including scaling, rotation and various other effects. This command can work on a single file or in batch.

For example, to convert a large number of tiff files to jpeg files you could type:

```
mogrify −format jpeg *.tiff
```

This command has the power to do a number of things in batch including making thumbnails of sets of images.

For this you could type:[8]

```
mogrify −geometry 120x120 *.jpg
```

showrgb

showrgb is used to uncompile an rgb colour−name database. The default is the one that X was built with. This database can be used to find the correct colour combination for a particular colour (well it can be used as a rough guide anyway).

To list the colours from the X database, simply type:

```
showrgb
```

Please note:

All tools listed, excluding *showrgb* are part of the ImageMagick package. Type *man ImageMagick* for a full list of available commands. Or see the ImageMagick site ImageMagick for further information.

Chapter 17. Working with MS−DOS files

Use the mtools programs to work with ms−dos based files, execute *mtools* for a full listing of available m* tools. There are a lot of files within the mtools package for working with ms−dos disks, also try the info documentation of mtools for more details.

The use of slashes

Note that with mtools commands you can use the slashes on the a: part either way (ie. backslash (windows−style) or forward slash (UNIX system style)).

mformat
: Formats an unmounted disk as an ms−dos floppy disk. Usage is similar to the ms−dos format utility, to format the first floppy disk you can type:

```
mformat a:
```

mcopy
: Copies files from an ms−dos disk when it's not mounted. Similar to the ms−dos copy command except it's more advanced.

Command syntax:

```
mcopy a:/file_or_files /destination/directory
```

mmount
: Mount an ms−dos disk, without using the normal UNIX system mount.

For example:

```
mmount a: /mnt/floppy
```

This will mount the floppy under /mnt/floppy (this option may or may not be necessary, it depends on your /etc/fstab setup).

mbadblocks
: Scans an ms−dos (fat formatted disk) for bad blocks, it marks any unused bad blocks as "bad" so they won't be used.

Example:

```
mbadblocks a:
```

dosfsck
: This program is used to check and repair ms−dos based filesystems. Use the *−a* option to automatically repair the filesystem (ie don't ask the user questions), the *−t* option to mark un−readable clusters as bad and the *−v* option to be more verbose (print more information).

Example:

```
dosfsck −at /dev/fd0
```

This would check your floppy disk for any errors (and bad sectors) and repair them automatically.

Chapter 18. Scheduling Commands to run in the background

There are two main tools used to perform scheduled tasks, *at* and *cron*. You may also like to try anacron if your computer does not run continuously, as cron will only work if your computer is left on (anacron can catch up with the scheduled tasks the next time the computer is on...).

at

'at' executes a command once on a particular day, at a particular time. *at* will add a particular command to be executed.

Examples:

```
at 21:30
```

You then type the commands you want executed then press the end−of−file key (normally **CTRL−D**). Also try:

```
at now + time
```

This will run at the current time + the hours/mins/seconds you specify (use *at now + 1 hour* to have command(s) run in 1 hour from now...)

You can also use the *−f* option to have at execute a particular file (a shell script).

```
at −f shell_script now + 1 hour
```

This would run the shell script 1 hour from now.

atq

Will list jobs currently in queue for the user who executed it, if root executes at it will list all jobs in queue for the at daemon. Doesn't need or take any options.

atrm

Will remove a job from the 'at' queue.

Command syntax:

```
atrm job_no
```

Will delete the job "job_no" (use *atq* to find out the number of the job)

cron

cron can be used to schedule a particular function to occur every minute, hour, day, week, or month.

It's normal to use the crontab to perform the editing functions as this automates the process for the cron daemon and makes it easier for normal users to use cron.

Anacron

anacron is another tool designed for systems which are not always on, such as home computers

While *cron* will not run if the computer is off, *anacron* will simply run the command when the computer is next on (it catches up with things).

crontab

crontab is used to edit, read and remove the files which the cron daemon reads.

Options for crontab (use *crontab −option(s)*):

◊ *−e* −−− to edit the file.
◊ *−l* −−− to list the contents of the file.
◊ *−u username* −−− use the *−u* with a username argument to work with another users crontab file.

When using *crontab −e* you have a number of fields (6) what they mean is listed below:

Field	Allowed Values
minute	0−59
hour	0−23
day of month	1−31
month	1−12 (or names, see below)
day of week	0−7 (0 or 7 is Sun, or use three letter names)

There are also a number of shortcut methods for common tasks, including:[9]

◊ *@reboot* −−− run command at reboot
◊ *@yearly* −−− same as 0 0 1 1 *
◊ *@annually* −−− same as @yearly
◊ *@monthly* −−− same as 0 0 1 * *
◊ *@weekly* −−− same as 0 0 * * 0
◊ *@daily* −−− same as 0 0 * * *
◊ *@midnight* −−− same as @daily
◊ *@hourly* −−− same as 0 * * * *

[10]

Note that * (asterisk) is used to mean anything (similar to the wildcard). For example if you leave the day part (the 5th place) with an asterisk it would mean everyday.

Lists are allowed. A list is a set of numbers (or ranges) separated by commas. Examples: ``1,2,5,9'', ``0−4,8−12''.

Step values can be used in conjunction with ranges. Following a range with ``/<number>'' specifies skips of the number's value through the range. For example, ``0−23/2'' can be used in the hours field to specify command execution every other hour (the alternative in the V7 standard is ``0,2,4,6,8,10,12,14,16,18,20,22''). Steps are also permitted after an asterisk, so if you want to say ``every two hours'', just use ``*/2''.

When writing a crontab entry you simply type in six fields separated by spaces, the first five are those listed in the table (using numbers or letters and numbers as appropriate), the 6th field is the command to be executed and any options, cron will read everything up until the newline.

Example:

```
5 4 * * sun echo "run at 5 after 4 every sunday"
```

This would run the echo command with the string shown at 4:05 every Sunday.

Chapter 19. Miscellaneous

The miscellaneous chapter contains commands that don't really fit into the other sections of this guide.

renaming extensions

To rename all of the files in the current directory with a '.htm' extension to '.html', type:

```
$ chcase -x 's/htm/html/' '*.htm'
```

You can get a copy of *the chcase* perl script *here.*

For more complex renaming you should read Section 7.3

rel[11]

Use rel to analyze text files for relevance to a given set of keywords. It outputs the names of those files that are relevant to the given keywords, ranked in order of relevance; if a file does not meet the criteria, it is not outputted in the relevance listing.

units man page

There is a man page, part of the Linux Programmers Manual called "units". It displays various information on the various scientific measurements (such as mega, giga et cetera). This manual page also has a short discussion about the argument over which standard should be used to measure data (ie. the kibibyte vs kilobyte).

To access this man page type:

```
man 7 units
```

fortune

fortune is a tool which will print a random, hopefully interesting quote or entertaining short piece of writing. There are options to customise which area the epigrams should come from. Just type *fortune* to get a random epigram from any section.

Simply type:

```
fortune
```

Chapter 20. Mini−Guides

The mini−guides chapter is a section of the document that describes certain concepts in more depth than the usual command descriptions. The information listed is fairly specific as I have tried to avoid the duplication of too much information that is already online.

20.1. RPM: Redhat Package Management System

Checking
Installed RPM's

Use the *rpm −V* option to check whether or not a package has been modified.

For example:

```
rpm −V textutils
```

If none of the files from the textutils package have changed then rpm will exit without outputting any data. If, on the other hand, the program has changed, you may see something like this:

```
U.5....T /bin/cat
```

This isn't as cryptic as it appears. The line returned from *rpm −V* contains any number of eight characters plus the full path to the file. Here are the characters and their meaning:[12]

- S −−− File size differs
- M −−− Mode differs (includes permissions and file type)
- 5 −−− MD5 sum differs
- D −−− Device major/minor number mis−match
- L −−− ReadLink(2) path mis−match
- U −−− User ownership differs
- G −−− Group ownership differs
- T −−− mTime differs

Mandriva Users Note

Mandriva Linux uses a customised version of RPM called urpmi (It consists of the urpm* commands, urpmi to install, urpme to remove and urpmf and urpmq to query).

This customised version has advantages over standard RPM, including automatic−dependency solving and Debian apt−get style functions (ability to download programs over the internet and have all dependencies resolved automatically).

The urpm* commands are all described in detail in Mandriva's documentation and various sources online.

20.2. Checking the Hard Disk for errors

Checking the hard disk for errors on your primary drive is very, very rarely required in GNU/Linux, most checking is automated on start−up if it is required. If you do need to check the hard disk for errors you will first need to unmount it. Then use the file system checker, *fsck*.

```
fsck.file_system_type
```

If you had an ext3 file−system then it would be:

```
fsck.ext3
```

Also try

You can also try using:

```
fsck −t file_system_type
```

20.3. Duplicating disks

This simple technique shows you how you would duplicate floppy disks in a GNU/Linux system using dd. This technique is not as useful as it used to be but can still be used for creating an image of a cd (although that is best done through the cd burning program).

This information has been taken from the Linux Online Classroom, see [4] in the *Bibliography* for further details.

```
$ dd if=/dev/fd0 of=floppy-image
$ dd if=floppy-image of=/dev/fd0
```

The first dd makes an exact image of the floppy to the file floppy−image, the second one writes the image to the floppy. (The user has presumably switched the floppy before the second command. Otherwise the command pair is of doubtful usefulness).

Similar techinques can be used when creating bootdisks, you simply use dd to transfer the image to the floppy disk.

20.4. Wildcards

Wildcards are useful in many ways for a GNU/Linux system and for various other uses. Commands can use wildcards to perform actions on more than one file at a time, or to find part of a phrase in a text file. There are many uses for wildcards, there are two different major ways that wildcards are used, they are globbing patterns/standard wildcards that are often used by the shell. The alternative is regular expressions, popular with many other commands and popular for use with text searching and manipulation.

Tip

If you have a file with wildcard expressions in it then you can use single quotes to stop bash expanding them or use backslashes (escape characters), or both.

For example if you wanted to create a file called 'fo*' (fo and asterisk) you would have to do it like this (note that you shouldn't create files with names like this, this is just an example):

```
  touch 'fo*'
```

Note that parts of both subsections on wildcards are based (at least in part) off the grep manual and info pages. Please see the *Bibliography* for further information.

20.4.1. Standard Wildcards (globbing patterns)

Standard wildcards (also known as globbing patterns) are used by various command−line utilities to work with multiple files. For more information on standard wildcards (globbing patterns) refer to the manual page by typing:

```
man 7 glob
```

Can be used by

Standard wildcards are used by nearly any command (including mv, cp, rm and many others).

? (question mark)
: this can represent any *single* character. If you specified something at the command line like "hd?" GNU/Linux would look for hda, hdb, hdc and every other letter/number between a−z, 0−9.

* (asterisk)
: this can represent any number of characters (including zero, in other words, zero or more characters). If you specified a "cd*" it would use "cda", "cdrom", "cdrecord" and *anything* that starts with "cd" also including "cd" itself. "m*l" could by mill, mull, ml, and anything that starts with an m and ends with an l.

[ ] (square brackets)
: specifies a range. If you did m[a,o,u]m it can become: mam, mum, mom if you did: m[a−d]m it can become anything that starts and ends with m and has any character a to d inbetween. For example, these would work: mam, mbm, mcm, mdm. This kind of wildcard specifies an "or" relationship (you only need one to match).

{ } (curly brackets)
: terms are separated by commas and each term must be the name of something or a wildcard. This wildcard will copy anything that matches either wildcard(s), or exact name(s) (an "or" relationship, one or the other).

  For example, this would be valid:

```
cp {*.doc,*.pdf} ~
```

  This will copy anything ending with .doc or .pdf to the users home directory. Note that spaces are not allowed after the commas (or anywhere else).

[!]
: This construct is similar to the [ ] construct, except rather than matching any characters inside the brackets, it'll match any character, as long as it is not listed between the [and]. This is a logical NOT. For example *rm myfile[!9]* will remove all myfiles* (ie. myfiles1, myfiles2 etc) but won't remove a file with the number 9 anywhere within it's name.

\ (backslash)
: is used as an "escape" character, i.e. to protect a subsequent special character. Thus, "\\" searches for a backslash. Note you may need to use quotation marks and backslash(es).

20.4.2. Regular Expressions

Regular expressions are a type of globbing pattern used when working with text. They are used for any form of manipulation of multiple parts of text and by various programming languages that work with text. For more information on regular expressions refer to the manual page or try an online tutorial, for example IBM Developerworks using regular expressions. For the manual page type:

Type:

```
man 7 regex
```

Regular expressions can be used by

Regular Expressions are used by *grep* (and can be used) by *find* and many other programs.

Tip

If your regular expressions don't seem to be working then you probably need to use single quotation marks over the sentence and then use backslashes on every single special character.

. (dot)
: will match *any single character*, equivalent to ? (question mark) in standard wildcard expressions. Thus, "m.a" matches "mpa" and "mea" but not "ma" or "mppa".

\ (backslash)
: is used as an "escape" character, i.e. to protect a subsequent special character. Thus, "\\" searches for a backslash. Note you may need to use quotation marks and backslash(es).

.* (dot and asterisk)
: is used to match any string, equivalent to * in standard wildcards.

* (asterisk)
: the proceeding item is to be matched *zero or more* times. ie. n* will match n, nn, nnnn, nnnnnnn but not na or any other character.

^ (caret)
: means "the beginning of the line". So "^a" means find a line starting with an "a".

$ (dollar sign)
: means "the end of the line". So "a$" means find a line ending with an "a".

For example, this command searches the file myfile for lines starting with an "s" and ending with an "n", and prints them to the standard output (screen):

```
cat myfile | grep '^s.*n$'
```

[ ] (square brackets)
: specifies a range. If you did m[a,o,u]m it can become: mam, mum, mom if you did: m[a−d]m it can become anything that starts and ends with m and has any character a to d inbetween. For example, these would work: mam, mbm, mcm, mdm. This kind of wildcard specifies an "or" relationship (you only need one to match).

|
: This wildcard makes a logical OR relationship between wildcards. This way you can search for something or something else (possibly using two different regular expressions). You may need to add a '\' (backslash) before this command to work, because the shell may attempt to interpret this as a pipe.

[^]
: This is the equivalent of [!] in standard wildcards. This performs a logical "not". This will match anything that is not listed within those square brackets. For example, *rm myfile[^9]* will remove all myfiles* (ie. myfiles1, myfiles2 etc) but won't remove a file with the number 9 anywhere within it's name.

20.4.3. Useful categories of characters (as defined by the POSIX standard)

This information has been taken from the grep info page with a tiny amount of editing, see [10] in the *Bibliography* for further information.

- [:upper:] uppercase letters
- [:lower:] lowercase letters
- [:alpha:] alphabetic (letters) meaning upper+lower (both uppercase and lowercase letters)
- [:digit:] numbers in decimal, 0 to 9
- [:alnum:] alphanumeric meaning alpha+digits (any uppercase or lowercase letters or any decimal digits)
- [:space:] whitespace meaning spaces, tabs, newlines and similar
- [:graph:] graphically printable characters excluding space
- [:print:] printable characters including space
- [:punct:] punctuation characters meaning graphical characters minus alpha and digits
- [:cntrl:] control characters meaning non−printable characters
- [:xdigit:] characters that are hexadecimal digits.

These are used with

The above commands will work with most tools which work with text (for example: *tr*).

For example (advanced example), this command scans the output of the dir command, and prints lines containing a capital letter followed by a digit:

```
ls -l | grep '[[:upper:]][[:digit:]]'
```

The command greps for [upper_case_letter][any_digit], meaning any uppercase letter followed by any digit. If you remove the [ ] (square brackets) in the middle it would look for an uppercase letter or a digit, because it would become [upper_case_letter any_digit]

Appendix A. Appendix

A.1. Finding Packages/Tools

A.1.1. Finding more useful tools

If you are looking to find more tools, the GNU project (GNU's Not Unix) maintains a directory, a website listing categorized links to various free−software tools (which they consider useful) called the GNU Directory.

Also try sites such as Sweet Code which offer mailing lists of useful tools which they find.

You may also try looking at the most highly rated, most active or most downloaded programs at SourceForge and FreshMeat.

A.1.2. Finding a particular tool(s)

Many of the tools listed in this guide are part of a package of tools, such as *diffutils* which contains the various tools used to find differences between files, such as *diff, sdiff, diff3, cmp*. Most small tools are bundled together in this fashion. Most major distribution's will offer a search function to help you search the packages by file, you can of course do this via the command−line interface or a GUI.

If you need to search the distribution's available packages via the command−line, the method will vary depending on the distribution you are using, see the subsections below or consult your distribution's documentation (or of course the internet):

A.1.2.1. Mandriva (urpm* commands, rpm based)

To find where a particular file came from use *urpmf*.

Command syntax:

```
urpmf file_name
```

The results are often overwhelming as this particular command will take a string and list every file of every package in it's database that contains the particular keyword (ie. both uninstalled and installed packages). To refine the results you may want to add a pipe to it and send it through *grep −w file_name* (the *−w* option will only show you only exact (whole word) matches). How you would do this is shown below:

```
urpmf file_name | grep −w file_name
```

For more information on the urpm* commands, please refer to the tip towards the end of this section: Section 20.1.

A.1.2.2. Red Hat (rpm)

To find which package a particular file came from use *rpm* with the *−qf* option.

Command syntax:

```
rpm -qf /path/to/the/file
```

This will find which package the file came from. You need to use *rpm −qf* not with a keyword but with the location of the actual file. To find more information on the particular package listed use *rpm* with the *−qi* option.

Command syntax:

```
rpm -qi package_name
```

Note that the package name is the name of the package without the *.arch.rpm* (often *.i386.rpm*) extension on the end.

For more information on the usage of rpm, please refer to this section Section 20.1.

A.1.2.3. Debian (deb)

To find where a particular file came from use dpkg with the *−S* option.

There are two ways to do this:

```
dpkg -S file_name
```

or:

```
dpkg -S /path/to/file
```

You may also like to try (if it's installed, it's generally a lot faster than the *dpkg* search):

```
dlocate -S file_name
```

For more information on dpkg and dlocate please refer to the relevant manual pages and online sources of information.

A.1.3. Finding package(s)

Packages can be found via the internet utilizing sites such as:

- RPMFind for RPM based packages.
- Debian Package Listfor deb packages.
- RPMSeek, this site intends to index Debian packages as well as RPM.
- TuxFinder where you can search for deb, rpm, tgz, iso and even documentation.

Also try the author's homepage and large sites such as FreshMeat and SourceForge.

A.2. Further Reading

A.2.1. General Further Reading

This guide is simply a short summary of some of the available tools of a GNU/Linux based distribution. If you find a particular command interesting and useful, you can look up the on−line manual, or/and info page to learn more about how to use this command or check the HOWTO's online at Linux Documentation Project.

The manual/info pages will always be an up−to−date source of information on how to use the command. Also have a look at the documentation installed on your distribution, its normally located in /usr/share/doc.

Check the references section of this document, *Bibliography*, for some links to useful resources which were used in the creation of this document.

Of course if you are having trouble with a particular command try using a search engine such as Google or AllTheWeb, or search the usenet groups Google Groups. If you still can't find a solution, look for a mailing list which is related to the topic you are having trouble with, or try a forum which is related to the topic.

Readers who would like another reference to commands may want to have at:

- Commands from "Linux in a Nutshell 3rd Edition" as published by Orielly −−− this document was not used in the creation of this guide, however it is a comprehensive guide to GNU/Linux Commands (it's an indexed listing). It lists and explains 379 commands taken from *Linux in a Nutshell 3rd Edition.*
- The Linux Newbie Admin guide list of commands −−− another list of commands from an excellent system administration guide for GNU/Linux.
- Comptechdoc's Linux Command Quickreference Guide −−− a good list of commands but only one line explanations of what they actually do...
- SS64.com list of bash commands −−− this page lists commands and links to their man pages online.

If you wish to learn more about GNU/Linux on a variety of subjects also see the various online (free) tutorials published by IBM Developerworks.

If you are looking for a general reference to everything GNU/Linux try the Rute User's Tutorial and Exposition. Or take a look at your distributions documentation, Debian maintains comprehensive documentation, debian documentation site.

A.2.2. Specific Further reading

The most obvious place to look for documentation is to find the homepage of the program. Although sometimes there are other sources of information such as the Linux Documentation Project or various online HOWTO's or similar guides. They are usually easily found using search engines. Try large sites such as (ibiblio) the publics library and digital archive or TuxFinder which can search for documentation.

Below is a very short list of some further reading for a few of the more complex tools:

- OpenSSH OpenSSH manual page
- vim The Vim HOWTO
- emacs The Emacs HOWTO
- RPM RPM HOWTO
- Samba Samba documentation site
- ImageMagick ImageMagick command−line tools
- BASH BASH reference manual
- Bash scripting Advanced bash scripting guide
- rsync rsync homepage

A.2.2.1. The UNIX tools philosophy further reading

- An article within the coreutils documentation (installed on nearly every GNU/Linux distro) provides further explanation of the UNIX tools philosophy. To access the article simply type:
  ```
  info coreutils
  ```
  Then type / (slash; runs a search) then the string *"toolbox"* (toolbox is the string to be searched for) then hit enter (follow hyperlink) and then go down to the "Toolbox introduction" section and hit enter. This will give you access to the article.
- Other articles online include an: Orielly article on the UNIX tools philosophy.
- A listing of important qualities of the philosophy.
- Linux Exposed The Unix Philosophy Explained
- Or an entire book which is considered the authoritative guide toward understanding the philosophy behind how the UNIX system was built. The book is called "The Unix Philosophy" ISBN: 1555581234.

A.2.3. Online Manual And Info Pages

While manual pages and info pages are usually installed with the program itself they are also available online if you need them, the listed links are usually listed by category or by the man page sections.

A.2.3.1. Online Manual Page Websites:

- Manual Page Resource Links (from the Linux Documentation Project)
- A RedHat Based Searchable Index
- Another Searchable Index
- Another Manual Page Site (searchable)

A.2.3.2. Downloadable Manual Pages:

- Downloadable Man Pages hosted by Ibiblio

A.2.3.3. Online Info Page Website:

- GNU Manual's

A.3. GNU Free Documentation License

A.3.1. PREAMBLE

A.3.2. APPLICABILITY AND DEFINITIONS

A.3.3. VERBATIM COPYING

A.3.4. COPYING IN QUANTITY

A.3.5. MODIFICATIONS

A.3.6. COMBINING DOCUMENTS

A.3.7. COLLECTIONS OF DOCUMENTS

A.3.8. AGGREGATION WITH INDEPENDENT WORKS

Bibliography

(1) Tony Steidler−Dennison, *Lockergnome Penguin Shell Series*, Lockergnome.

Responsible for many of the commands listed in this document. In particular Lockergnome inspired much of the wildcards section: Section 20.4

(2) Brandon Rhodes, *Linux Network Commands Page*.

Responsible for parts of the network commands section: Chapter 13

(3) Michael Stutz, *Linux Cookbook Homepage*, No Starch Press.

Many of these commands have come from the Linux Cookbook (version 1.2). I highly recommend this book to any novice or intermediate GNU/Linux user, have a look at it online, and then of course buy it :).

(4) Michael Jordan, *Linux Online Classroom*, Linux Online.

Some very small sections of this document were taken from the Beginner's course on the Linux Online Website.

(5) man and info pages.

The man and info pages of various tools listed in this document have been used as a resource to assist in the creation of this document. They are a useful resource of up−to−date information on a program and should be consulted when you require information about a particular tool.

(6) Focus On Unix −− Unix.about.com.

Some of the tutorials under the power commands section of the unix.about.com site were used in the construction of this guide. In particular parts of the *xargs* command: Chapter 8 and parts of the cut command: Section 11.4 were used from their tutorials.

(7) MandrakeSoft Command Line Manual, MandrakeSoft.

The Command Line Manual developed for Mandake Linux 9.0 was used in the creation of this document. A small section (in regard to command−line completion) was used from this document. If you are running mandrake you will most likely find this guide here.

(8) MandrakeSoft Starter Guide, MandrakeSoft.

The MandrakeSoft Starter Guide, a guide developed for Mandake Linux 9.0 was used in the creation of this document. A small section (in regard to how to recover from a system freeze) was used from this document. If you are running a mandrake system you will most likely find the document here.

(9) Hrvoje Niksic, *Wget Manual page*, Free Software Foundation.

A section of the *wget* manual page was used in this guide, from this page, Wget Manual page. In particular relating to downloading multiple files while using the http protocol.

(10) Grep, Free Software Foundation.

Both wildcards subsections are based off the grep manual and info pages. The Useful Categories of Characters (as defined from the POSIX standard) was taken from the grep info page.

(11) Marc Ewing, Jeff Johnson, and Erik Troan, *RPM Manual Page*, Red Hat.

A small section of the RPM manual page was used in the creation of the RPM verifying subsection, without any kind of editing.

(12) Markku Rossi, *Enscript Manual Page*, Free Software Foundation.

The examples for *enscript* are based off those shown in the enscript manual page.

(13) Paul Vixie, *Cron Manual Page*, 4th Berkeley Distribution.

The information from the crontab section (below and including the table) was taken (unedited, but with small additions) from the crontab manual pages. Type *man 1 crontab* and *man 5 crontab* to access the 2 different manual pages.

(14) IBM Developerworks.

Some parts of the IBM Developerworks tutorials have been used in the creation of this document. IBM Developerworks frequently publishes new tutorials on a variety of subjects, visit the IBM Developerworks Linux site (see link above) for more information on their GNU/Linux tutorials.

(15) Suso Banderas, *Num−utils homepage*.

The num−utils manual pages were used in the creation of the maths section. In particular all the description of the num−utils tools are based off the manual pages on the num−utils homepage.

(16) Carla Schroder, *Archive of the LinuxChix posting*.

This particular LinuxChix posting was made through a mailing list discussion about *cron* under the TechTalk mailing list. The posters homepage is http://www.tuxcomputing.com.

(17) Joe Barr, *CLI for noobies: import, display, mogrify*.

This particular article by Joe Barr was used in the description of the *mogrify* tool in particular the example on creating thumbnails.

(18) Kyle Rankin, *Please, For the Love of All That's Recoverable, Shred Your Hard Drive!*.

This particular article by Kyle Rankin was used (only a paragraph) for information on the shred command.