Leaflet U5

Thirty Useful Unix Commands

Last revised April 1997

This leaflet contains basic information on thirty of the most frequently used Unix Commands. It is intended for Unix beginners who need a guide to the names and details of commands that are likely to be of use to them.

Every effort has been made to make this leaflet as generally valid as possible, but there are many different versions of Unix available within the University, so if you should find a command option behaving differently on your local machine you should consult the on-line manual page for that command. Some commands have numerous options and there is not enough space to detail them all here, so for fuller information on these commands use the relevant on-line manual page.

The names of commands are printed in bold, and the names of objects operated on by these commands (e.g. files, directories) are printed in italics.

Index of Commands

cat - display or concatenate files

`cat` takes a copy of a file and sends it to the standard output (i.e. to be displayed on your terminal, unless redirected elsewhere), so it is generally used either to read files, or to string together copies of several files, writing the output to a new file.

`cat` `ex`
: displays the contents of the file `ex`.

`cat` `ex1 ex2` **`>`** `newex`
: creates a new file `newex` containing copies of `ex1` and `ex2`, with the contents of `ex2` following the contents of `ex1`.

cd - change directory

`cd` is used to change from one directory to another.

`cd` `dir1`
: changes directory so that `dir1` is your new current directory. `dir1` may be either the full pathname of the directory, or its pathname relative to the current directory.

`cd`
: changes directory to your home directory.

`cd ..`
: moves to the parent directory of your current directory.

chmod - change the permissions on a file or directory

`chmod` alters the permissions on files and directories using either symbolic or octal numeric codes. The symbolic codes are given here:-

```
u  user      +  to add a permission                  r  read
g  group     -  to remove a permission               w  write
o  other     =  to assign a permission explicitly    x  execute (for files),
                                                        access (for directori
```

The following examples illustrate how these codes are used.

`chmod u=rw` `file1`
: sets the permissions on the file `file1` to give the user read and write permission on `file1`. No other permissions are altered.

`chmod u+x,g+w,o-r` `file1`
: alters the permissions on the file `file1` to give the user execute permission on `file1`, to give members of the user's group write permission on the file, and prevent any users not in this group from reading it.

`chmod u+w,go-x` `dir1`
: gives the user write permission in the directory `dir1`, and prevents all other users having access to that directory (by using **`cd`**. They can still list its contents using **`ls`**.)

compress - compress a file

`compress` reduces the size of named files, replacing them with files of the same name extended by **`.Z`** . The amount of space saved by compression varies. If no saving of space would occur, then the file will not be altered.

`compress` `file1`
: results in a compressed file called `file1.Z`, and deletes `file1`.

`compress` `-v file2`
: compresses `file2` and gives information, in the format shown below, on the percentage of the file's size that has been saved by compression:-

```
file2 : Compression 50.26 -- replaced with file2.Z
```

To restore files to their original state use the command **`uncompress`**. If you have a compressed file `file2.Z`, then

`uncompress` `file2`
: will replace `file2.Z` with the uncompressed file `file2`.

cp - copy a file

The command **`cp`** is used to make copies of files and directories.

`cp` `file1 file2`
: copies the contents of the file `file1` into a new file called `file2`. **`cp`** cannot copy a file onto itself.

`cp` `file3 file4 dir1`
: creates copies of `file3` and `file4` (with the same names), within the directory `dir1`. `dir1` must already exist for the copying to succeed.

`cp -r` `dir2 dir3`
: recursively copies the directory `dir2`, together with its contents and subdirectories, to the directory `dir3`. If `dir3` does not already exist, it is created by **`cp`**, and the contents and subdirectories of `dir2` are recreated within it. If `dir3` does exist, a subdirectory called `dir2` is created within it, containing a copy of all the contents of the original `dir2`.

date - display the current date and time

`date` returns information on the current date and time in the format shown below:-

```
Tue Mar 25 15:21:16 GMT 1997
```

It is possible to alter the format of the output from date. For example, using the command line

`date 'The date is d/m/y, and the time is H:M:S.'`

at exactly 3.10pm on 14th December 1997, would produce the output

```
The date is 14/12/97, and the time is 15:10:00.
```

diff - display differences between text files

`diff` `file1 file2` reports line-by-line differences between the text files `file1` and `file2`. The default output will contain lines such as **`n1 a n2,n3`** and **`n4,n5 c n6,n7`** , (where n1 a n2,n3 means that `file2` has the extra lines n2 to n3 following the line that has the number n1 in file1, and **`n4,n5 c n6,n7`** means that lines n4 to n5 in **`file1`** differ from lines n6 to n7 in `file2`). After each such line, **`diff`** prints the relevant lines from the text files, with **`<`** in front of each line from `file1` and **`>`** in front of each line from `file2`.

There are several options to **`diff`**, including **`diff -i`**, which ignores the case of letters when comparing lines, and **`diff -b`**, which ignores all trailing blanks.

`diff -c``n`
: produces a listing of differences within `n` lines of context, where the default is three lines. The form of the output is different from that given by **`diff`**, with **`+`** indicating lines which have been added, **`-`** indicating lines which have been removed, and **`!`** indicating lines which have been changed.

`diff` `dir1 dir2`
: will sort the contents of directories `dir1` and `dir2` by name, and then run **`diff`** on the text files which differ.

echo - echo arguments to the standard output

`echo` echoes given arguments to the standard output, and is generally used in shell programs.

`echo` `argument1`
: writes `argument1` to the standard output.

file - determine the type of a file

`file` tests named files to determine the categories their contents belong to.

`file` `file1`
: can tell if `file1` is, for example, a source program, an executable program or shell script, an empty file, a directory, or a library, but (a warning!) it does sometimes make mistakes.

find - find files of a specified name or type

`find` searches for files in a named directory and all its subdirectories.

`find . -name` `'*.f'` **`-print`**
: searches the current directory and all its subdirectories for files ending in `.f`, and writes their names to the standard output. In some versions of Unix the names of the files will only be written out if the **`-print`** option is used.

find `/local` **-name** `core` **-user** `user1` **-print**
searches the directory `/local` and its subdirectories for files called `core` belonging to the user `user1` and writes their full file names to the standard output.

finger - display information about a user

finger can be used to obtain information on users on your own and other machines.

finger
on its own will give information on all users currently logged onto your machine: their user names, real names, the terminal they are using and its idle time, the time they logged on, and the name of the machine from which they logged on:-

```
Login    Name         TTY   Idle   When        Where
user1     Julian Brown   p1      2      Mon 09:04    sole.cam.ac.uk
user2     Joyce Smith    p3     17      Tue 08:24    carp.cam.ac.uk
```

finger `user1`
gives further information about `user1`. The kind of information given varies from machine to machine, but may include the name of `user1`'s home directory and login shell, the last time `user1` received mail, and when mail was last read. It may also display the contents of `user1`'s `.plan` and `.project` files, if these exist. **finger -l** `user1` may give more information than **finger** `user1`. It is also possible to obtain this kind of information about users on other hosts, if these machines permit. So

finger `user1@sole.cam.ac.uk`
could give similar information about Julian Brown's account on the host `sole.cam.ac.uk`.

ftp - file transfer program

ftp is an interactive file transfer program. While logged on to one machine (described as the local machine), **ftp** is used to logon to another machine (described as the remote machine) that files are to be transferred to or from. As well as file transfers, it allows the inspection of directory contents on the remote machine. There are numerous options and commands associated with **ftp**, and **man** `ftp` will give details of those.

A simple example **ftp** session, in which the remote machine is the Central Unix Service (CUS), is shown below:-

```
ftp cus.cam.ac.uk
```

If the connection to CUS is made, it will respond with the prompt:-

```
Name  (cus.cam.ac.uk:user1)  :
```

(supposing `user1` is your username on your local machine). If you have the same username on CUS, then just press `Return`; if it is different, enter your username on CUS before pressing `Return`. You will then be prompted for your CUS password, which will not be echoed.

After logging in using **ftp** you will be in your home directory on CUS Some Unix commands, such as **cd**, **mkdir**, and **ls**, will be available. Other useful commands are:

help
lists the commands available to you while using **ftp**.

get `remote1 local1`
creates a copy on your local machine of the file `remote1` from CUS. On your local machine this new file will be called `local1`. If no name is specified for the file on the local machine, it will be given the same name as the file on CUS.

send `local2 remote2`
copies the file `local2` to the file `remote2` on CUS, i.e. it is the reverse of **get**.

quit
finishes the **ftp** session. **bye** and **close** can also be used to do this.

Some machines offer a service called **"anonymous ftp"**, usually to allow general access to certain archives. To use such a service, enter `anonymous` instead of your username when you **ftp** to the machine. It is fairly standard practice for the remote machine to ask you to give your email address as a password. Once you have logged on you will have read access in a limited set of directories, usually within the `/pub` directory tree. It is good etiquette to follow the guidelines laid down by the administrators of the remote machine, as they are being generous in allowing such access. See leaflet G72: File transfer methods and FTP for more detailed examples of using **ftp**.

grep - searches files for a specified string or expression

grep searches for lines containing a specified pattern and, by default, writes them to the standard output.

grep `motif1 file1`
searches the file `file1` for lines containing the pattern `motif1`. If no file name is given, **grep** acts on the standard input. **grep** can also be used to search a string of files, so

grep `motif1 file1 file2 ... filen`
will search the files `file1`, `file2`, `. . .` , `filen`, for the pattern `motif1`.

grep `-c motif1 file1`
will give the number of lines containing `motif1` instead of the lines themselves.

grep `-v motif1 file1`
will write out the lines of `file1` that do NOT contain `motif1`.

kill - kill a process

To kill a process using **kill** requires the process id (PID). This can be found by using **ps**. Suppose the PID is 3429, then

kill `3429`
    should kill the process.

lpr - print out a file

`lpr` is used to send the contents of a file to a printer. If the printer is a laserwriter, and the file contains PostScript, then the PostScript will be interpreted and the results of that printed out.

`lpr` `-Pprinter1 file1`
    will send the file `file1` to be printed out on the printer `printer1`. To see the status of the job on the printer queue use

`lpq` `-Pprinter1`
    for a list of the jobs queued for printing on `printer1`. (This may not work for remote printers.)

ls - list names of files in a directory

`ls` lists the contents of a directory, and can be used to obtain information on the files and directories within it.

`ls` `dir1`
    lists the names of the files and directories in the directory `dir1`, (excluding files whose names begin with .). If no directory is named, **`ls`** lists the contents of the current directory.

`ls` `-a dir1`
    will list the contents of `dir1`, (including files whose names begin with .).

`ls` `-l file1`
    gives details of the access permissions for the file `file1`, its size in kbytes, and the time it was last altered.

`ls` `-l dir1`
    gives such information on the contents of the directory `dir1`. To obtain the information on `dir1` itself, rather than its contents, use

`ls` `-ld dir1`

man - display an on-line manual page

`man` displays on-line reference manual pages.

`man` `command1`
    will display the manual page for `command1`, e.g **`man`** `cp`, **`man`** `man`.

`man -k` `keyword`
    lists the manual page subjects that have keyword in their headings. This is useful if you do not yet know the name of a command you are seeking information about.

`man -M``path command1`
    is used to change the set of directories that man searches for manual pages on `command1`

mkdir - make a directory

`mkdir` is used to create new directories. In order to do this you must have write permission in the parent directory of the new directory.

`mkdir` `newdir`
    will make a new directory called `newdir`.

`mkdir -p` can be used to create a new directory, together with any parent directories required.

`mkdir -p` `dir1/dir2/newdir`
    will create newdir and its parent directories `dir1` and `dir2`, if these do not already exist.

more - scan through a text file page by page

`more` displays the contents of a file on a terminal one screenful at a time.

`more` `file1`
    starts by displaying the beginning of `file1`. It will scroll up one line every time the return key is pressed, and one screenful every time the space bar is pressed. Type **?** for details of the commands available within **`more`**. Type **`q`** if you wish to quit more before the end of `file1` is reached.

`more -``n file1`
    will cause `n` lines of `file1` to be displayed in each screenful instead of the default (which is two lines less than the number of lines that will fit into the terminal’s screen).

mv - move or rename files or directories

`mv` is used to change the name of files or directories, or to move them into other directories. **`mv`** cannot move directories from one file-system to another, so, if it is necessary to do that, use **`cp`** instead.

`mv` `file1 file2`
    changes the name of a file from `file1` to `file2` unless `dir2` already exists, in which case `dir1` will be moved into `dir2`.

`mv` `dir1 dir2`
    changes the name of a directory from `dir1` to `dir2`.

`mv` `file1 file2 dir3`
    moves the files `file1` and `file2` into the directory `dir3`.

nice - change the priority at which a job is being run

`nice` causes a command to be run at a lower than usual priority. **`nice`** can be particularly useful when running a long program that could cause annoyance if it slowed down the execution of other users' commands. An example of the use of **`nice`** is

`nice compress` `file1`
: which will execute the compression of `file1` at a lower priority.

passwd - change your password

Use **`passwd`** when you wish to change your password. You will be prompted once for your current password, and twice for your new password. Neither password will be displayed on the screen.

ps - list processes

`ps` displays information on processes currently running on your machine. This information includes the process id, the controlling terminal (if there is one), the cpu time used so far, and the name of the command being run.

`ps`
: gives brief details of your own processes.

`ps -a`
: gives information on other users' processes as well as your own.

`ps` is a command whose options vary considerably in different versions of Unix (such as BSD and SystemV). Use **`man`** `ps` for details of all the options available on the machine you are using.

pwd - display the name of your current directory

The command **`pwd`** gives the full pathname of your current directory.

quota - disk quota and usage

`quota` gives information on a user's disk space quota and usage.

`quota`
: will only give details of where you have exceeded your disc quota on local disks, whereas

`quota -v`
: will display your quota and usage, whether the quota has been exceeded or not, and includes information on disks mounted from other machines, as well as the local disks.

rm - remove files or directories

`rm` is used to remove files. In order to remove a file you must have write permission in its directory, but it is not necessary to have read or write permission on the file itself.

`rm` `file1`
: will delete the file `file1`. If you use

`rm -i` `file1`
: instead, you will be asked if you wish to delete `file1`, and the file will not be deleted unless you answer **`y`**. This is a useful safety check when deleting lots of files.

`rm -r` `dir1`
: recursively deletes the contents of `dir1`, its subdirectories, and `dir1` itself, and should be used with suitable caution.

rmdir - remove a directory

`rmdir` removes named empty directories. If you need to delete a non-empty directory **`rm -r`** can be used instead.

`rmdir` `exdir`
: will remove the empty directory `exdir`.

sort - sort and collate lines

The command **`sort`** sorts and collates lines in files, sending the results to the standard output. If no file names are given, **`sort`** acts on the standard input. By default, **`sort`** sorts lines using a character by character comparison, working from left to right, and using the order of the ASCII character set.

`sort -d`
: uses "dictionary order", in which only letters, digits, and white-space characters are considered in the comparisons.

`sort -r`
: reverses the order of the collating sequence.

`sort -n`
: sorts lines according to the arithmetic value of leading numeric strings. Leading blanks are ignored when this option is used, (except in some System V versions of **`sort`**, which treat leading blanks as significant. To be certain of ignoring leading blanks use **`sort -bn`** instead.).

talk - "talk" to another user

`talk` copies lines from your terminal to another user's terminal. Use

`talk` `user1`
: if you wish to talk with user `user1`, who is logged onto the same machine. If the user is on another

machine, called, say, `sole`, then use

```
talk user1@sole
```

Initially, **`talk`** will send a message to the other user, requesting a **`talk`** connection. A **`talk`** session will only be established if that user responds by using **`talk`** to ask to talk to you. So if your username is `user2` and your host machine is `carp`, then `user1` must reply

```
talk user2@carp
```

Then two windows are formed on both terminals for the users to type their messages to each other. To exit from talk use the interrupt character.

wc - display a count of lines, words and characters

`wc` counts the number of lines, words, and characters in files. If no filename is given, **`wc`** will count the standard input instead.

```
wc file1
```

will produce output of the form

```
 3     12     184     file1
```

showing that `file1` contains 3 lines, 12 words, and 184 characters. There are options to **`wc`** that restrict the count to lines, or words, or characters. **`wc -l`** will just count lines, **`wc -w`** will only count words, and **`wc -c`** only counts characters.

Documentation | Computing Service home | University home