

UNIX Operating System

The UNIX operating system

The UNIX operating system is made up of three parts; the kernel, the shell and the programs.

The kernel

The kernel of UNIX is the nub of the operating system: it allocates time and memory to programs and handles the filestore and communications in response to system calls.

As an illustration of the way that the shell and the kernel work together, suppose a user types `rm myfile` (which has the effect of removing the file "myfile"). The shell searches the filestore for the file containing the program `rm`, and then requests the kernel, through system calls, to execute the program `rm` on `myfile`. When the process `rm myfile` has finished running, the shell then returns the UNIX prompt `%` to the user, indicating that it is waiting for further commands.

The shell

The shell acts as an interface between the user and the kernel. When a user logs in, the login program checks the username and password, and then starts another program called the shell. The shell is a command line interpreter (CLI). It interprets the commands the user types in and arranges for them to be carried out. The commands are themselves programs: when they terminate, the shell gives the user another prompt.

The adept user can customize his/her own shell, and users can use different shells on the same machine.

Tcsh Shell

The **tcsh** shell has certain features to help the user inputting commands.

Filename Completion - By typing part of the name of a command, filename or directory and pressing the [Tab] key, the tcsh shell will complete the rest of the name automatically. If the shell finds more than one name beginning with those letters you have typed, it will beep, prompting you to type a few more letters before pressing the tab key again.

History - The shell keeps a list of the commands you have typed in. If you need to repeat a command, use the cursor keys to scroll up and down the list.

UNIX Tutorial One

- 1) Log into the computer by using the **root** ID and type in the appropriate password.
- 2) **Optional:** You can start Xwindows by typing:
xinit or **startx** and then start the *Konsole* application that gives you the command prompt.
- 3) Determining the shell type by running the following command:
echo \$shell

- 4) The running shell is _____
- 5) To determine the environment variables, type in the following command:

setenv

- 6) Use the displayed information to determine the values of these variables:

USER _____

HOME _____

HOST _____

SHELL _____

OSTYPE _____

PWD _____

- 7) Determine which shells are installed by using the following command:

cat /etc/shells

- 8) List the shells installed: _____

- 9) In order to determine who is logged in, type:

echo \$user or **whoami**.

- 10) To shutdown the system, type:

shutdown -h now

or to reboot the system, type:

shutdown -r now

Files and processes

Everything in UNIX is either a file or a process.

A process is an executing program identified by a unique PID (process identifier).

A file is a collection of data. They are created by users using text editors, running compilers etc.

Examples of files:

? a document (report, essay etc.)

? the text of a program written in some high-level programming language

- ? instructions comprehensible directly to the machine and incomprehensible to a casual user, for example, a collection of binary digits (an executable or binary file);
- ? a directory, containing information about its contents, which may be a mixture of other directories (subdirectories) and ordinary files.

The Directory Structure

All the files are grouped together in the directory structure. The file-system is arranged in a hierarchical structure, like an inverted tree. The top of the hierarchy is traditionally called "root".

UNIX Tutorial Two

Listing files and directories

ls (list schema)

- 1) When you first login, your current working directory is your home directory. Your home directory has the same name as your user-name. In order to determine your home directory, type the following:

echo \$home

Home Directory _____

- 2) To ensure that you are in your home directory, type

pwd

This command displays the current directory that you are in.

- 3) To find out what is in your home directory, type

ls

The `ls` command lists the contents of your current working directory.

There may be no files visible in your home directory, in which case, the UNIX prompt will be returned. Alternatively, there may already be some files inserted by the System Manager.

`ls` does not, in fact, cause all the files in your home directory to be listed, but only those ones whose name does not begin with a dot (`.`) Files beginning with a dot (`.`) are known as hidden files and usually contain important program configuration information. They are hidden because you should not change them unless you are very familiar with UNIX and Xwindows.

To list all files in your home directory including those ones, whose names begin with a dot, type

ls -a

`ls` is an example of a command which can take options: `-a` is an example of an option. The options change the behavior of the command (cf. the difference between `ls` and `ls -a`). There are online manual pages that tell you which options a particular command can take, and how each option modifies the behavior of the command. (See later in this tutorial)

Making Directories

mkdir (make directory)

We will now make a subdirectory in your home directory to hold the files you will be creating and using in the course of this tutorial. To make a subdirectory called `unixstuff` in your current working directory type

```
mkdir unixstuff
```

To see the directory you have just created, type

```
ls
```

Changing to a different directory

cd (change directory)

The command `cd directory` means "change the current working directory to 'directory'". The current working directory may be thought of as the directory you are in, i.e. your current position in the file-system tree.

To change to the directory you have just made, type

```
cd unixstuff
```

Type `ls` to see the contents (which should be empty)

Exercise

Make another directory inside the `unixstuff` directory called `projects`

The directories `.` and `..`

Still in the `unixstuff` directory, type

```
% ls -a
```

As you can see, in the `unixstuff` directory (and in all other directories), there are two special directories called `"."` and `".."`

In UNIX, `"."` means the current directory, so typing

```
cd .
```

means stay where you are (the `unixstuff` directory).

This may not seem very useful at first, but using `."` as the name of the current directory will save a lot of typing, as we shall see later in the tutorial.

`.."` means the parent of the current directory, so typing

```
cd ..
```

will take you one directory up the hierarchy (back to your home directory). Try it now.

Note: typing `cd` with no argument always returns you to your home directory. This is very useful if you are lost in the file system.

Pathnames

pwd (print working directory)

Pathnames enable you to work out where you are in relation to the whole file-system. For example, to find out the absolute pathname of your home-directory, `cd` to your home-directory and type

```
pwd
```

Exercise

Use the commands `ls`, `pwd` and `cd` to explore the file system.

(Remember, if you get lost, type `cd` by itself to return to your home-directory)

More about home directories and pathnames

Go to your home directory, and type

```
ls unixstuff
```

Now type

```
ls projects
```

You will get a message like this -

```
projects: No such file or directory
```

The reason is, `projects` is not in your current working directory. To use a command on a file (or directory) not in the current working directory (the directory you are currently in), you must specify its full pathname. To list the contents of your `projects` directory, you must type

```
ls unixstuff/projects
```

Home directories can also be referred to by the tilde `~` character. It can be used to specify paths starting at your home directory. So typing

```
ls ~/unixstuff
```

will list the contents of your `unixstuff` directory, no matter where you currently are in the file system.

What do you think

```
ls ~
```

would list?

What do you think

```
ls ~/.
```

would list?

Summary

| | |
|---------------------------|-------------------------------------------|
| <code>ls</code> | list files and directories |
| <code>ls -a</code> | list all files and directories |
| <code>mkdir</code> | make a directory |
| <code>cd directory</code> | change to named directory |
| <code>cd</code> | change to home-directory |
| <code>cd ..</code> | change to parent directory |
| <code>pwd</code> | display the path of the current directory |

UNIX Tutorial Three

Creating Files

In order to create a file, do the following:

- 1) Go to the `unixstuff` directory in your home directory.
- 2) Type `pwd` to ensure that you are in that directory.
- 3) Type `cat >test`. What this does is that it start recording everything you type into the test file.
- 4) Type the following lines beginning with the numbers and the colon character:


```
1:We are learning UNIX.
2:University of Phoenix
3:Vancouver
4:BC
5:Canada
6:North America
7:America
8:Earth
9:Planet
10:Universe
11:Solar System
12:Galaxy
```
- 5) Then press **Ctrl-D** to signal EOF.
- 6) The `test` file will contain the text that you type.
- 7) To see the contents of the `test` file, do the following:

```
cat test
```

Copying Files

cp (copy)

`cp file1 file2` is the command which makes a copy of `file1` in the current working directory and calls it `file2`

What we are going to do now, is to take a file stored in an open access area of the file system, and use the `cp` command to copy it to your `unixstuff` directory.

First, change directory to your `unixstuff` directory.

Then at the UNIX prompt, type,

```
cp test projects/test
```

The above command means copy the file `test` to the `projects` directory, keeping the name the same.

Moving files

mv (move)

`mv file1 file2` moves (or renames) `file1` to `file2`

To move a file from one place to another, use the `mv` command. This has the effect of moving rather than copying the file, so you end up with only one file rather than two.

It can also be used to rename a file, by "moving" the file to the same directory, but giving it a different name.

Change directories to your `unixstuff` directory.

We are now going to rename (move) the file `test` to `myfile` by typing

```
mv test myfile
```

Type `ls` to see if it has worked.

Removing files and directories

rm (remove), rmdir (remove directory)

To delete (remove) a file, use the `rm` command. As an example, we are going to remove the copy of the `science.txt` file in your `project` directory.

1) First change into your `project` directory, then type

```
rm test
```

to delete the test file

- 2) Then change directory to the *unixstuff* directory.
- 3) Use the **rmdir** command to remove a directory (make sure it is empty first). Try to remove the *project* directory.

Displaying the contents of a file on the screen

clear (clear screen)

Before you start the next section, you may like to clear the terminal window of the previous commands so the output of the following commands can be clearly understood.

At the prompt, type

```
clear
```

This will clear all text and leave you with the % prompt at the top of the window.

cat (concatenate)

The command `cat` can be used to display the contents of a file on the screen. Type:

```
cat myfile
```

more

The command `more` writes the contents of a file onto the screen a page at a time. Type

```
more myfile
```

Press the space-bar if you want to see another page, type **q** if you want to quit reading. As you can see, `more` is used in preference to `cat` for long files .

head

The `head` command writes the first ten lines of a file to the screen.

First **clear** the screen then type

```
head myfile
```

Then type

```
head -5 myfile
```

What difference did the `-5` do to the `head` command?

tail

The `tail` command writes the last ten lines of a file to the screen.

Clear the screen and type

```
tail myfile
```

How can you view the last 5 lines of the file?

Searching the contents of a file

Searching using the *grep* command

grep is one of many standard UNIX utilities. It search files for specified words or patterns. First clear the screen, then type

```
grep Canada myfile
```

As you can see, *grep* has printed out each line containing the word Canada.

Or has it???

Try typing

```
grep canada myfile
```

The *grep* command is "case sensitive"; it distinguishes between Canada and canada.

To ignore upper/lower case distinctions, use the *-i* option, i.e. type

```
grep -i canada myfile
```

To search for a phrase or pattern, you must enclose it in single quotes. For example to search for spinning top, type

```
grep -i 'North America' myfile
```

Some of the other options of *grep* are:

```
-v      display those lines that do NOT match
-n      precede each matching line with the line number
-c      print only the total count of matched lines
```

Try some of them and see the different results. Don't forget, you can use more than one option at a time, for example *grep -ivc*

wc (word count)

A handy little utility is the *wc* command, short for word count. To do a word count on myfile, type

```
wc -w myfile
```

To find out how many lines the file has, type

```
wc -l myfile
```

Summary

| | |
|-----------------|---------------------------------------|
| cp file1 file2 | copy file1 and call it file2 |
| mv file1 file2 | move or rename file1 to file2 |
| rm file | remove a file |
| rmdir directory | remove a directory |
| cat file | display a file |
| more file | display a file a page at a time |
| head file | display the first few lines of a file |
| tail file | display the last few lines of a file |

```
grep 'keyword' file      search a file for keywords
wc file                  count number of lines/words/characters in
                        file
```

UNIX Tutorial Four

Redirection

Most processes initiated by UNIX commands write to the standard output (that is, they write to the terminal screen), and many take their input from the standard input (that is, they read it from the keyboard). There is also the standard error, where processes write their error messages, by default, to the terminal screen.

We have already seen one use of the `cat` command to write the contents of a file to the screen.

Now type `cat` without specifying a file to read

```
cat
```

Then type a few words on the keyboard and press the **[Return]** key.

Finally hold the **[Ctrl]** key down and press **d** (written as **^D** for short) to end the input.

What has happened?

If you run the `cat` command without specifying a file to read, it reads the standard input (the keyboard), and on receiving the "end of file" (**^D**), copies it to the standard output (the screen).

In UNIX, we can redirect the input and the output of commands.

Redirecting the Output

We use the `>` symbol to redirect the output of a command. For example, to create a file called `list1` containing a list of fruit, type

```
cat > list1
```

Then type in the names of some fruit. Press **[Return]** after each one.

```
pear
banana
apple
^D      [control d to stop]
```

The `>` redirects `cat`'s output, which normally goes to the screen, into a file called `list1`

To read the contents of the file, type

```
cat list1
```

Exercise

Using the above method, create another file called `list2` containing the following fruit:

```
Orange
plum
mango
grapefruit.
```

Read the contents of `list2`

The form `>>` appends standard output to a file. So to add more items to the file `list1`, type

```
cat >> list1
```

Then type in the names of more fruit

```
peach
grape
pineapple
^D      [control d to stop]
```

To read the contents of the file, type

```
% cat list1
```

We will now use the `cat` command to join (concatenate) `list1` and `list2` into a new file called `biglist`. Type

```
cat list1 list2 > biglist
```

What this is doing is reading the contents of `list1` and `list2` in turn, then outputting the text to the file `biglist`

To read the contents of the file, type

```
cat biglist
```

Redirecting the Input

We use the `<` symbol to redirect the input of a command.

The command `sort` alphabetically or numerically sorts a list. Type

```
sort
```

Then type in the names of some vegetables. Press [Return] after each one.

```
potato
carrot
onion
^D      [control d to stop]
```

The output will be

```
carrot
onion
potato
```

Using `<` you can redirect the input to come from a file rather than the keyboard. For example, to sort the list of fruit, type

```
sort < biglist
```

and the sorted list will be output to the screen.

To output the sorted list to a file, type,

```
sort < biglist > slist
```

Use `cat` to read the contents of the file `slist`

Pipes (Optional)

To see who is on the system with you, type

```
% who
```

One method to get a sorted list of names is to type,

```
% who > names
% sort < names
```

This is a bit slow and you have to remember to remove the temporary file called names when you have finished. What you really want to do is connect the output of the `who` command directly to the input of the `sort` command. This is exactly what **pipes** do. The symbol for a pipe is `|`

For example, typing

```
% who | sort
```

will give the same result as above, but quicker and cleaner.

To find out how many users are logged on, type

```
% who | wc -l
```

Exercise (optional)

`a2ps textfile > psfile` is the command to convert a text file to a postscript file.

`lpr -Pgoya psfile` is the command to print a postscript file to the printer "goya".

Use the above series of commands to print the text file `slist` to the postscript printer "goya".

Summary

| | |
|-----------------------------------------|------------------------------------------------------|
| <code>command > file</code> | redirect standard output to a file |
| <code>command >> file</code> | append standard output to a file |
| <code>command < file</code> | redirect standard input from a file |
| <code>command1 command2</code> | pipe the output of command1 to the input of command2 |
| <code>cat file1 file2 > file0</code> | concatenate file1 and file2 to file0 |
| <code>sort</code> | sort data |
| <code>who</code> | list users currently logged in |
| <code>a2ps textfile psfile</code> | convert text file to postscript |
| <code>lpr -Pprinter psfile</code> | print postscript file to named printer |

UNIX Tutorial Five

Wildcards

The characters `*` and `?`

The character `*` is called a wildcard, and will match against none or more character(s) in a file (or directory) name. For example, in your `unixstuff` directory, type

```
ls list*
```

This will list all files in the current directory starting with "list..."

Try typing

```
ls *list
```

This will list all files in the current directory ending with "...list"

The character `?` will match exactly one character.
 So `ls ?ouse` will match files like house and mouse, but not grouse.
 Try typing

```
ls ?list
```

Filename conventions

We should note here that a directory is merely a special type of file. So the rules and conventions for naming files apply also to directories.

In naming files, characters with special meanings such as `/` (slash) should be avoided. The safest way to name a file is to use only alphanumeric characters, that is, letters and numbers, together with `_` (underscore) and `.` (dot).

File names conventionally start with a lower-case letter, and may end with a dot followed by a group of letters indicating the contents of the file. For example, all files consisting of Pascal code may be named with the ending `.p`, for example, `prog1.p`. Then in order to list all files containing Pascal code in your home directory, you need only type `ls *.p` in that directory.

Beware: some applications give the same name to all the output files they generate. For example, some compilers, unless given the appropriate option, produced compiled files named `a.out`. Should you forget to use the flag, you are advised to rename the compiled file immediately, otherwise the next such file will overwrite it and it will be lost.

Getting Help

On-line Manuals

There are on-line manuals which gives information about most commands. The manual pages tell you which options a particular command can take, and how each option modifies the behaviour of the command. Type `man command` to read the manual page for a particular command.

For example, to find out more about the `wc` (word count) command, type

```
man wc
```

Alternatively

```
whatis wc
```

gives a one-line description of the command, but omits any information about options etc.

Apropos

When you are not sure of the exact name of a command,

```
apropos keyword
```

will give you the commands with keyword in their manual page header. For example, try typing

```
apropos copy
```

Summary

| | |
|-----------------------------|-------------------------------------------|
| <code>*</code> | match any number of characters |
| <code>?</code> | match one character |
| <code>man command</code> | read the online manual page for a command |
| <code>whatis command</code> | brief description of a command |

apropos keyword
pages

match commands with keyword in their man

UNIX Tutorial Six

File system security (access rights)

In your unixstuff directory, type

```
ls -l    (l for long!)
```

You will see that you now get lots of details about the contents of your directory, similar to the example below.

```
total 4
-rwxrw-r--  1 user1    group1      2450 Sept29 11:52 file1
drwx-----  2 user1    group1      1024 Sep 12  1998 bin
-rw-r--r--  1 user1    group1       176 Sep 12  1998 file2
drwx-----  3 user1    group1      1024 Sep 14  1998 mail
```

Each file (and directory) has associated access rights, which may be found by typing **ls -l**.

In the left-hand column is a 10 symbol 'word' consisting of the symbols d, r, w, x, -, and, occasionally, s or S. If d is present, it will be at the left hand end of the word, and indicates a directory: otherwise '-' will be the starting symbol of the word.

The 9 remaining symbols indicate the permissions, or access rights, and are taken as three groups of 3.

- ? The left group of 3 gives the file permissions for the owner of the file (or directory) (ee51ab in the above example);
- ? the middle group gives the permissions for the group of people to whom the file (or directory) belongs (eebeng95 in the above example);
- ? the rightmost group gives the permissions for everyone else.

The symbols r, w, etc., have slightly different meanings depending on whether they refer to a simple file or to a directory.

Access rights on files.

- ? r (or -), indicates read permission (or otherwise), that is, the presence or absence of permission to read and copy the file
- ? w (or -), indicates write permission (or otherwise), that is, the permission (or otherwise) to change a file
- ? x (or -), indicates execution permission (or otherwise), that is, the permission to execute a file, where appropriate

Access rights on directories.

- ? r allows users to list files in the directory;

- ? w means that users may delete files from the directory or move files into it;
- ? x means the right to access files in the directory. This implies that you may read files in the directory provided you have read permission on the individual files.

So, in order to read a file, you must have execute permission on the directory containing that file, and hence on any directory containing that directory as a subdirectory, and so on, up the tree.

Some examples

```
-rwxrwxrwx    a file that everyone can read, write and execute.
-rw-----    a file that only the owner can read and write- no-one
else
your          can read or write and no-one has execution rights (e.g.
              mailbox file).
```

Changing access rights

chmod (changing a file mode)

Only the owner of a file can use chmod to change the permissions of a file. The options of chmod are as follows

| Symbol | Meaning |
|--------|----------------------|
| u | user |
| g | group |
| o | other |
| a | all |
| r | read |
| w | write (and delete) |
| x | execute |
| + | add permission |
| - | take away permission |

For example, to remove read write and execute permissions on the file `biglist` for the group and others, type

```
chmod go-rwx biglist
```

This will leave the other permissions unaffected.

To give read and write permissions on the file `biglist` to all,

```
chmod a+rw biglist
```

Try changing the file permissions on the file `myfile`

Use `ls -l` to check that the permissions have changed.

Processes and Jobs

A process is an executing program identified by a unique PID (process identifier). To see information about your processes, with their associated PID and status, type

```
ps
```

A process may be in the foreground, in the background, or be suspended. In general the shell does not return the UNIX prompt until the current process has finished executing.

Some processes take a long time to run and hold up the terminal. Backgrounding a long process has the effect that the UNIX prompt is returned immediately, and other tasks can be carried out while the original process continues executing.

To background a process

To background a process, type an `&` at the end of the command line. For example, the command `sleep` waits a given number of seconds before continuing. Type

```
sleep 10
```

This will wait 10 seconds before returning the command prompt `%`. Until the command prompt is returned, you can do nothing except wait.

To run `sleep` in the background, type

```
sleep 10 &
[1] 6259
```

The `&` runs the job in the background and returns the prompt straight away, allowing you to run other programs while waiting for that one to finish. The first line in the above example is typed in by the user; the next line, indicating job number and PID, is returned by the machine. The user is notified of a job number (numbered from 1) enclosed in square brackets, together with a PID and is notified when a background process is finished. Backgrounding is useful for jobs which will take a long time to complete.

Backgrounding a current foreground process

At the prompt, type

```
sleep 100
```

You can suspend the process running in the foreground by holding down the control key and typing `z` (written as `^Z`). Then to put it in the background, type

```
bg
```

Note: do not background programs that require user interaction e.g. `elm`

Listing suspended and background processes

When a process is running, backgrounded or suspended, it will be entered onto a list along with a job number. To examine this list, type

```
jobs
```

An example of a job list could be

```
[1]    Suspended          sleep 100
[2]    Running            netscape
[3]    Suspended          asedit
```

To restart (foreground) a suspended process, type

```
% fg %job_number
```

For example, to restart `sleep 100`, type

```
% fg %1
```

Typing `fg` with no job number foregrounds the last suspended process.

Killing a process

It is sometimes necessary to kill a process (for example, when an executing program is in an infinite loop)

To kill a job running in the foreground, type `^C` (control c). For example, run `sleep 100` then kill it with `^C`

For a suspended or background process, type

```
kill %job_number
```

For example, run **sleep 100 &** then type **jobs** to see its job number. If it is job number 4, type

```
kill %4
```

To check whether this has worked, examine the job list again to see if the process has been removed.

Alternatively, processes can be killed by finding their process numbers (PIDs) and using kill PID_number.

Run **sleep 100 &** again, then type

```
ps
```

```

PID      TT      S          TIME    COMMAND
20077   pts/5   S          0:05    sleep 100
21563   pts/5   T          0:00    netscape
21873   pts/5   S          0:25    asedit
```

To kill off the process sleep 100, type

```
kill 20077
```

and then type **ps** again to see if it has been removed from the list.

If a process refuses to be killed, uses the **-9** option, i.e. type

```
kill -9 20077
```

Note: It is not possible to kill off other users' processes

Summary

| | |
|----------------------|-------------------------------------------|
| ls -lag | list access rights for all files |
| chmod [options] file | change access rights for named file |
| command & | run command in background |
| ^C | kill the job running in the foreground |
| ^Z | suspend the job running in the foreground |
| bg | background the suspended job |
| jobs | list current jobs |
| fg %1 | foreground job number 1 |
| kill %1 | kill job number 1 |
| ps | list current processes |
| kill 26152 | kill process number 26152 |

UNIX Tutorial Seven

Other useful UNIX commands

Quota (optional)

All students are allocated a certain amount of disk space on the file system for their personal files, usually about 5 Megabytes (equivalent to 4 floppy disks worth). If you go "over-quota", you are given 7 days to remove excess files.

| |
|--------------------------------------------------------------------|
| To check your current quota and how much of it you have used, type |
|--------------------------------------------------------------------|

```
quota -v
```

df

The df command reports on the space left on the file system. For example, to find out how much space is left on the fileserver, type

```
df .
```

du

The du command outputs the number of kilobytes used by each subdirectory. Useful if you have gone over quota and you want to find out which directory has the most files. In your home-directory, type

```
du
```

compress

This reduces the size of a file, thus freeing valuable disk space. For example, type

```
ls -l myfile
```

and note the size of the file. Then to compress `myfile`, type

```
compress myfile
```

This will compress the file and place it in a file called `myfile.Z`

To see the change in size, type `ls -l` again.

To uncompress the file, use the **uncompress** command.

```
uncompress myfile.Z
```

file

`file` classifies the named files according to the type of data they contain, for example `ascii` (text), pictures, compressed data, etc.. To report on all files in your home directory, type

```
file *
```

? [file](#) - identifies the "type" of file. The command syntax is:

?

? `file filename`

For example:

```
file *           - reports all files in current
                  directory and their types. The
                  output might appear as shown below:

about.html:      ascii text
bin:             directory
staff.directory: English text
bggen:          executable or object module not stripped
bmbinc:         commands text
```

```

machines.sp1:      [nt]roff, tbl, or eqn input text
man2html:         executable or object module not stripped
man2html.c:       ascii text

```

? [find](#) - finds files. The syntax of this command is:

?

```
?      find pathname -name filename -print
```

The pathname defines the directory to start from. Each subdirectory of this directory will be searched.

The -print option must be used to display results.

You can define the filename using wildcards. If these are used, the filename must be placed in 'quotes'.

```

find . -name mtg_jan92 -print - looks for the file
                                mtg_jan92 in current
                                directory
find ~/ -name README -print  - looks for files called
                                README throughout your
                                home directory
find . -name '*.fm' -print   - looks for all files with
                                .fm suffix in current
                                directory
find /usr/local -name gnu -type d -print
                                - looks for a directory
                                called gnu within the
                                /usr/local directory

```

? [diff](#) - comparing two files or directories. Indicates which lines need be added (a), deleted (d) or changed (c). Lines in file1 are identified with a (<) symbol: lines in file2 with a (>) symbol

?

```

?      diff file1 file2          - compares file1 to file2
?      diff -iw file1 file2     - compares two files ignoring
?                                letter case and spaces
?      diff dir1 dir2          - compares two directories
?                                showing files which are
?                                unique to each and also,
?                                line by line differences
?                                between any files in common.

```

For example, if file1 and file2 are:

```

John erpl08@ed          John erpl08@ed
Joe  CZT@cern.ch       Joe  CZT@cern.ch
Kim  ks@x.co           Jean JRS@pollux.ucs.co
Keith keith@festival   Jim  jim@frolix8
                                Kim  ks@x.co
                                Keith keith@festival

```

Using the diff command: `diff file1 file2` Yields the output:

```
2a3,4
> Jean JRS@pollux.ucs.co
> Jim jim@frolix8
```

Which means that to make these files match you need to add (a) lines 3 and 4 (3,4) of file2 (>) after line 2 in file1.

- ? [sdiff](#) - similar to diff, but displays each line of the two files side by side, making it easier for you to see the differences between them

Lines that are different are shown with a | symbol. Lines unique to file1 are identified by a < symbol; lines unique to file2 with a > symbol. Identical lines appear next to each other. The option -w 80 is used to set the width of the output from the command to 80 characters. The default is 130 characters.

```
sdiff -w 80 file1 file2
Mike erpl08@ed          | John erpl08@ed
Joe CZT@cern.ch        | Joe CZT@cern.ch
                        | > Jean JRS@pollux.ucs.co
                        | > Jim jim@frolix8
Kim ks@x.co            | Kim ks@x.co
Sam s.wally@aston      | <
Keith keith@festival    | Keith keith@festival
```
