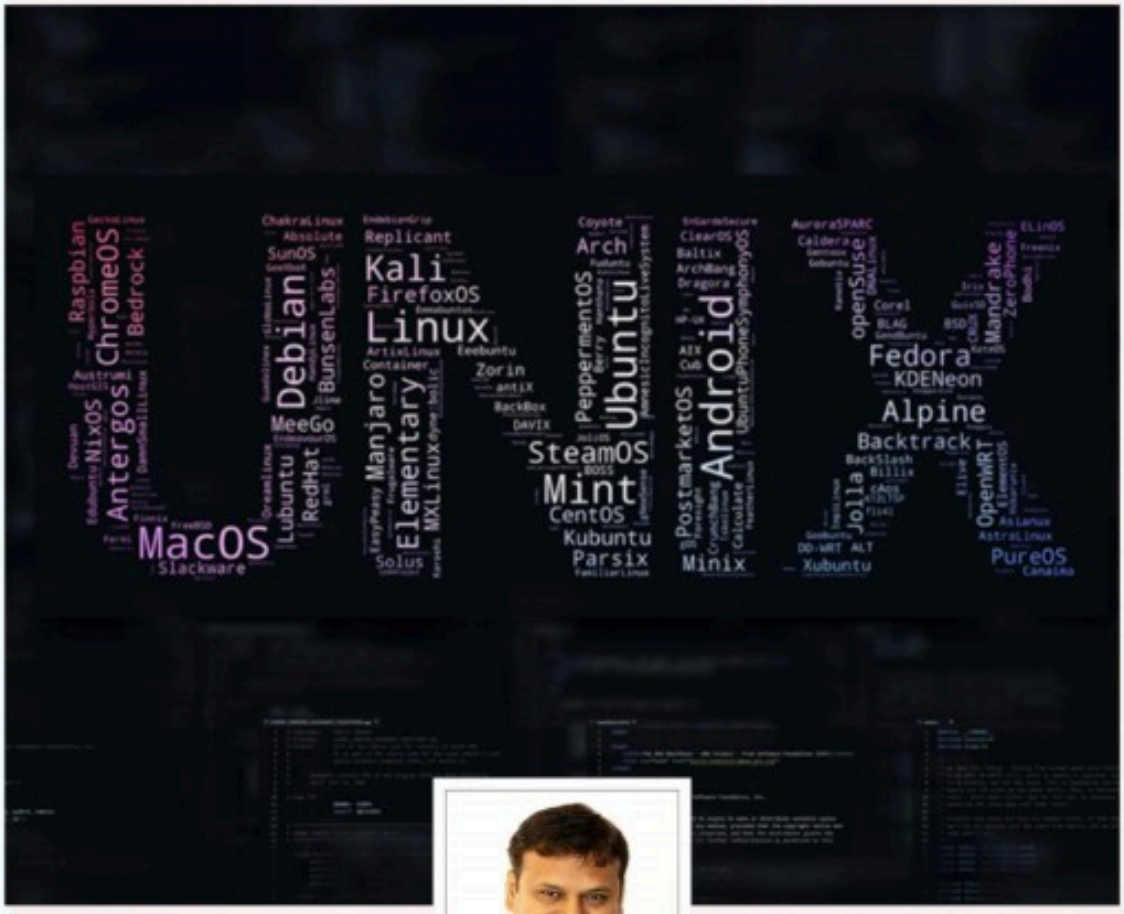
unix shell scripting

Jump2Learn
The Online Learning Place

UNIX
& SHELL PROGRAMMING
CHAPTER 2 - OVERVIEW



AUTHOR
Mr. Manish Dolia
M.C.A.,Ph.D.(Pursuing)
DOLAT USHA INSTITUTE OF
APPLIED SCIENCES, VALSAD

2.1 Logging in & out

Login Unix When you first connect to a Unix system, you usually see a prompt such as the following:

LOGIN:

TO LOG IN

- Have your userid (user identification) and password ready. Contact your system administrator if you don't have these yet.
- Type your userid at the login prompt, then press ENTER. Your userid is casesensitive, so be sure you type it exactly as your system administrator has instructed.
- Type your password at the password prompt, then press ENTER. Your password is also case-sensitive.
- If you provide the correct userid and password, then you will be allowed to enter into the system. Read the information and messages that comes up on the screen, which is as follows.

login : **manish**

manish's password:
Last login: Sun Jun 14 09:32:32 2020 from 62.61.164.73
$
You will be provided with a command prompt (sometime called the $ prompt) where you type all your commands.
For example, to check calendar, you need to type the cal command as follows – $ cal June 2020

Change Password

All Unix systems require passwords to help ensure that your files and data remain your own and that the system itself is secure from hackers and crackers.

Following are the steps to change your password –

Step 1: To start, type password at the command prompt as shown below.
Step 2: Enter your old password, the one you're currently using.
Step 3: Type in your new password. Always keep your password complex enough so that nobody can guess it. But make sure, you remember it.
Step 4: You must verify the password by typing it again.

```
$ passwd
Changing password for amrood (current)
Unix password:******
New Unix password:*******
Retype new Unix password:*******
passwd: all authentication tokens updated successfully
$
```

Note – We have added asterisk (*) here just to show the location where you need to enter the current and new passwords otherwise at your system. It does not show you any character when you type.

LISTING DIRECTORIES AND FILES

All data in Unix is organized into files. All files are organized into directories.

These directories are organized into a tree-like structure called the file system. You can use the ls command to list out all the files or directories available in a directory.

Following is the example of using ls command with -l option.

```
$ ls -l total 19621
drwxrwxr-x 2 amrood amrood 4096 Dec 25 09:59 uml
```

WHO ARE YOU?

While you're logged into the system, you might be willing to know : Who am I? The easiest way to find out "who you are" is to enter the whoami command –

```
$ whoami
manish
$
```

Try it on your system.

This command lists the account name associated with the current login. You can try who am i command as well to get information about yourself.

WHO IS LOGGED IN?

Sometime you might be interested to know who is logged in to the computer at the same time. There are three commands available to get you this information, based on how much you wish to know about the other users:

users, who, and w.

```
$ users
Manish yatin divyesh
```

LOGGING OUT

When you finish your session, you need to log out of the system. This is to ensure that nobody else accesses your files.

TO LOG OUT

• Just type the logout command at the command prompt, and the system will clean up everything and break the connection.

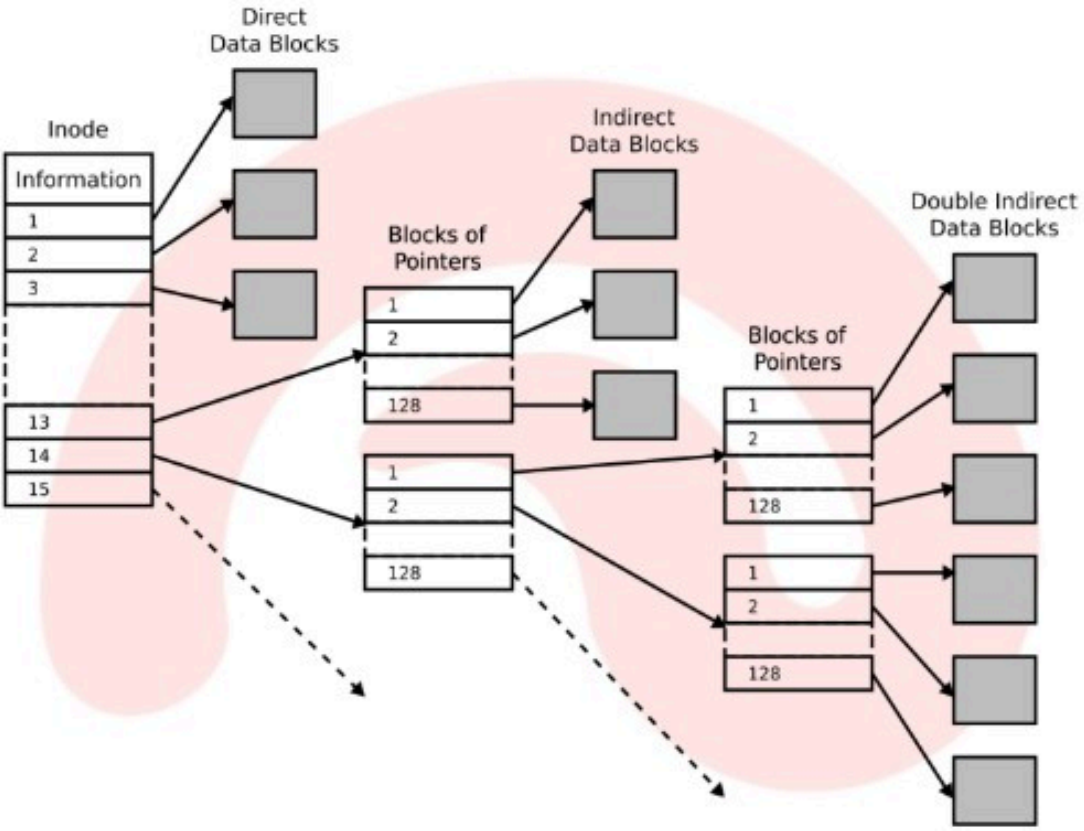
2.2 I NODE AND FILE STRUCTURE

Inode Basics

An Inode number points to an Inode. An Inode is a data structure that stores the following information about a file :

- Size of file
- Device ID
- User ID of the file
- Group ID of the file
- The file mode information and access privileges for owner, group and others
- File protection flags
- The timestamps for file creation, modification etc
- link counter to determine the number of hard links
- Pointers to the blocks storing file's contents

Please note that the above list is not exhaustive. Also, the name of the file is not stored in Inodes.



When a file is created inside a directory then the file-name and Inode number are assigned to file. These two entries are associated with every file in a directory. The user might think that the directory contains the complete file and all the extra information related to it but this might not be the case always. So we see that a directory associates a file name with its Inode number.

When a user tries to access the file or any information related to the file then he/she uses the file name to do so but internally the file-name is first mapped with its Inode number stored in a table. Then through that Inode number the corresponding Inode is accessed. There is a table (Inode table) where this mapping of Inode numbers with the respective Inodes is provided.

WHY NO FILE-NAME IN INODE INFORMATION?

As pointed out earlier, there is no entry for file name in the Inode, rather the file name is kept as a separate entry parallel to Inode number. The reason for separating out file name from the other information related to same file is for maintaining hard-links to files. This means that once all the other information is separated out from the file name then we can have various file names which point to same Inode.

For example :

$ touch a

$ ln a a1

$ ls –al

drwxr-xr-x 48 himanshu himanshu 4096 2012-01-14 16:30 .

drwxr-xr-x 3 root root 4096 2011-03-12 06:24 ..

-rw-r--r-- 2 himanshu family 0 2012-01-14 16:29 a

-rw-r--r-- 2 himanshu family 0 2012-01-14 16:29 a1

In the above output, we created a file 'a' and then created a hard link a1. Now when the command 'ls -al' is run, we can see the details of both 'a' and 'a1'. We see that both the files are indistinguishable. Look at the second entry in the output. This entry specifies number of hard links to the file. In this case the entry has value '2' for both the files.

Note that Hard links cannot be created on different file systems and also they cannot be created for directories.

WHEN ARE INODES CREATED?

As we all now know that Inode is a data structure that contains information of a file. Since data structures occupy storage then an obvious question arises about when the Inodes are created in a system? Well, space for Inodes is allocated when the operating system or a new file system is installed and when it does its initial structuring. So this way we can see that in a file system, maximum number of Inodes and hence maximum number of files are set.

Now, the above concept brings up another interesting fact. A file system can run out of space in two ways :

- No space for adding new data is left
- All the Inodes are consumed.

Well, the first way is pretty obvious but we need to look at the second way. Yes, its possible that a case arises where we have free storage space but still we cannot add any new data in file system because all the Inodes are consumed. This may happen in a case where file system contains very large number of very small sized files. This will consume all the Inodes and though there would be free space from a Hard-disk-drive point of view but from file system point of view no Inode available to store any new file.

The above use-case is possible but less encountered because on a typical system the average file size is more than 2KB which makes it more prone to running out of hard disk space first. But, nevertheless there exists an algorithm which is used to create number of Inodes in a file system. This algorithm takes into consideration the size of the file system and average file size. The user can tweak the number of Inodes while creating the file system.

Commands to access Inode numbers

FOLLOWING ARE SOME COMMANDS TO ACCESS THE INODE NUMBERS FOR FILES :

1) Ls -i Command

The flag -i is used to print the Inode number for each file.

$ ls -i

1448240 a 1441807 Desktop 1447344 mydata 1441813 Pictures 1442737 testfile 1448145 worm

1448240 a1 1441811 Documents 1442707 my_ls 1442445 practice 1442739 test.py

1447139 alpha 1441808 Downloads 1447278 my_ls_alpha.c 1441810 Public 1447099 Unsaved Document 1

1447478 article_function_pointer.txt 1575132 google 1447274 my_ls.c 1441809 Templates 1441814 Videos

1442390 chmodOctal.txt 1441812 Music 1442363 output.log 1448800 testdisk.log 1575133 vlc

See that the Inode number for ‘a’ and ‘a1’ are same as we created ‘a1’ as hard link.

2) DF -I COMMAND

df -i command displays the inode information of the file system.

```
$ df -i
Filesystem      Inodes  IUsed  IFree IUse% Mounted on
/dev/sda1       1875968 293264 1582704   16% /
none             210613    764  209849    1% /dev
none             213415      9  213406    1% /dev/shm
none             213415     63  213352    1% /var/run
none             213415      1  213414    1% /var/lock
/dev/sda2       7643136 156663 7486473    3% /home
```

The flag -i is used for displaying Inode information.

3) STAT COMMAND

Stat command is used to display file statistics that also displays inode number of a file

```
$ stat a

File: `a'

Size: 0 Blocks: 0 IO Block: 4096 regular empty file

Device: 805h/2053d Inode: 1448240 Links: 2

Access: (0644/-rw-r--r--) Uid: ( 1000/himanshu) Gid: ( 1001/ family)

Access: 2012-01-14 16:30:04.871719357 +0530

Modify: 2012-01-14 16:29:50.918267873 +0530

Change: 2012-01-14 16:30:03.858251514 +0530
```

Example Usage Scenario of an Inode number

1. Suppose there exist a file name with some special character in it. For example: "ab*
2. Try to remove it normally using rm command; you will not be able to remove it.
3. However using the inode number of this file you can remove it.

Lets see these steps in this example :

1) CHECK IF THE FILE EXISTS:

$ ls -i
1448240 a 1447274 my_ls.c
1448240 a1 1442363 output.log
1448239 "ab* 1441813 Pictures
1447139 alpha
So we have a file with name "ab* in this directory

2) TRY TO REMOVE IT NORMALLY:

$ rm "ab*
> ^C
$ rm "ab*
> ^C
$

3) REMOVE THE FILE USING INODE NUMBER:

you can search for a file using inode number and delete it.
$ find . -inum 1448239 -exec rm -i {} \;
rm: remove regular empty file `./"ab*'? y

$ ls -i
1448240 a 1447274 my_ls.c
1448240 a1 1442363 output.log
1447139 alpha 1441813 Pictures

So we used the find command specifying the Inode number of the file we need to delete. The file got deleted. Though we could have deleted the file otherwise also by using the command rm \"ab* instead of using the complicated find command example above but still I used it to demonstrate one of the use of Inode numbers for users.

2.3 FILE SYSTEM STRUCTURE AND FEATURES

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 UNIX FILE SYSTEM CONTAINS SEVERAL DIFFERENT TYPES OF FILES:

- **ORDINARY FILES**
  - Used to store your information, such as some text you have written or an image you have drawn. This is the type of file that you usually work with.
  - Always located within/under a directory file
  - Do not contain other files

- **DIRECTORIES**
  - Branching points in the hierarchical tree
  - Used to organize groups of files
  - May contain ordinary files, special files or other directories
  - Never contain "real" information which you would work with (such as text). Basically, just used for organizing files.
  - All files are descendants of the root directory, (named /) located at the top of the tree.

- **SPECIAL FILES**
  - Used to represent a real physical device such as a printer, tape drive or terminal, used for Input/Ouput (I/O) operations
  - Unix considers any device attached to the system to be a file - including your terminal:
    - By default, a command treats your terminal as the standard input file (stdin) from which to read its input
    - Your terminal is also treated as the standard output file (stdout) to which a command's output is sent
    - Stdin and stdout will be discussed in more detail later

- Two types of I/O: character and block
- Usually only found under directories named /dev

- **PIPES**
- UNIX allows you to link commands together using a pipe. The pipe acts a temporary file which only exists to hold data from one command until it is read by another
- For example, to pipe the output from one command into another command:

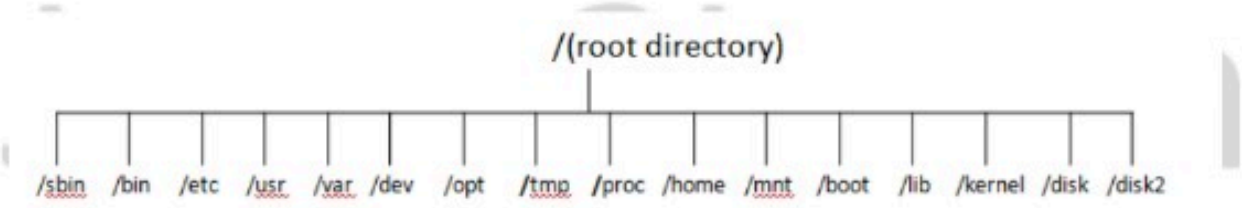
WHO | WC -L

This command will tell you how many users are currently logged into the system. The standard output from the who command is a list of all the users currently logged into the system. This output is piped into the wc command as its standard input. Used with the -l option this command counts the numbers of lines in the standard input and displays the result on its standard output - your terminal.

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 (written as a slash /)

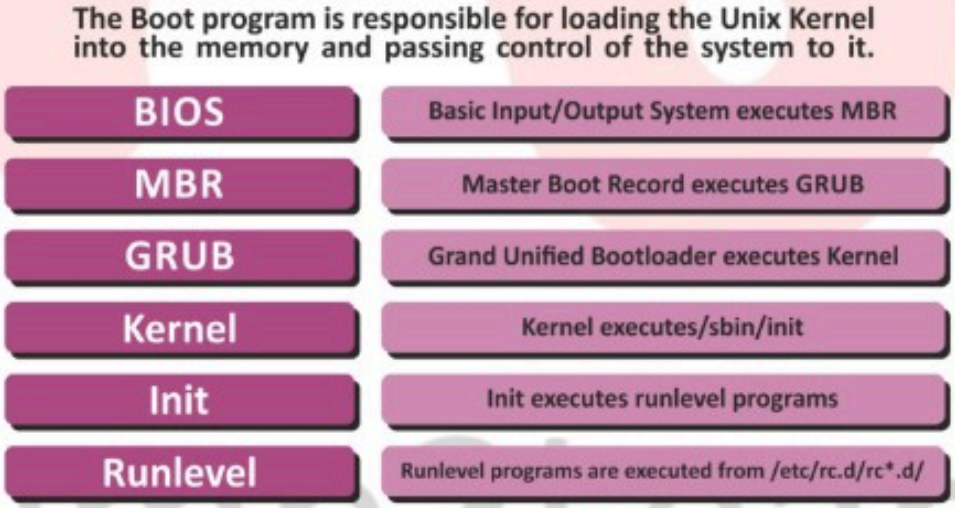
All the data of Unix is organized into files. All files are then organized into directories. These directories are further organized into a tree-like structure called the filesystem.



/ (root directory) : It is top most directory in Unix operating system which maintains all the sub-directories and it is also named as root directory.
/sbin : It is a directory which contains all the super user executable commands.
/bin : It is a directory which contains all normal user executable commands.
/etc : It is a directory which contains all system configuration files.

/usr : It is a directory which contains manual pages of operating system.
/var : It is a directory which contains mails, error messages and log messages.
/dev : It is a directory which contains all the logical names of physical devices.
/opt : It is a directory which used to maintain all the third party software's.
/tmp : It Holds temporary files used between system boots.
/proc : Contains all processes marked as a file by process number or other information that is dynamic to the system.
/home : Contains the home directory for users and other accounts.
/mnt : Used to mount other temporary file systems, such as cdrom and floppy for the CD-ROM drive and floppy diskette drive, respectively.
/boot : Contains files for booting the system.
/lib : Contains shared library files and sometimes other kernel-related files.
/kernel : Contains kernel files.
/disk1 : It is a directory which is used to store any application related data.
/disk2 : It is a directory which is used to store any application related data.

2.4 BOOTING SEQUENCE & INIT PROCESS



Booting process of UNIX

Init is the parent of all processes, executed by the kernel during the booting of a system. Its principle role is to create processes from a script stored in the file **/etc/inittab.** It usually has entries which cause init to spawn gettys on each line that users can log in. It controls autonomous processes required by any particular system.

After reading this file, how the system should be set up in each runlevel is determined by init and also set default runlevel. Init starts all background process after setting default runlevel for the system.

Runlevels

Runlevel, a software configuration of the system which allows only a selected group of processes to exist. The processes produced by init for each of these runlevels are defined in the **/etc/inittab** file.
Init can be in one of these eight runlevels: 0-6 and S or s. The runlevel can be changed by having a privileged user run telinit, which sends appropriate signals to init, telling it which runlevel to change to.

Here,

- 0,1,6-reserved runlevels.
- S or s are same.
- 7-9 are valid runlevels. though not really documented as "traditional" Unix variants don't use them. runlevels S and s are in fact the same. Internally they are aliases for the same runlevel.

Runlevels	Functions
0	To halt the system
1	To get the system down into single user mode
2	To get multiuser mode without networking
3	To get multiuser mode with networking
4	Not used
5	To get multiuser mode with networking and X windows
6	To reboot the system
S or s	Not used directly.

Booting

After invoking init as the last step of the kernel boot sequence, it sees if an entry of the type **initdefault** is present in the file **/etc/inittab**. The **initdefault** entry determines the initial runlevel of the system. If no such entry (or no **/etc/inittab** at all) is present there, a runlevel must be entered at the system console.

Changing Runlevels

After specifying all the processes, **init** waits for one of its descendant processes to die, a powerfail signal, or until it is signaled by telinit to change the system's runlevel. It re-examines the **/etc/inittab** file, when one of the above three conditions occurs.

init still waits for one of the above three conditions to occur. For providing an instantaneous response, the telinit Q or q command can wake up init to re-examine the/etc/inittab file.

If init is not in single user mode and receives a powerfail signal (SIGPWR), it reads the file /etc/powerstatus. Then it starts a command based on the contents of this file

Tag	Description
F(AIL)	Power is failing, UPS is providing the power. Execute the powerwait and powerfail entries.
O(K)	Power has been restored, execute the powerokwait entries.
L(OW)	The power is failing and the UPS has a low battery. Execute the powerfailnow entries.

If /etc/powerstatus contains anything else then the letters F, O or L or doesn't exist, init will behave as if it has read the letter F.

Usage of SIGPWR and /etc/powerstatus is discouraged. If Someone wanting to interact with init should use the /dev/initctl control channel - see the source code of the sysvinit package for more documentation about this.

When init is requested to change the runlevel, it sends the warning signal SIGTERM to all processes that are undefined in the new runlevel. Then It waits 5 seconds before forcibly terminating these processes via the SIGKILL signal.

Init assumes that all these processes and their descendants remain in the same process group which init originally created for them. It will not receive these signals, if any process changes its process group affiliation. Such processes need to be terminated separately.

TELINIT

/sbin/telinit is linked to /sbin/init which takes a one-character argument and signals init to perform the appropriate action. The following arguments serve as directives to telinit –

Tag	Description
0,1,2,3,4,5 or 6	tell init to switch to the specified run level.
a, b, c	tell init to process only those /etc/inittab file entries having runlevel a, b or c.
Q or q	tell init to re-examine the /etc/inittab file.
S or s	tell init to switch to single user mode.
U or u	tell init to re-execute itself (preserving the state). No re-examining of/etc/inittab file happens. Request would be silently ignored, if Run level is not one of Ss12345.

It can also tell init how long it should wait between sending processes the SIGTERM and SIGKILL signals. 5 seconds is the default, but this can be changed with the -t sec option.

2.5 FILE ACCESS PERMISSIONS

Unix Permissions: File Permissions with Examples

ACCESS TO A FILE HAS THREE LEVELS:

- Read permission – If authorized, the user can read the contents of the file.
- Write permission – If authorized, the user can modify the file.
- Execute permission – If authorized, the user can execute the file as a program.

EACH FILE IS ASSOCIATED WITH A SET OF IDENTIFIERS THAT ARE USED TO DETERMINE WHO CAN ACCESS THE FILE:

- User ID (UID) – Specifies the user that owns the file. By default, this is the creator of the file.
- Group ID (GID) – Specifies the user-group that the file belongs to.

FINALLY, THERE ARE THREE SETS OF ACCESS PERMISSIONS ASSOCIATED WITH EACH FILE:

- User permission – Specifies the level of access given to the user matching the file's UID.
- Group permission – Specifies the level of access given to users in groups matching the file's GID.
- Others permission – Specifies the level of access given to users without a matching UID or GID.

Together, this scheme of access controls makes the Unix system extremely secure while simultaneously providing the flexibility required of a multi-user system.

The ls -l command can be used to view the permissions associated with each of the files in the current folder.

Example output of this command is given below.

Example:

FLAGS LINKS OWNER GROUP SIZE MODIFIED-DATE NAME TOTAL OF 24

drwxr-xr-x 7 user staff 224 Jun 21 15:26 .

drwxrwxrwx 8 user staff 576 Jun 21 15:02.

-rw-r--r-- 1 user staff 6 Jun 21 15:04 .hfile

drwxr-xr-x 3 user staff 96 Jun 21 15:17 dir1

drwxr-xr-x 2 user staff 64 Jun 21 15:04 dir2

-rw-r--r-- 1 user staff 39 Jun 21 15:37 file1

-rw-r--r-- 1 user staff 35 Jun 21 15:32 file2

In this output, the 'total 24' indicates the total number of blocks occupied by the listed files.

THE REMAINING COLUMNS ARE:

- flags – A collection of flags indicating the file mode and the file permissions.
- links – The number of links associated with the file.
- owner – The UID that owns the file.
- group – The GIDs associated with the file.
- size – The size of the file in bytes.
- modified-date – The month, date, hour and minute of the last modification to the file.
- name – The name of the file or directory.

THE FLAGS IN THE FIRST COLUMN SPECIFY THE FILE MODE AND THE DIFFERENT SETS OF PERMISSIONS:

#1) THE FIRST CHARACTER INDICATES THE TYPE OF FILE:

- : represents an ordinary file
- d: represents a directory
- c: represents a character device file
- b: represents a block device file

#2) THE NEXT THREE CHARACTERS INDICATE USER PERMISSIONS:

- The first of these three indicates whether the user has read permission:
  - -: indicates that the user does not have read permission.
  - r: indicates that the user has read permission.
- The second character indicates whether the user has to write permission:
  - -: indicates the user does not have write permission.
  - w: indicates the user has to write permission.
- The last character indicates whether the user has executed permission:
  - -: indicates that the user does not have to execute permission.
  - x: indicates that the user has executed permission.

#3) The next three characters indicate group permissions, similar to the user permissions above.

#4) The final three characters indicate public permissions, similar to the user permissions above.

In case the file is an ordinary file, read permission allows the user to open the file and examine its contents. Write permission allows the user to modify the contents of the file. And execute permission allows the user to run the file as a program.

In case the file is a directory, read permission allows the user to list the contents of the directory. Write permission allows the users to create a new file in the directory, and to remove a file or directory from it. Execute permission allows the user to run a search on the directory.

UNIX COMMAND TO CHANGE THE ACCESS PERMISSIONS

UNIX PROVIDES A NUMBER OF COMMAND-LINE TOOLS TO CHANGE THE ACCESS PERMISSIONS:

Note that only the owner of the file can change the access permissions.

To change file and directory permissions, use the command chmod (change mode). The owner of a file can change the permissions for user (u), group (g), or others (o) by adding (+) or subtracting (-) the read, write, and execute permissions. There are two basic ways of using chmod to change file permissions: The symbolic method and the absolute form.

Symbolic method

The first and probably easiest way is the relative (or symbolic) method, which lets you specify permissions with single letter abbreviations. A chmod command using this method consists of at least three parts from the following lists:

Access class	Operator	Access Type
u (user)	+ (add access)	r (read)
g (group)	- (remove access)	w (write)
o (other)	#NAME?	x (execute)
a (all: u, g, and o)		

For example, to add permission for everyone to read a file in the current directory named myfile, at the Unix prompt, enter:

chmod a+r myfile

The a stands for "all", the + for "add", and the r for "read".

1. chmod: change file access permissions

description: This command is used to change the file permissions. These permissions are read, write and execute permission for the owner, group, and others.

syntax (symbolic mode):

chmod [ugoa][[+-=][mode]] file

The first optional parameter indicates who – this can be (u)ser, (g)roup, (o)thers or (a)ll
The second optional parameter indicates opcode – this can be for adding (+), removing (-) or assigning (=) permission.
The third optional parameter indicates the mode – this can be (r)ead, (w)rite, or e(x)ecute.

Example: Add write permission for user, group and others for file1
$ ls -l
-rw-r–r– 1 user staff 39 Jun 21 15:37 file1
-rw-r–r– 1 user staff 35 Jun 21 15:32 file2
$ chmod ugo+w file1

$ ls -l
-rw-rw-rw- 1 user staff 39 Jun 21 15:37 file1
-rw-r–r– 1 user staff 35 Jun 21 15:32 file2
$ chmod o-w file1
$ ls -l
-rw-rw-r– 1 user staff 39 Jun 21 15:37 file1
-rw-r–r– 1 user staff 35 Jun 21 15:32 file2
syntax (numeric mode):

chmod [mode] file
The mode is a combination of three digits – the first digit indicates the permission for the user, the second digit for the group, and the third digit for others.
Each digit is computed by adding the associated permissions. Read permission is '4', write permission is '2' and execute permission is '1'.

Example: Give read/write/execute permission to the user, read/execute permission to the group, and execute permission to others.
$ ls -l
-rw-r–r– 1 user staff 39 Jun 21 15:37 file1
-rw-r–r– 1 user staff 35 Jun 21 15:32 file2
$ chmod 777 file1
$ ls -l
-rwxrwxrwx 1 user staff 39 Jun 21 15:37 file1
-rw-r–r– 1 user staff 35 Jun 21 15:32 file2

2. chown: change ownership of the file.

description: Only the owner of the file has the rights to change the file ownership.
syntax: chown [owner] [file]
Example: Change the owner of file1 to user2 assuming that it is currently owned by the current user
$ chown user2 file1

3. chgrp: change the group ownership of the file.

description: Only the owner of the file has the rights to change the file ownership.
syntax: chgrp [group] [file]
Example: Change group of file1 to group2 assuming it is currently owned by the current user.
$ chgrp group2 file1
While creating a new file, Unix sets the default file permissions. Unix uses the value stored in a variable called umask to decide the default permissions. The umask value tells Unix which of the three sets of permissions need to be disabled.
The flag consists of three octal digits, each representing the permissions masks for the user, the group, and others. The default permissions are determined by subtracting the umask value from '777' for directories and '666' for files. The default value of the umask is '022'.

4. umask: change default access permissions
description: This command is used to set the default file permissions. These permissions are read, write and execute permission for owner, group, and others.
syntax: umask [mode]

The mode is a combination of three digits – the first digit indicates the permission for the user, the second digit for the group, and the third digit for others.
Each digit is computed by adding the associated permissions. Read permission is '4', write permission is '2' and execute permission is '1'.

Example: Give read/write/execute permission to the user, and no permissions to group or others. i.e. the permission for files will be 600, and for directories will be 700.
$ umask 077

Example: Give read/write/execute permission to the user, read/execute permissions to group or others for directories and read-only permission to group or others for other files. i.e. the permission for files will be 644, and for directories will be 755.

$ umask 022

PRACTICAL WITH FILE AND PROCESS COMMAND

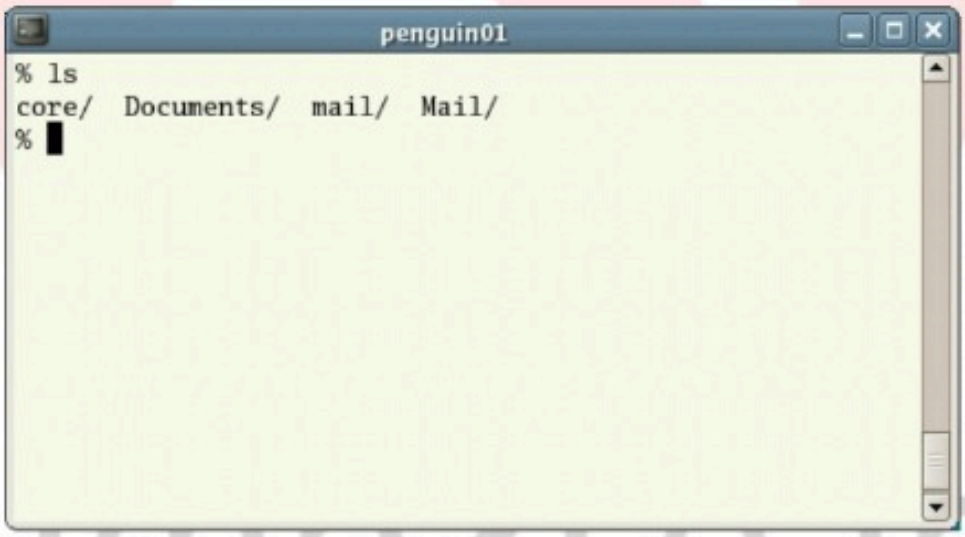
1.1 LISTING FILES AND DIRECTORIES

LS (LIST)

When you first login, your current working directory is your home directory. Your home directory has the same name as your user-name, for example, ee91ab, and it is where your personal files and subdirectories are saved. To find out what is in your home directory, type

% ls

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

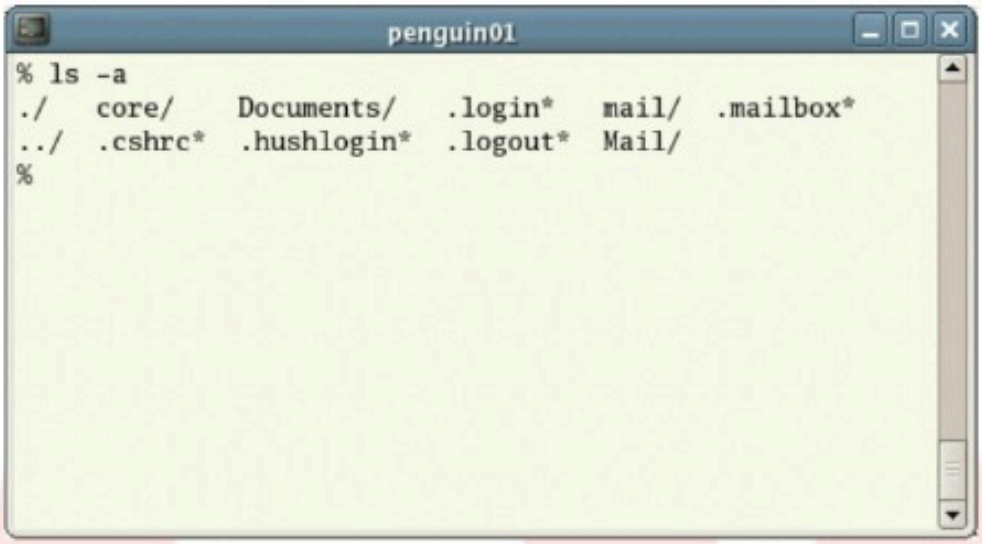


There may be no files visible in your home directory, in which case, the UNIX prompt will be returned. Alternatively, there may already be some files inserted by the System Administrator when your account was created. ls does not, in fact, cause all the files in your home directory to be listed, but 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!!!

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

% ls –a

As you can see, ls -a lists files that are normally hidden.



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. 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)

1.2 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

1.3 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 1a
Make another directory inside the unixstuff directory called backups

1.4 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 (..)

THE CURRENT DIRECTORY (.)

In UNIX, (.) means the current directory, so typing
% cd .
NOTE: there is a space between cd and the dot
means stay where you are (the unix stuff 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.

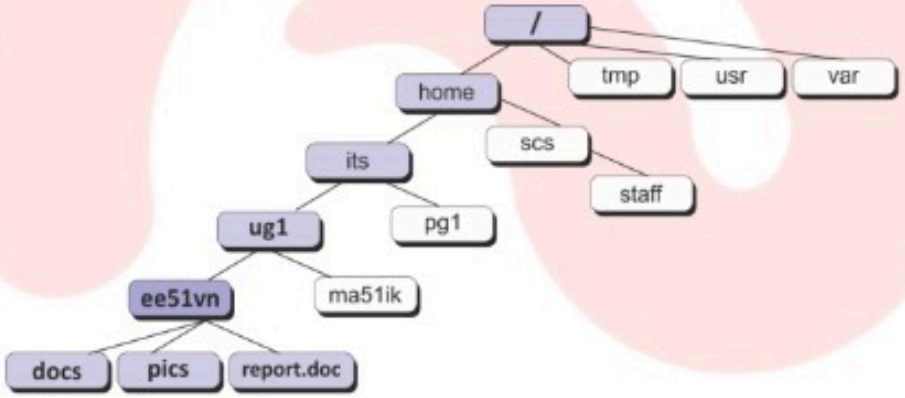
THE PARENT DIRECTORY (..)

(..) 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.

1.5 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, type cd to get back to your home-directory and then type
% pwd
The full pathname will look something like this -
/home/its/ug1/ee51vn
which means that ee51vn (your home directory) is in the sub-directory ug1 (the group directory),which in turn is located in the its sub-directory, which is in the home sub-directory, which is in the top-level root directory called " / " .



Exercise 1b
Use the commands cd, ls and pwd to explore the file system.
(Remember, if you get lost, type cd by itself to return to your home-directory)

1.6 MORE ABOUT HOME DIRECTORIES AND PATHNAMES

Understanding pathnames
First type cd to get back to your home-directory, then type
% ls unixstuff
to list the conents of your unixstuff directory.
Now type
% ls backups
You will get a message like this -
backups: No such file or directory

The reason is, backups 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 either cd to the correct directory, or specify its full pathname. To list the contents of your backups directory, you must type

% ls unixstuff/backups

~ (your home directory)

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

Command	Meaning
Ls	list files and directories
ls –a	list all files and directories
Mkdir	make a directory
cd directory	change to named directory
Cd	change to home-directory
cd ~	change to home-directory
cd ..	change to parent directory
Pwd	display the path of the current directory

2.1 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, cd to your unixstuff directory.

% cd ~/unixstuff

Then at the UNIX prompt, type,

% cp /vol/examples/tutorial/science.txt .

Note: Don't forget the dot . at the end. Remember, in UNIX, the dot means the current directory.

The above command means copy the file science.txt to the current directory, keeping the name the same.

(Note: The directory /vol/examples/tutorial/ is an area to which everyone in the school has read and copy access. If you are from outside the University, you can grab a copy of the file here. Use 'File/Save As..' from the menu bar to save it into your unixstuff directory.)

Exercise 2a

Create a backup of your science.txt file by copying it to a file called science.bak

2.2 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.

We are now going to move the file science.bak to your backup directory.

First, change directories to your unixstuff directory (can you remember how?). Then, inside the unixstuff directory, type

% mv science.bak backups/.

Type ls and ls backups to see if it has worked.

2.3 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 create a copy of the science.txt file then delete it.
Inside your unixstuff directory, type
% cp science.txt tempfile.txt
% ls
% rm tempfile.txt
% ls
You can use the rmdir command to remove a directory (make sure it is empty first). Try to remove the backups directory. You will not be able to since UNIX will not let you remove a non-empty directory.
Exercise 2b
Create a directory called tempstuff using mkdir , then remove it using the rmdir command.

2.4 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 science.txt
As you can see, the file is longer than than the size of the window, so it scrolls past making it unreadable.

LESS

The command less writes the contents of a file onto the screen a page at a time. Type
% less science.txt

Press the [space-bar] if you want to see another page, and type [q] if you want to quit reading. As you can see, less 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 science.txt

Then type

% head -5 science.txt

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 science.txt

Q. How can you view the last 15 lines of the file?

2.5 SEARCHING THE CONTENTS OF A FILE

SIMPLE SEARCHING USING LESS

Using less, you can search though a text file for a keyword (pattern). For example, to search through science.txt for the word 'science', type

% less science.txt

then, still in less, type a forward slash [/] followed by the word to search

/science

As you can see, less finds and highlights the keyword. Type [n] to search for the next occurrence of the word.

GREP

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

% grep science science.txt

As you can see, grep has printed out each line containg the word science.

Or has it ????

Try typing

% grep Science science.txt

The grep command is case sensitive; it distinguishes between Science and science.

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

% grep -i science science.txt

To search for a phrase or pattern, you must enclose it in single quotes (the apostrophe symbol). For example to search for spinning top, type
% grep -i 'spinning top' science.txt

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, the number of lines without the words science or Science is
% grep -ivc science science.txt

WC (WORD COUNT)

A handy little utility is the wc command, short for word count. To do a word count on science.txt, type
% wc -w science.txt
To find out how many lines the file has, type
% wc -l science.txt

Summary

COMMAND	MEANING
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
less 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

3.1 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 specifing 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 specifing 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 both the input and the output of commands.

3.2 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 {this means press [Ctrl] and [d] to stop}
What happens is the cat command reads the standard input (the keyboard) and the > redirects the output, which normally goes to the screen, into a file called list1
To read the contents of the file, type
% cat list1
Exercise 3a
Using the above method, create another file called list2 containing the following fruit: orange, plum, mango, grapefruit. Read the contents of list2

3.2.1 APPENDING TO A FILE

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
orange
^D (Control D to stop)
To read the contents of the file, type
% cat list1
You should now have two files. One contains six fruit, the other contains four fruit. 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 outputing the text to the file biglist
To read the contents of the new file, type
% cat biglist

3.3 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 animals. Press [Return] after each one.
dog
cat
bird
ape
^D (control d to stop)
The output will be
ape
bird
cat
dog

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

3.4 PIPES

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.txt

% sort < names.txt

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 the vertical bar |

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 3b

Using pipes, display all lines of list1 and list2 containing the letter 'p', and sort the result.

Answer available here

Summary

COMMAND	MEANING
command > file	redirect standard output to a file
command >> file	append standard output to a file
command < file	redirect standard input from a file
command1 | command2	pipe the output of command1 to the input of command2

cat file1 file2 > file0	concatenate file1 and file2 to file0
sort	sort data
who	list users currently logged in

4.1 WILDCARDS

THE * WILDCARD

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 ? wildcard

The character ? will match exactly one character.

So ?ouse will match files like house and mouse, but not grouse.

Try typing

% ls ?list

4.2 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 / * & % , should be avoided. Also, avoid using spaces within names. The safest way to name a file is to use only alphanumeric characters, that is, letters and numbers, together with _ (underscore) and . (dot).

GOOD FILENAMES	BAD FILENAMES
project.txt	project
my_big_program.c	my big program.c
fred_dave.doc	fred & dave.doc

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 C code may be named with the ending .c, for example, prog1.c . Then in order to list all files containing C code in your home directory, you need only type ls *.c in that directory.

4.3 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

COMMAND	MEANING
*	match any number of characters
?	match one character
man command	read the online manual page for a command
whatis command	brief description of a command
apropos keyword	match commands with keyword in their man pages

5.1 FILE SYSTEM SECURITY (ACCESS RIGHTS)

In your unixstuff directory, type
% ls -l (l for long listing!)
You will see that you now get lots of details about the contents of your directory, similar to the example below.

Each file (and directory) has associated access rights, which may be found by typing ls -l. Also, ls -lg gives additional information as to which group owns the file (beng95 in the following example):
-rwxrw-r-- 1 ee51ab beng95 2450 Sept29 11:52 file1
In the left-hand column is a 10 symbol string 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 string, and indicates a directory: otherwise - will be the starting symbol of the string.
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 user that owns 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 all others.

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

5.2 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 (and access directory)
+	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
Exercise 5a
Try changing access permissions on the file science.txt and on the directory backups
Use ls -l to check that the permissions have changed.

5.3 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.

RUNNING BACKGROUND PROCESSES

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 do 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 be 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
At the prompt, type
% sleep 1000
You can suspend the process running in the foreground by typing ^Z, i.e.hold down the [Ctrl] key and type [z]. Then to put it in the background, type
% bg
Note: do not background programs that require user interaction e.g. vi

5.4 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 1000
[2] Running netscape
[3] Running matlab
To restart (foreground) a suspended processes, type
% fg %jobnumber
For example, to restart sleep 1000, type
% fg %1
Typing fg with no job number foregrounds the last suspended process.

5.5 KILLING A PROCESS

KILL (TERMINATE OR SIGNAL 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

^C

To kill a suspended or background process, type

% kill %jobnumber

For example, run

% sleep 100 &

% jobs

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.

PS (PROCESS STATUS)

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

% sleep 1000 &

% ps

PID TT S TIME COMMAND

20077 pts/5 S 0:05 sleep 1000

21563 pts/5 T 0:00 netscape

21873 pts/5 S 0:25 nedit

To kill off the process sleep 1000, 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

COMMAND	MEANING
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

GZIP

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

% ls -l science.txt

and note the size of the file using ls -l . Then to compress science.txt, type

% gzip science.txt

This will compress the file and place it in a file called science.txt.gz

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

To expand the file, use the gunzip command.

% gunzip science.txt.gz

ZCAT

zcat will read gzipped files without needing to uncompress them first.

% zcat science.txt.gz

If the text scrolls too fast for you, pipe the output though less .

% zcat science.txt.gz | less

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 *

DIFF

This command compares the contents of two files and displays the differences. Suppose you have a file called file1 and you edit some part of it and save it as file2. To see the differences type

% diff file1 file2

Lines beginning with a < denotes file1, while lines beginning with a > denotes file2.

FIND

This searches through the directories for files and directories with a given name, date, size, or any other attribute you care to specify. It is a simple command but with many options - you can read the manual by typing man find.

To search for all fies with the extention .txt, starting at the current directory (.) and working through all sub-directories, then printing the name of the file to the screen, type

% find . -name "*.txt" -print

To find files over 1Mb in size, and display the result as a long listing, type

% find . -size +1M -ls

HISTORY

The C shell keeps an ordered list of all the commands that you have entered. Each command is given a number according to the order it was entered.

% history (show command history list)

If you are using the C shell, you can use the exclamation character (!) to recall commands easily.

% !! (recall last command)

% !-3 (recall third most recent command)

% !5 (recall 5th command in list)

% !grep (recall last command starting with grep)

You can increase the size of the history buffer by typing

% set history=100