History of UNIX

UNIX development was started in 1969 at Bell Laboratories in New Jersey. Bell Laboratories was (1964–1968) involved on the development of a multi-user,time-sharing operating system called Multics (Multiplexed Information and Computing System). Multics was a failure. In early 1969, Bell Labs withdrew from the Multics project.

Bell Labs researchers who had worked on Multics (Ken Thompson, Dennis Ritchie, Douglas McIlroy, Joseph Ossanna, and others) still wanted to develop an operating system for their own and Bell Labs' programming, job control, and resource usage needs. When Multics was withdrawn Ken Thompson and Dennis Ritchie needed to rewrite an operating system in order to play space travel on another smaller machine (a DEC PDP-7 [Programmed Data Processor 4K memory for user programs). The result was a system called UNICS (UNiplexed Information and Computing Service) which was an 'emasculated Multics'.

The first version of Unix was written in the low-level PDP-7 assembler language. Later, a language called TMG was developed for the PDP-7 by R. M. McClure. Using TMG to develop a FORTRAN compiler, Ken Thompson instead ended up developing a compiler for a new high-level language he called B, based on the earlier BCPL language developed by Martin Richard. When the PDP-11 computer arrived at Bell Labs, Dennis Ritchie built on B to create a new language called C. Unix components were later rewritten in C, and finally with the kernel itself in 1973.

Unix V6, released in 1975 became very popular. Unix V6 was free and was distributed with its source code.

In 1983, AT&T released Unix System V which was a commercial version.

Meanwhile, the University of California at Berkeley started the development of its own version of Unix. Berkeley was also involved in the inclusion of Transmission Control Protocol/Internet Protocol (TCP/IP) networking protocol.

The following were the major mile stones in UNIX history early 1980's

• AT&T was developing its System V Unix.

• Berkeley took initiative on its own Unix BSD (Berkeley Software Distribution) Unix.

• Sun Microsystems developed its own BSD-based Unix called SunOS and later was renamed to Sun Solaris.

• Microsoft and the Santa Cruz operation (SCO) were involved in another version of UNIX called XENIX.

• Hewlett-Packard developed HP-UX for its workstations.

• DEC released ULTRIX.

• In 1986, IBM developed AIX (Advanced Interactive eXecutive).

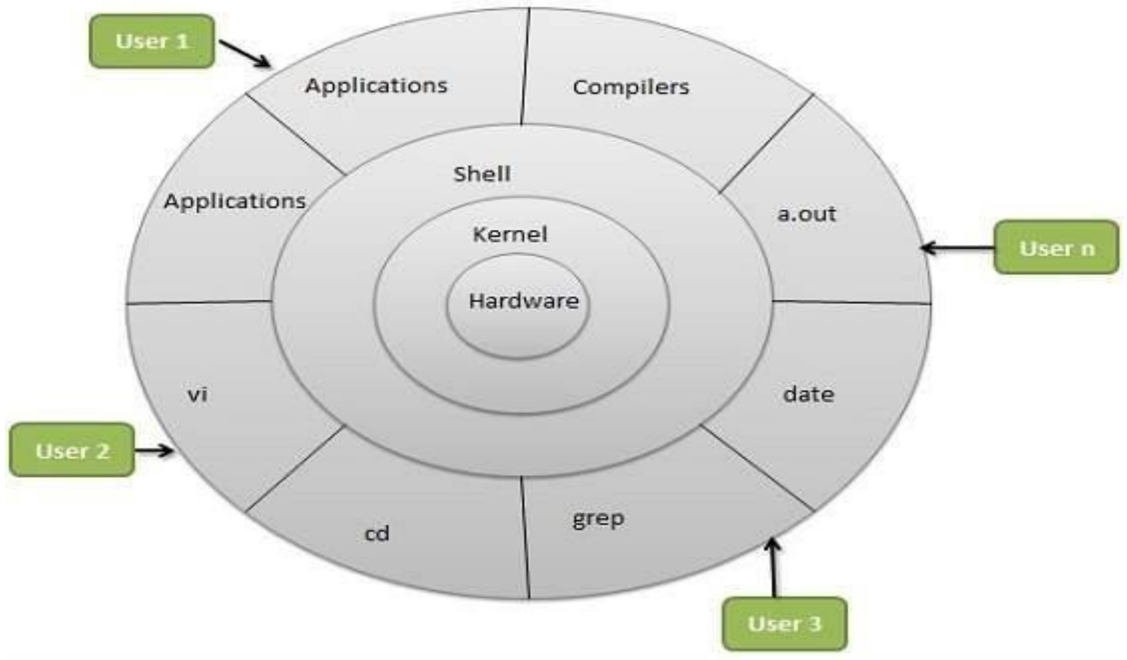
Salient Features of UNIX

Following are some of the important features of Unix Operating System.

- **Portable** − Portability means software can work on different types of hardware in same way. Unix kernel and application programs supports their installation on any kind of hardware platform.
- **Open Source** − Unix source code is freely available and it is community based development project. Multiple teams work in collaboration to enhance the capability of Unix operating system and it is continuously evolving.
- **Multi-User Capability: -** Multi-user operational system permits several users to use the same computer to carry out their job. Several terminals are connected to a single powerful computer and each user of the terminal can be a programmer, program access files and prints document at the same time. Need for the multi-user environment arises when several programmers work on developing module of the same software. Multi-user environment ensures complete coordination and compatibility and saves a considerable amount of time by allowing several users on a set of information at a time. Buying single multi-user computer is far more economical and efficient than buying several single users computers.
- **Multiprogramming** − Unix is a multiprogramming system means multiple applications can run at same time.
- **Multitasking Capability: -** This capability allows the system to perform several tasks simultaneously. For instance, UNIX can print one document, edit another and sort a list of files at the same time. Multiple tasks can be carried out by placing other tasks in the background while you work on one task at a time. The current tasks are said to be in the foreground. Normally the tasks that do not require active interaction from the user are placed in the background; the lower will be the system response.
- **Hierarchical File System** − Unix provides a standard file structure in which system files/ user files are arranged.
- **Shell** − Unix provides a special interpreter program which can be used to execute commands of the operating system. It can be used to do various types of operations, call application programs. etc.
- **Security** − Unix provides user security using authentication features like password protection/ controlled access to specific files/ encryption of data.
- **Communication: -** UNIX supports two major types of communication:
  1. Communication between different terminals connected to the same computer.
  2. Communication between users of the one computer at the specific location to the users of another type and size of a computer located elsewhere. The two computers may be located in different offices or different countries or continents. These types of communication is achieved by a mail system based on wide area and may be connected through telephone lines or satellite.

UNIX Architecture

The following illustration shows the architecture of a UNIX system −



The architecture of a UNIX System consists of the following layers –

Kernel

The kernel of UNIX is the hub of the operating system: it allocates time and memory to programs and handles the file store 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 my file (which has the effect of removing the file **my file**). The shell searches the file store for the file containing the program rm, and then requests the kernel, through system calls, to execute the program rm on my file. When the process rm my file has finished running, the shell then returns the UNIX prompt % to the user, indicating that it is waiting for further commands.

Amongst the functions performed by the kernel are:

- Managing the machine's memory and allocating it to each process.
- Scheduling the work done by the CPU so that the work of each user is carried out as efficiently as is possible. Organising the transfer of data from one part of the machine to another.
- Accepting instructions from the shell and carrying them out.
- Enforcing the access permissions that are in force on the file system.

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 (% on our systems). The user can customise his/her own shell, and users can use different shells on the same machine.
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 or type history for a list of previous commands. You can use any one of these shells if they are available on your system. And you can switch between the different shells once you have found out if they are available.

- ✓ **Bourne shell (sh)**
- ✓ **C shell (csh)**
- ✓ **TC shell (tcsh)**
- ✓ **Korn shell (ksh)**
- ✓ **Bourne Again SHell (bash)**
- ✓ **Features of Shell:**

Shell Features	Description
Command interpreter	Shell interprets the valid user entered commands to the kernel understandable language. Shell should understand which program to be invoked in order to perform that command.
Prompts	Ideally prompts are shown as “$” or “#”
Shell scripts	Uses multiple shell commands logically to create a script i.e. runs more than one commands at a time.
I/O redirection and Pipes	These features in Shell allow connecting programs together and making programs process their inputs from files

Hardware layer − Hardware consists of all peripheral devices (RAM/ HDD/ CPU etc).

Utilities and Application

The final layer of the UNIX OS is the Utilities and Applications layer. This layer includes the commands, word processors, graphic programs and database management programs. Traditionally, these programs were accessed by typing the commands to start the program on the command line. They can still be accessed in this way, but they can now also be accessed through the GUI.

UNIX command format

Syntax of UNIX Commands

UNIX commands can be very simple one word commands or they can take a number of additional arguments (parameters) as part of the command. In general, a UNIX command has the following general form...

Command options(s) filename(s)

Let's look at each of these parts a bit more closely...

- ❖ The command is the name of the utility or program that we are going to execute.
- ❖ The options are passed into the command to modify the way the command works. It is typical for these options to have be a hyphen followed by a single character, such as-**l**. It is also a common convention under Linux to have options that are in the form of 2 hyphens followed by a word or hyphenated words, such as **--color** or **--pretty-print**.
- ❖ The filename is the last argument for a lot of UNIX commands. It is simply the file or files that you want the command to work on. Take note that not all commands work on files, such as ssh which takes the name of a host as its argument.

Common UNIX Conventions

- ➢ In UNIX the command is almost always entered in all lower case characters.
- ➢ Typically, any options come before filenames.
- ➢ Many times individual options may need a word after them to designate some additional meaning to the command.

UNIX commands are classified into two types

- ➢ Internal Commands - Ex: cd, source, fg
- ➢ External Commands - Ex: ls, cat

Let us look at these in detail

Internal Command:

Internal commands are something which is built into the shell. For the shell built in commands, the execution speed is really high. It is because no process needs to be spawned for executing it. For example, when using the "cd" command, no process is created. The current directory simply gets changed on executing it.

External Command:

External commands are not built into the shell. These are executables present in a separate file. When an external command has to be executed, a new process has to be spawned and the command gets executed. For example, when you execute the "cat" command, which usually is at /usr/bin, the executable /usr/bin/cat gets executed.

How to get the list of Internal commands?

You can get only if you are in bash shell. Bash shell has a command called "help" which will list out all the built-in shell commands.

```
$ help
alias [-p] [name[=value] ... ]
bg [job_spec ...]
bind [-lpvsPVS] [-m keymap] [-f fi break [n] builtin [shell-builtin [arg ...]]  caller [EXPR]
```

```
case WORD in [PATTERN [| PATTERN]. cd [-L|-P] [dir]
command [-pVv] command [arg ...]   compgen [-abcdefgjksuv] [-o option
.....]
```

How to find out whether a command is internal or external?

type command:

```
$ type cd

cd is a shell builtin
$ type cat
cat is /bin/cat
```

For the internal commands, the type command will clearly say its shell built-in, however for the external commands, it gives the path of the command from where it is executed. **Internal vs External?** The question whether should we use an internal command or an external command OR which is better always does not make sense. Because in most of the situations you will end up using the command which does your job which could be either internal or external.

The big difference in internal vs external command is performance. Internal command is much much faster compared to external for the simple reason that no process needs to be spawned for an internal command since it is all built-into the shell. So, as the size of a script gets bigger, using external commands a lot does adds to its performance.

Not always we get a choice to choose an internal over an external command. However, a careful look at our scripting practices, we might find quite a few places where we can avoid external commands.

Example:

Say to add 2 numbers say x & y: Not good:

```
z=`expr $x+$y`
```

Good:

```
let z=x+y
```

let is a shell built-in command, whereas expr is an external command. Using expr will be slower. This might be very negligible when you are using it at an one-off instance. Using it in a place say on every record of a file containing million records does give a different dimension to it.

Directory related commands

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, **sgt**, 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 (lowercase L and lowercase S) 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.

Change directory cd

This command changes your current directory location. By default, your Unix login session begins in your home directory.

To switch to a subdirectory (of the current directory) named myfiles, enter:

```
cd myfiles
```

To switch to a directory named /home/dvader/empire_docs, enter:

```
cd /home/dvader/empire_docs
```

To move to the parent directory of the current directory, enter:

```
cd ..
```

To move to the root directory, enter:

```
cd /
```

To return to your home directory, enter:

```
cd
```

Make directory mkdir
This command will make a new subdirectory.
To create a subdirectory named mystuff in the current directory, enter:

```
mkdir mystuff
```

To create a subdirectory named morestuff in the existing directory named /tmp, enter:

```
mkdir /tmp/morestuff
```

Note: To make a subdirectory in a particular directory, you must have permission to write to that directory.

Present Working Directory pwd
This command reports the current directory path. Enter the command by itself:

```
pwd
```

Remove Directory rmdir
This command will remove a subdirectory. To remove a subdirectory named oldstuff, enter:

```
rmdir oldstuff
```

Note: The directory you specify for removal must be empty. To clean it out, switch to the directory and use the ls and rm commands to inspect and delete files.

set
This command displays or changes various settings and options associated with your Unix session.
To see the status of all settings, enter the command without options:

```
set
```

If the output scrolls off your screen, combine set with less:

```
set | less
```

The syntax used for changing settings is different for the various kinds of Unix shells; see the man entries for set and the references listed at the end of this document for more information.

File related commands

Display ContentofaFile

cat
Read/scan the man page for **cat** with the command:
man cat
Use this command to display the contents of a file. What happens?
cat *filename*
Now try this command notice the difference. How many lines are in the file?

cat -n *filename*

The **cat** command is more often used for purposes other than just displaying a file. Try these commands to "concatenate" two files into a new, third file:

```
cat file1              - first, show file1
cat file2              - then, show file2
cat file1 file2 > newfile  - now do the actual concatenate
cat newfile                - finally, show the result
```

Counting Words in a File

You can use the **wc** command to get a count of the total number of lines, words, and characters contained in a file. Following is a simple example to see the information about the file created above −

```
$ wc filename
2  19 103 filename
$
```

Here is the detail of all the four columns −

First Column − Represents the total number of lines in the file.

Second Column − Represents the total number of words in the file.

Third Column − Represents the total number of bytes in the file. This is the actual size of the file.

Fourth Column − Represents the file name.

You can give multiple files and get information about those files at a time. Following is simple syntax −

```
$ wc filename1 filename2 filename3
```

Copying Files

To make a copy of a file use the **cp** command. The basic syntax of the command is −

```
$ cp source_file destination_file
```

Following is the example to create a copy of the existing file **filename**.

```
$ cp filename copyfile
```

You will now find one more file **copy file** in your current directory. This file will exactly be the same as the original file **filename**.

Copying and moving file cp

Read/scan the man page for **cp** with the command:

man cp

Copy an existing file in your current directory to another file in the current directory and then list your directory to prove that it was done:

cp file*1 file2*
ls

Copy a file from a different directory to your current directory and then list your directory to prove that it was done:

cp /etc/ifconfig my.config ls

Use the copy command with the "inquire" option. What happens when you try to copy one file to an already existing file?

cp -i *file1 file2*

Use the recursive option to copy an entire subdirectory to a new subdirectory and then list both directories to prove that it worked:

cp -R subdir1 subdir4 ls
subdir1 subdir4

The copy command accepts "wildcard" characters. Try the command below. What did it do? List the subdir1 subdirectory to find out.
cp samp* subdir1

ls subdir1
If you are copying a file from another location to the current directory and want its name to remain the same, you can use the shorthand "." to indicate the current directory. Try this for example and note what happens:

cp /etc/ifconfig . ls

Renaming a file mv
Read/scan the man page for **mv** with the command:

man mv

The **mv** command can be used for renaming files. Try this command and then list your files to prove that the command worked:
mv my.config new.config ls

mv can be used to rename directories also. Try this command and then list your files to prove that the command worked:
mv subdir4 dir4 ls

The **mv** command is also used for moving files. Use the command below to move new. sample into a new location, then list your files to prove that the command worked:

mv new. Sample dir4 ls
ls dir4

Like many other UNIX commands, **mv** recognizes wildcard characters. Try the command below and then list your files to prove that the command worked:

mv *env* dir4 ls
ls dir4

more
Read/scan the man page for **more** with the command:

man more
Use the more command to read a file:

more ***filename***
As you are reading, notice the "More" prompt at the bottom of the page. Try pressing the return key - what happens?

Try pressing the space bar once - what happens? Type the letter **b** - what happens?
Use the search forward feature to find a word say, "MARCH" by entering the command:

/MARCH
Now use the search backward feature to find a string of words say, "black hole" by entering the command:
?black hole

more will continue until the end of the file is reached or until you type **q** for quit. Try typing **q** to quit.

head and tail Read/scan the man page for head **with the command:**
man head
Display the top of a file with the command below. How many lines do you count?
head ***filename***
Now try this command and note how many lines are displayed this time?
head -5 ***filename***
Read/scan the man page for **tail** with the command:
man tail
Display the same file with the tail command. How is it different and how many lines do you count?
tail ***filename*** Now try this command and note how many lines are displayed this time?

tail -5 ***filename***

Remove a file rm
Read/scan the man page for **rm** with the command:
man rm

Use the **rm** command to delete a file. List your directory after the command completes.
rm new.config ls

cd to another subdirectory say, **subdir2**. List the directory to view its contents. Then use the "*" wildcard to remove all of the files. NOTE: using **rm** in this manner can be dangerous! If you are in the wrong directory, you'll remove files you didn't mean to remove. You may want to use the **-i** option to protect yourself from accidents.

cd subdir2
ls
rm -i *
ls

Get out of the **subdir2** subdirectory by using the command **cd ..** Now try to use **rm** to remove a directory. What happens

s**cd ..**

rm subdir3
This time, include the **-r** option when you try to remove a directory. For example, try removing some subdirectory **subdir3**. What happens?

rm -r subdir3 ls

file
Read/scan the man page for **file** with the command:
man file
Use the **file** command to determine a file's type:
file file1
Now try it with a directory say, **dir4**:
file dir4
Finally, try it with a wildcard character;
file *

find
Read/scan the man page for **find** with the command:
man find
Use the find command to find the file new.config.
find . -name new.config -print
Now use the find command to find all files with "file" as part of their name. Don't forget to put the wildcard specification in quotes - it won't work otherwise:
find . -name 'file*' -print

Try to find only directories with "file" as part of their name. Are there any?

find . -name 'file*' -type d -print diff and

sdiff
Read/scan the man page for **diff** with the command:
man diff
Use the **diff** command to determine the differences between two files:
diff names1 names2
Use the **diff** command to determine the differences between two directories.

diff subdir1 dir4

Try using **sdiff** to display the differences between names1 and names2:

sdiff names1 names2

Now try sdiff specifying that the screen width is 80 characters...not the default 130:

sdiff -w 80 names1 names2

sort

Read/scan the man page for **sort** with the command:

man sort

Use the **cat** command to look at an unsorted list of names. Notice that there are multiple columns of information and that the list is not alphabetically sorted.

cat name.list

Use the **sort** command to perform a basic sort of the list:

sort name.list

Do the same sort, but send the output to a file. Then use the **cat** utility to view the file:

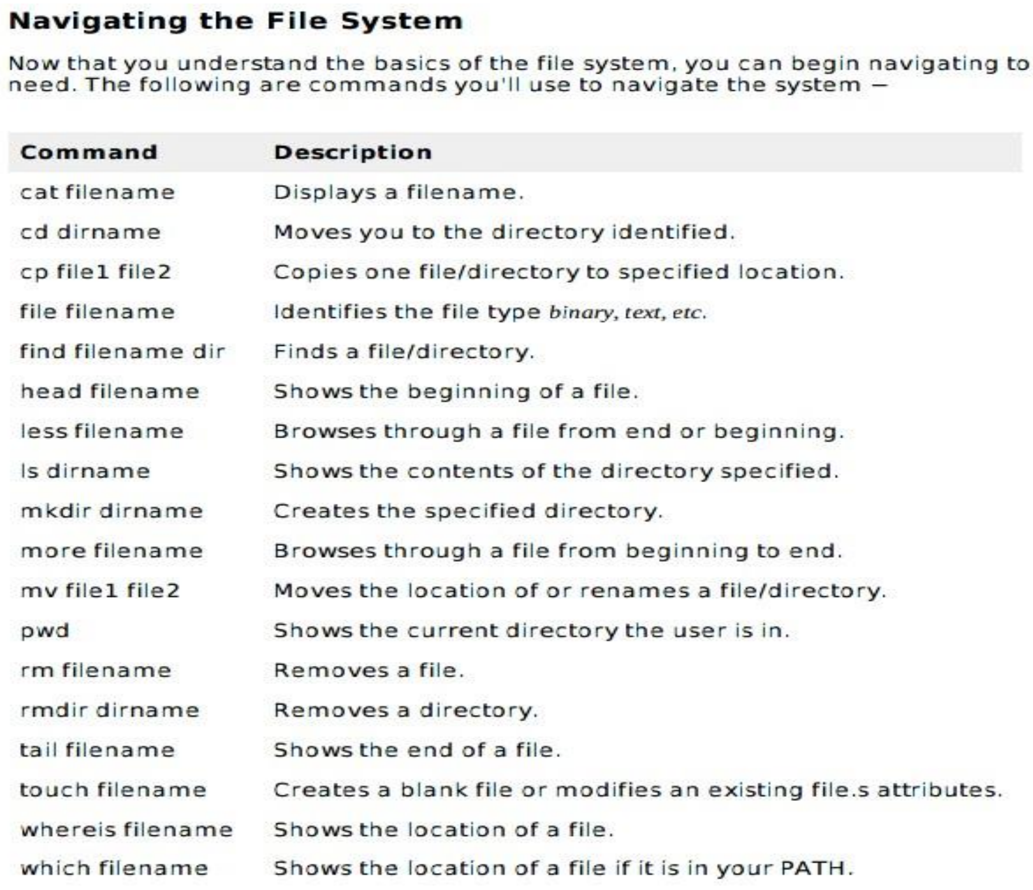
sort name.list > sorted.list ; cat sorted.list

Sort the list by first names. This requires skipping over the first field, which is the last name:

sort +1 name.list

Sort the list by department. This requires skipping over the first four fields and using the **- b** option to ignore blank characters:

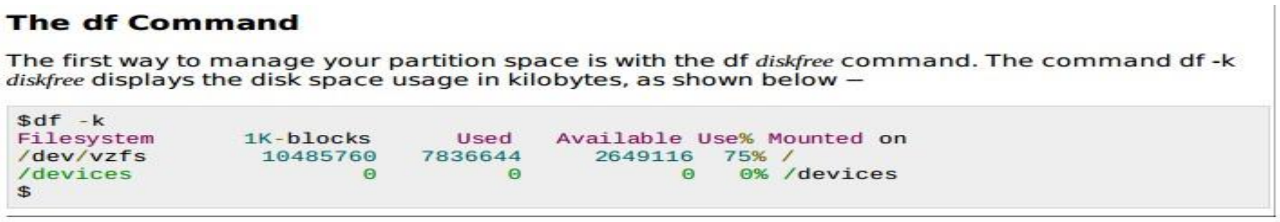
sort -b +4 name.list



Navigating the File System

Now that you understand the basics of the file system, you can begin navigating to the files you need. The following are commands you'll use to navigate the system —

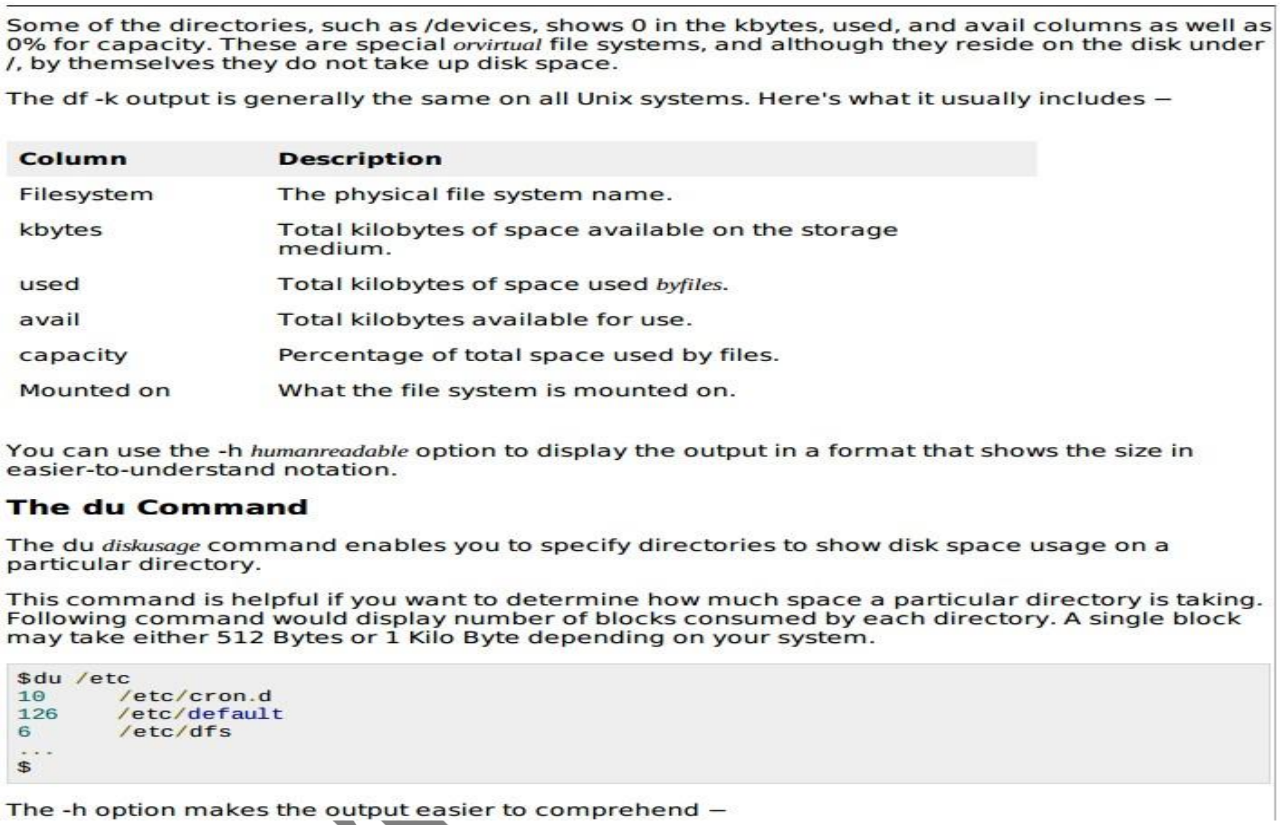
Command	Description
cat filename	Displays a filename.
cd dirname	Moves you to the directory identified.
cp file1 file2	Copies one file/directory to specified location.
file filename	Identifies the file type *binary, text, etc.*
find filename dir	Finds a file/directory.
head filename	Shows the beginning of a file.
less filename	Browses through a file from end or beginning.
ls dirname	Shows the contents of the directory specified.
mkdir dirname	Creates the specified directory.
more filename	Browses through a file from beginning to end.
mv file1 file2	Moves the location of or renames a file/directory.
pwd	Shows the current directory the user is in.
rm filename	Removes a file.
rmdir dirname	Removes a directory.
tail filename	Shows the end of a file.
touch filename	Creates a blank file or modifies an existing file.s attributes.
whereis filename	Shows the location of a file.
which filename	Shows the location of a file if it is in your PATH.



The df Command

The first way to manage your partition space is with the df *diskfree* command. The command df -k *diskfree* displays the disk space usage in kilobytes, as shown below −

```
$df -k
Filesystem      1K-blocks      Used   Available Use% Mounted on
/dev/vzfs        10485760   7836644     2649116  75% /
/devices                0         0           0   0% /devices
$
```



Some of the directories, such as /devices, shows 0 in the kbytes, used, and avail columns as well as 0% for capacity. These are special *orvirtual* file systems, and although they reside on the disk under /, by themselves they do not take up disk space.

The df -k output is generally the same on all Unix systems. Here's what it usually includes −

Column	Description
Filesystem	The physical file system name.
kbytes	Total kilobytes of space available on the storage medium.
used	Total kilobytes of space used *byfiles*.
avail	Total kilobytes available for use.
capacity	Percentage of total space used by files.
Mounted on	What the file system is mounted on.

You can use the -h *humanreadable* option to display the output in a format that shows the size in easier-to-understand notation.

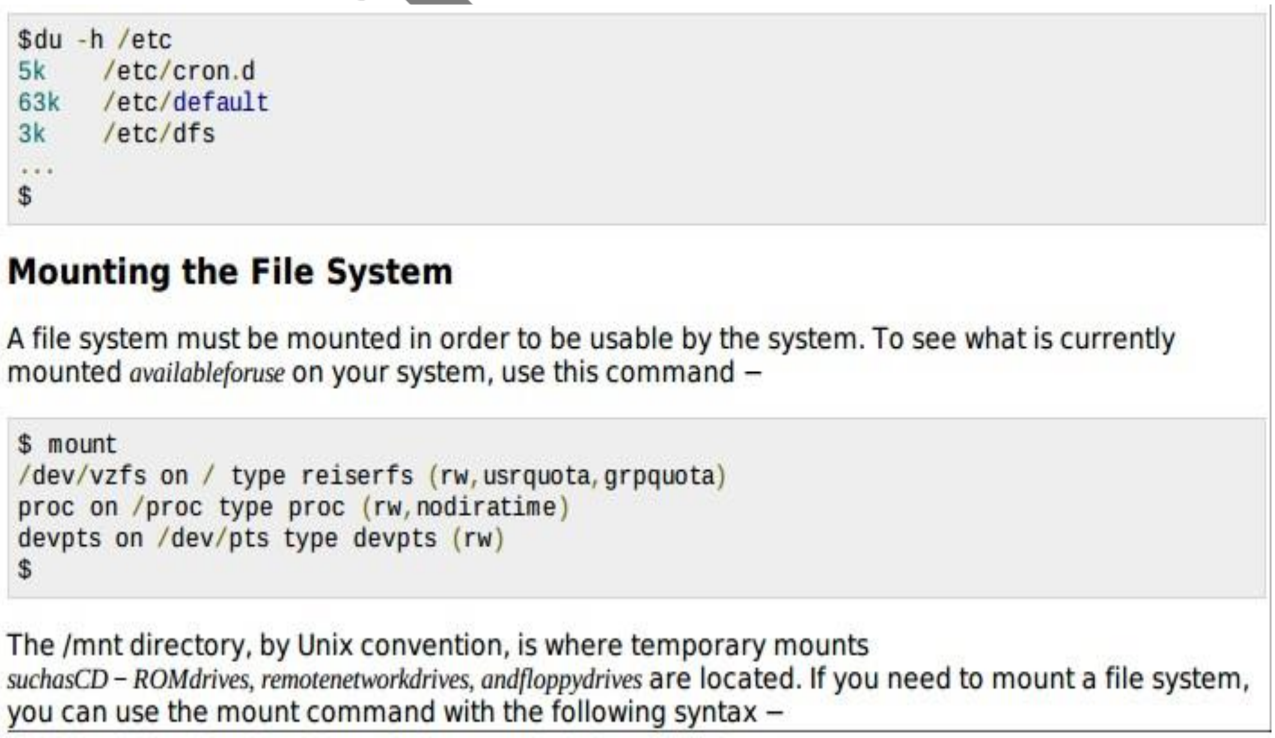
The du Command

The du *diskusage* command enables you to specify directories to show disk space usage on a particular directory.

This command is helpful if you want to determine how much space a particular directory is taking. Following command would display number of blocks consumed by each directory. A single block may take either 512 Bytes or 1 Kilo Byte depending on your system.

```
$du /etc
10      /etc/cron.d
126     /etc/default
6       /etc/dfs
...
$
```

The -h option makes the output easier to comprehend −



```
$du -h /etc
5k    /etc/cron.d
63k   /etc/default
3k    /etc/dfs
...
$
```

Mounting the File System

A file system must be mounted in order to be usable by the system. To see what is currently mounted *availableforuse* on your system, use this command −

```
$ mount
/dev/vzfs on / type reiserfs (rw,usrquota,grpquota)
proc on /proc type proc (rw,nodiratime)
devpts on /dev/pts type devpts (rw)
$
```

The /mnt directory, by Unix convention, is where temporary mounts *suchasCD − ROMdrives, remotenetworkdrives, andfloppydrives* are located. If you need to mount a file system, you can use the mount command with the following syntax −

```
mount -t file_system_type device_to_mount directory_to_mount_to
```

For example, if you want to mount a CD-ROM to the directory /mnt/cdrom, for example, you can type –

```
$ mount -t iso9660 /dev/cdrom /mnt/cdrom
```

This assumes that your CD-ROM device is called /dev/cdrom and that you want to mount it to /mnt/cdrom. Refer to the mount man page for more specific information or type mount -h at the command line for help information.

After mounting, you can use the cd command to navigate the newly available file system through the mountpoint you just made.

Unmounting the File System

To unmount *remove* the file system from your system, use the **umount** command by identifying the mountpoint or device

For example, to unmount cdrom, use the following command –

```
$ umount /dev/cdrom
```

The mount command enables you to access your file systems, but on most modern Unix systems, the automount function makes this process invisible to the user and requires no intervention.

User and Group Quotas

User and group quotas provide the mechanisms by which the amount of space used by a single user or all users within a specific group can be limited to a value defined by the administrator.

Quotas operate around two limits that allow the user to take some action if the amount of space or number of disk blocks start to exceed the administrator defined limits –

- **Soft Limit** – If the user exceeds the limit defined, there is a grace period that allows the user to free up some space.
- **Hard Limit** – When the hard limit is reached, regardless of the grace period, no further files or blocks can be allocated.

There are a number of commands to administer quotas –

Command	Description
quota	Displays disk usage and limits for a user of group.
edquota	This is a quota editor. Users or Groups quota can be edited using this command.
quotacheck	Scan a filesystem for disk usage, create, check and repair quota files
setquota	This is also a command line quota editor.
quotaon	This announces to the system that disk quotas should be enabled on one or more filesystems.

INFORMATION COMMANDS

history	Lists the commands typed during the session. Options: -r displays the list in reverse.
hostname	Displays the computer's or server's name on the terminal.
who	Displays who is on the system. who am
Displays the invoking user.	i
wc	Counts and displays the number of lines, words and characters of a file. Usage: wc [options] Options: -c count character only. -l count lines only. -w count words only.
whatis	Displays the command description.
id	Displays the user id and the group id of the invoking user.
tty	Displays users terminal name
Date	Displays the system date & time
Cal	Display the calendar

FILE SYSTEM IN UNIX (UNIX FILE SYSTEM)

A file system is the methods and data structures that an operating system uses to keep track of files on a disk or partition; that is, the way the files are organized on the disk. The word is also used to refer to a partition or disk that is used to store the files or the type of the file system.

What is a file in Unix?

Everything in **Unix** is a **file** or a process. In **Unix** a **file** is just a destination for or a source of a stream of data. Thus a printer, for example, is a **file** and so is the screen. A process is a program that is currently running.

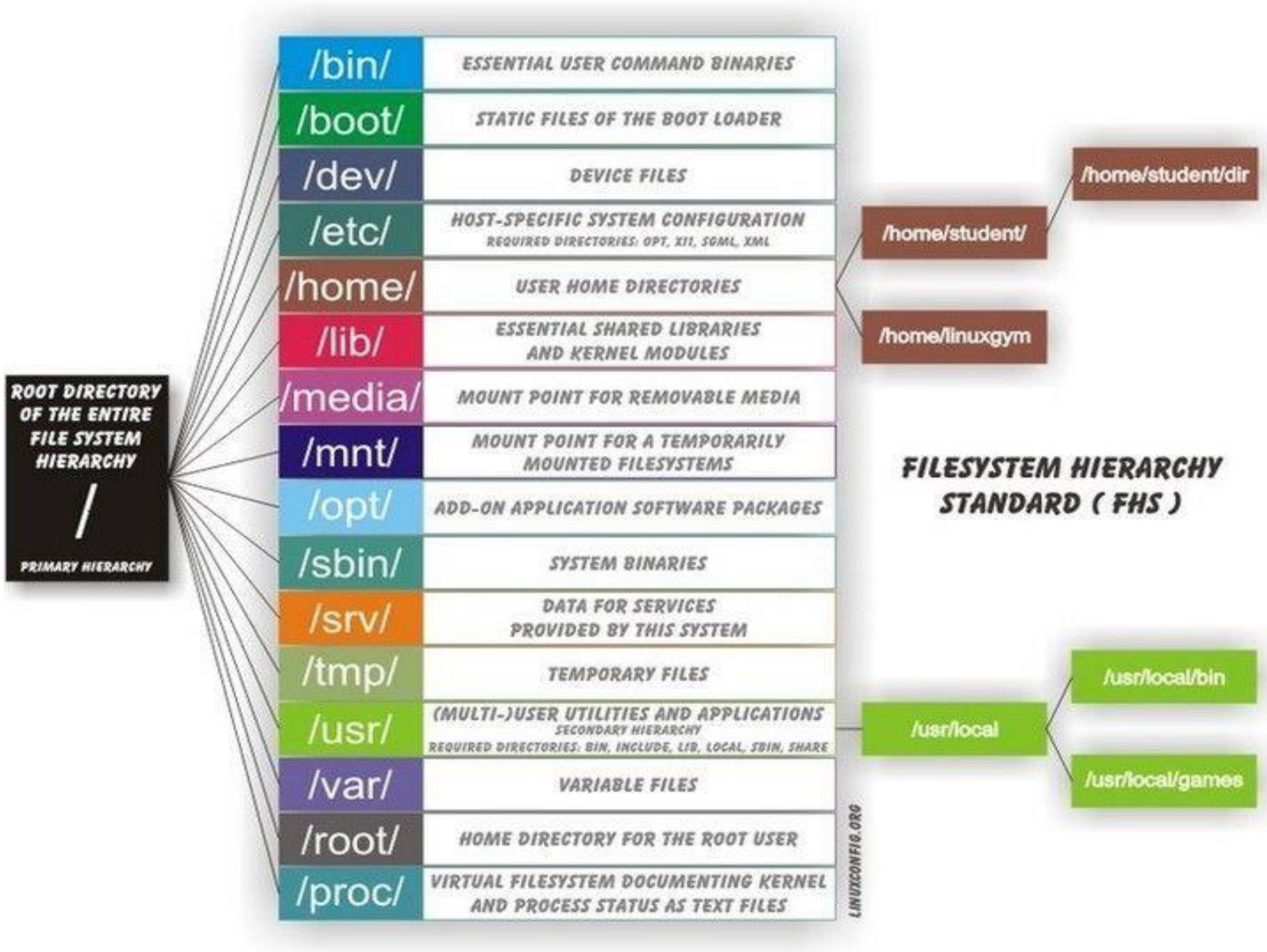
What is the use of directory in Unix?

Like that legacy operating system, the files on a Linux system are arranged in what is called a **hierarchical directory structure**. This means that they are organized in a tree-like pattern of directories (called folders in other systems), which may contain files and other directories.

What is the meaning of mounting in Unix?

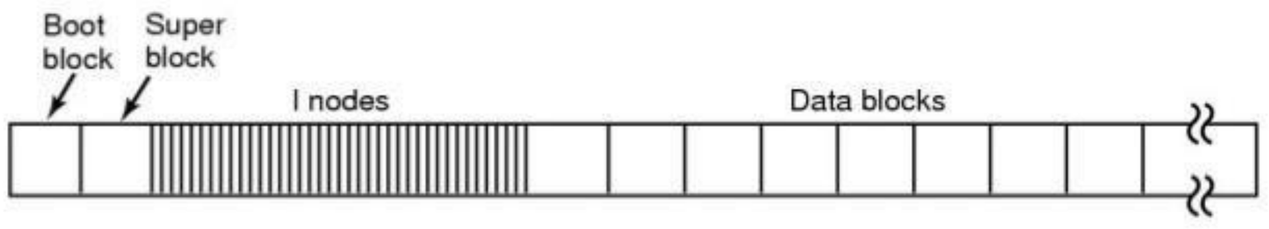
The **mount** command mounts a storage device or file system, making it accessible and attaching it to an existing directory structure. The umount command "unmounts" a mounted file system, informing the system to complete any pending read or write operations, and safely detaching it.

File system structure or Directory structure


ROOT DIRECTORY OF THE ENTIRE FILE SYSTEM HIERARCHY
/
PRIMARY HIERARCHY
/bin/
ESSENTIAL USER COMMAND BINARIES
/boot/
STATIC FILES OF THE BOOT LOADER
/dev/
DEVICE FILES
/etc/
HOST-SPECIFIC SYSTEM CONFIGURATION
REQUIRED DIRECTORIES: OPT, X11, SGML, XML
/home/
USER HOME DIRECTORIES
/lib/
ESSENTIAL SHARED LIBRARIES AND KERNEL MODULES
/media/
MOUNT POINT FOR REMOVABLE MEDIA
/mnt/
MOUNT POINT FOR A TEMPORARILY MOUNTED FILESYSTEMS
/opt/
ADD-ON APPLICATION SOFTWARE PACKAGES
/sbin/
SYSTEM BINARIES
/srv/
DATA FOR SERVICES PROVIDED BY THIS SYSTEM
/tmp/
TEMPORARY FILES
/usr/
(MULTI-)USER UTILITIES AND APPLICATIONS
SECONDARY HIERARCHY
REQUIRED DIRECTORIES: BIN, INCLUDE, LIB, LOCAL, SBIN, SHARE
/var/
VARIABLE FILES
/root/
HOME DIRECTORY FOR THE ROOT USER
/proc/
VIRTUAL FILESYSTEM DOCUMENTING KERNEL AND PROCESS STATUS AS TEXT FILES
/home/student/
/home/student/dir
/home/linuxgym
FILESYSTEM HIERARCHY STANDARD (FHS)
/usr/local
/usr/local/bin
/usr/local/games
LINUXCONFIG.ORG

A UFS volume is composed of the following parts:

- A few blocks at the beginning of the partition reserved for boot blocks (which must be initialized separately from the file system)
- A superblock, containing a magic number identifying this as a UFS file system, and some other vital numbers describing this file system's geometry and statistics and behavioral tuning parameters
- A collection of cylinder groups. Each cylinder group has the following components:
- A backup copy of the superblock
- A cylinder group header, with statistics, free lists, etc., about this cylinder group, similar to those in the superblock
- A number of inodes, each containing file attributes
- A number of data blocks



DISK LAYOUT IN CLASSICAL UNIX SYSTEM

Boot Block: located in the first few sectors of a file system. The boot block contains the initial bootstrap program used to load the operating system. Typically, the first sector contains a bootstrap program that reads in a larger bootstrap program from the next few sectors, and so forth.

Super Block: The superblock is a unique data structure in a file system (though multiple copies exist to guard against corruption). The superblock holds metadata about the file system, like which inode is the top-level directory and the type of file system used.

Each file system has one super block (+ duplicate super block) it contains info about

1. type of file system (ext2, ext3...)
2. the block size
3. pointers to a list of free blocks
4. the inode number of the root directory
5. magic number

inode list: An inode (short for "index node") is a bunch of attributes about a file that Unix stores. There is one inode for each file (though with some file systems, Unix has to create its own inodes because the information is spread around the file system). The inode stores information like who owns the file, how big the file is, and who is allowed to open the file. Each inode also contains a number unique to the file system partition; it's like a serial number for the file described by that inode.

This structure consists of info of file about

1. file ownership indication
2. file type (e.g., regular, directory, special device, pipes, etc.)
3. file access permissions. May have setuid (sticky) bit set.
4. time of last access, and modification
5. number of links (aliases) to the file
6. pointers to the data blocks for the file
7. size of the file in bytes (for regular files), major and minor device numbers for special devices.

Inodes include pointers to the data blocks. Each inode contains 15 pointers:

- the first 12 pointers point directly to data blocks
- the 13th pointer points to an indirect block, a block containing pointers to data blocks
- the 14th pointer points to a doubly-indirect block, a block containing 128 addresses of singly indirect blocks
- the 15th pointer points to a triply indirect block (which contains pointers to doubly indirect blocks, etc.

inode

inode a data structure that keeps track of all the information about a file.

You store your information in a file, and the operating system stores the information about a file in an inode (sometimes called as an inode number).

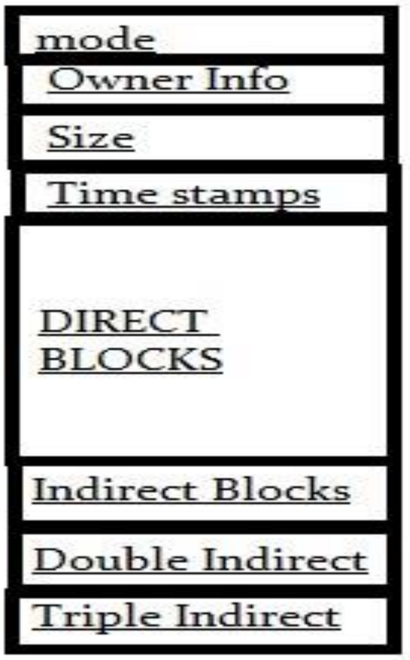
Information about files(data) are sometimes called metadata. So you can even say it in another way, ***"An inode is metadata of the data."***

henever a user or a program needs access to a file, the operating system first searches for the exact and unique inode (inode number), in a table called as an inode table. In fact the program or the user who needs access to a file, reaches the file with the help of the inode number found from the inode table.

To reach a particular file with its "name" needs an inode number corresponding to that file. But to reach an inode number you don't require the file name. In fact, with the inode number you can get the data.

Inode Structure of a File

Now let's see how the structure of an inode of a file look like.



Mode:

This keeps information about two things, one is the permission information, the other is the type of inode, for example an inode can be of a file, directory or a block device etc.

Owner Info: Access details like owner of the file, group of the file etc.

Size: This location stores the size of the file in terms of bytes.

Time Stamps: it stores the inode creation time, modification time, etc.

Now comes the important thing to understand about how a file is saved in a partition with the help of an inode.

Block Size: Whenever a partition is formatted with a file system.It normally gets formatted with a default block size. Now block size is the size of chunks in which data
will be spread. So if the block size is 4K, then for a file of 15K it will take 4 blocks(because
4K*4 16), and technically speaking you waste 1 K.

Direct Block Pointers:

In an ext2 file system an inode consists of only 15 block pointers. The first 12 block pointers are called as Direct Block pointers. Which means that these pointers point to the address of the blocks containing the data of the file. 12 Block pointers can point to 12 data blocks. So in total the Direct Block pointers can address only 48K(12 * 4K) of data. Which means if the file is only of 48K or below in size, then inode itself can address all the blocks
containing the data of the file.

Now What if the file size is above 48K?

Indirect Block Pointers:

whenever the size of the data goes above 48k(by considering the block size as 4k), the 13th pointer in the inode will point to the very next block after the data(adjacent block after 48k of data), which inturn will point to the next block address where data is to be copied.

Now as we have took our block size as 4K, the indirect block pointer, can point to 1024 blocks containing data(by taking the size of a block pointer as 4bytes, one 4K block can point to 1024 blocks because 4 bytes * 1024 = 4K).

which means an indirect block pointer can address, upto 4MB of data(4bytes of block pointer in 4K block, can point and address 1024 number of 4K blocks which makes the data size of
4M)

Double indirect Block Pointers:

Now if the size of the file is above 4MB + 48K then the inode will start using Double Indirect Block Pointers, to address data blocks. Double Indirect Block pointer in an inode will point to the block that comes just after 4M + 48K data, which intern will point to the blocks where the data is stored.

Double Indirect block pointer also is inside a 4K block as every blocks are 4K, Now block pointers are 4 bytes in size, as mentioned previously, so Double indirect block pointer can address 1024 Indirect Block pointers (which means 1024 * 4M =4G). So with the help of a double indirect Block Pointer the size of the data can go upto 4G.

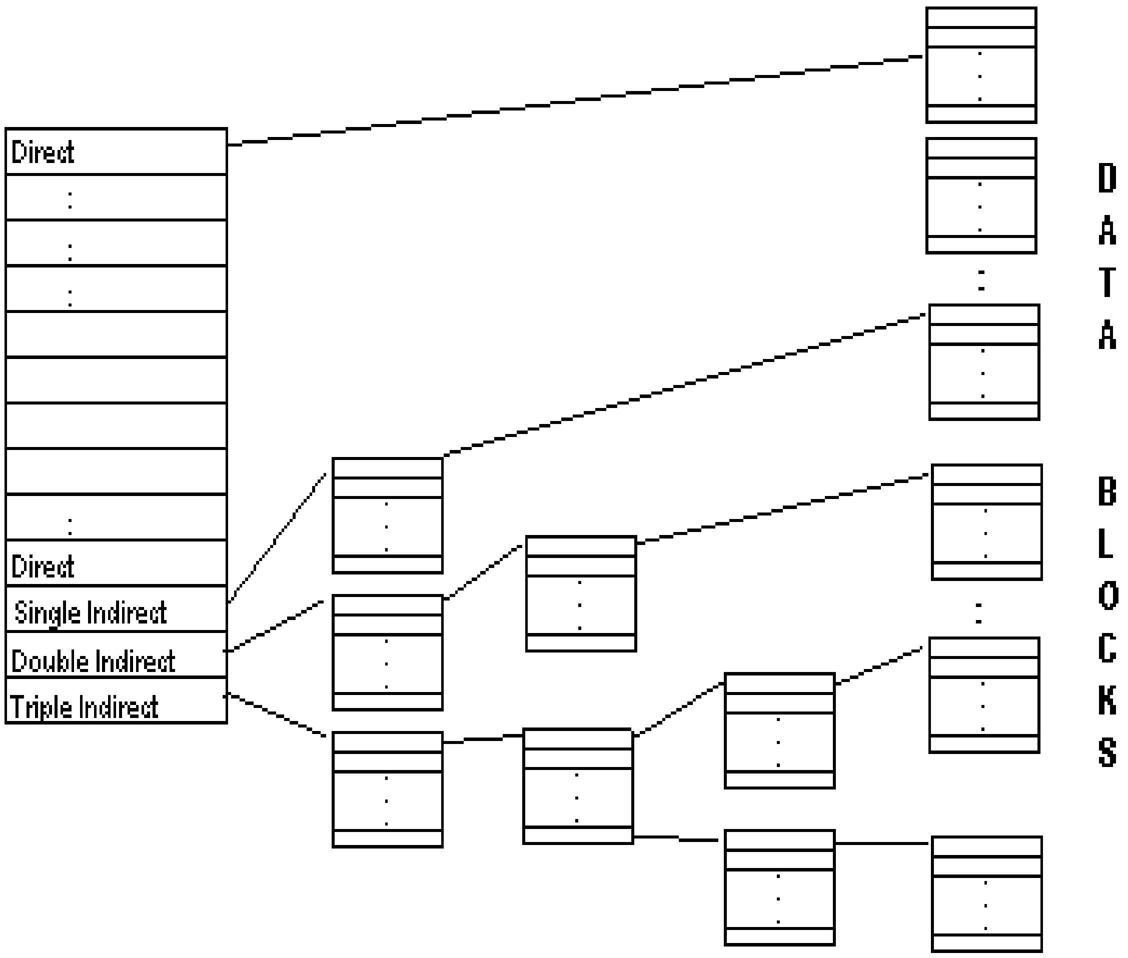
Triple Indirect Block Pointers:

Now this triple Indirect Block Pointers can address upto 4G * 1024 = 4TB, of file size. The fifteenth block pointer in the inode will point to the block just after the 4G of data, which intern will point to 1024 Double Indirect Block Pointers.

So after the 12 direct block pointers, 13th block pointer in inode is for Indirect block pointers, and 14th block pointer is for double indirect block pointers, and 15th block pointer is for triple indirect block pointers.

Now this is the main reason why there are limits to the full size of a single file that you can have in a file system.

<u>Direct and Indirect Blocks in Inode</u>

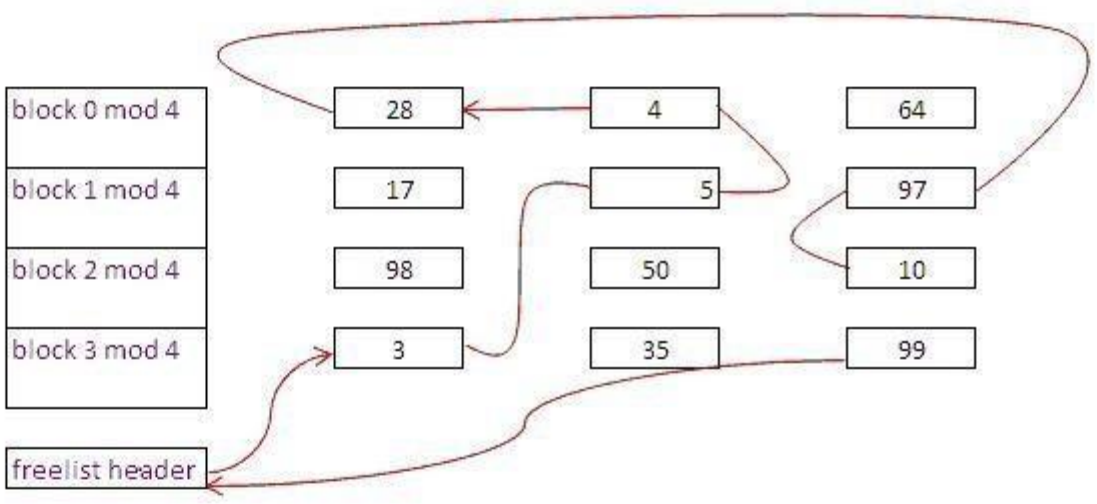


Inodes resides on the disk and the kernel reads them into an into memory which we can call as in-core inodes. Disk inodes contains the following information:

File access permissions and time (last access / modified etc.)
File ownership information.
Type of the file (regular / directory/block special/pipe)
Number of links to the file
- File size and organization on disk (the file data may spread across several different and far- spaced disk location)

The In-core copy of inodes contains all of the above information, but it also contains the following additional information:

- Status (locked / process is waiting for it to become unlocked / in-core copy has been modified and thus differs from the copy on the disk / mounted)
- Logical Device number of the file system
- Inode Number. Since inodes are stored in a sequential manner on the disk, the kernel uses an identifier of that array to refer to its in-core copy.
- Pointer to other in-core inodes. Kernel maintains a hash queue of inodes according to the logical device number and the inode numbers. Kernel also maintains a list of free inodes.
- Reference count which indicates the number of instances of the file that are currently active.



* ***Structure of a Regular file on Disk*** As stated previously inodes contains the table of content of the file data on disk. As each disk block can be referenced by a number, the table of content is nothing but a sequence of disk block numbers. The file data may not always be stored in contiguous memory locations, hence

we will need to keep track of all the block numbers on the disk. The ***system V Unix*** systems have the following 13 entries as the table of contents :

The blocks marked "direct" can refer to a single disk block that contains the real data. "single indirect" block contains the number of a disk block which in itself contains a list of block numbers that we can reference and they have the real data. Going on the same line, we have "double indirect" and "triple indirect" blocks. Lets now try to get an estimate of the max limit on the size of a file that Unix File System can handle.
Assume one block is of 1 Kbytes and a block number is an integer of 4 bytes (32 bits). Thus a block can have 256 block numbers.
10 direct blocks => 10 K bytes
1 single indirect block with 256 block number entries => 256 K bytes
1 double indirect block with 256 single indirect entries => 64 M bytes
1 triple indirect block with 256 double indirect entries => 16 G bytes
which is far more then what the 4-byte memory address can handle (***2^32 => 4 G bytes)***.
So, whenever a process wants to access any particular offset in a file, it will simply use this table of indexes and thus would load the appropriate disk block into memory.

**** Structure of directory***
Directories are the files that give the file system its hierarchical structure. A directory on Unix file systems is a file which contains a sequence of entries where each entry contains an inode number and the name of the file contained in the directory. UNIX System V restricts name to a maximum of 14 characters, and 2 byte entry for the inode number making it 16 bytes
per entry.

Byte Offset	***Inode number***	***File names***
0	83	.
16	2	..
32	1798	init
48	1276	fsck
64	85	clri
80	1268	motd
96	1799	mount

So, that was an insight into the inode data-structure. Follow this link for an understanding of the interaction between different data structure that link up with inodes.