# UNIX Study material

## Hardware

Kernal (heart of UNIX)
- Memory management
- Task scheduling
- File management
- Shell - process the request

## Vi Editor:

To undo in vi editor: come out of insert mode and type :undo similarly use :redo
Listing directories in UNIX: ls -ltr | grep ^d
^ - start of line.
$ - end of line.
Se nu - to set line in vi editor
Se nu!
Se wrap, se nowrap
Se list, se nolist - to highlight blank space
Se ic - set ignore case
HJKL - move between the lines
dd - to delete lines
dw -delete the current word
x -delete the current word
X -delete the word previous to current cursor
D - delete words from the current cursor location to end of line
Shift o - insert a new line before the cursor point
o - insert a new line after the cursor point

Replacing text:
:%s/search/replace/g

Search:
The / command searches forwards (downwards) in the file.
The ? command searches backwards (upwards) in the file.
The n and N commands repeat the previous search command in the same or opposite direction, respectively.

| | |
|---|---|
| [redacted] | [redacted] |
| . | Matches a single character. |
| * | Matches zero or more of the previous character. |
| [redacted] | [redacted] |

Run command form vi editor : - :! <command name> like :! ls -ltr

yy - copy the current line
yw -copy current word
P - paste the copied content

https://www.tutorialspoint.com/unix/unix-vi-editor.htm

## ls

ls -s (listing file sorting by size)

ls -ltrh (human readable; size of file in KB)

ls -l --time=atime
atime — updated when file is read
mtime — updated when the file changes.
ctime — updated when the file or owner or permissions changes.

**Find group where user belongs to:**
groups <user name>
w - Show who is logged on and what they are doing.
Users
Who
Finger

**Cat**
Cat -b filename to display line number.

ls -ltr | awk '{print $3}'
ls -ltr | awk '/^d/ {print $3}'

Create recursive directories : mkdir [redacted] a/b/c , here a, b ,c is directory name.

**Initialization in UNIX**
When you log in to the system, the shell undergoes a phase called initialization to set up the environment. This is usually a two-step process that involves the shell reading the following files -

- /etc/profile
- profile

The process is as follows -

- The shell checks to see whether the file /etc/profile exists.
- If it exists, the shell reads it. Otherwise, this file is skipped. No error message is displayed.
- The shell checks to see whether the file .profile exists in your home directory. Your home directory is the directory - that you start out in after you log in.
- If it exists, the shell reads it; otherwise, the shell skips it. No error message is displayed.

**Permissions in UNIX:**

Owner(u) group(g) Others(o)
xxx xxx xxx
Read - 4
Write - 2
Execute - 1

chmod u+r,u+x <filename>
chown and chgrp

chown changes owner of file; chown <new owner> < filename>
chgrp changes owner of file; chgrp <new owner> <filename>

'b'
block special file
'c'
character special file

(PRD) user@server$ ls -ltr null
[redacted]rw-rw-rw- 1 root root 1, 3 Feb 10 11:58 null

## suid(set user id) and sgid(set group id):

When you execute a program that has the SUID bit enabled, you inherit the permissions of that program's owner. Programs that do not have the SUID bit set are run with the permissions of the user who started the program.
Additional permissions are given to programs via a mechanism known as the Set User ID ( SUID) and Set Group ID ( SGID) bits.

```
$ ls -l /usr/bin/passwd
-r-[redacted]r-xr-x  1   root   bin  19031 Feb 7 13:47  /usr/bin/passwd*

$ chmod ug+[redacted] dirname
$ ls -l
drwsr-sr-x 2 root root  4096 Jun 19 06:45 dirname
$
```

```
 mailx -s "Test message" <something>@<something>.com
(echo -e "$Message"; uuencode $file1 $file1) | mailx -s "<mail Subject>"  $TO
```

## ECHO:

echo -n; here **-n** option let echo to avoid printing a new line character.
Pasted from <https://www.tutorialspoint.com/unix/unix-shell-loops.htm>

Echo -e  Enable interpretation of backslash escape sequences (see below for a list of these).

```
$ echo -e "hello \n world"
hello
 world
```

https://www.tutorialspoint.com/unix/unix-shell-substitutions.htm

## GREP:

Grep -c --count of matched pattern <File name> : <count of occurrence>
grep -l  -- display the name of the file where pattern matched. <file name>
Gerp -n -- Print the matched line and its line number. <file name> : <line number>: <matched line>
Grep -i - ignore case
Grep - H -  Print the filename for each match. <file name> : <matched line>
grep -B will display lines before the matched pattern grep -A 1 "learn" file.txt
grep -r to search content recursively in folders.
grep --color "John" employee.txt -
gerp -w - search exact word only
Grep -f https://www.howtoforge.com/linux-fgrep-command/

```
(PRD) guptvag@lonrs10639$ grep -f b.txt a.txt (Right out join)
1
2
3
10
20
30
(PRD) guptvag@lonrs10639$ grep -x -f b.txt a.txt (intersect)
1
2
3
(PRD) guptvag@lonrs10639$ cat a.txt
1
```

2
3
10
20
30
(PRD) guptvag@lonrs10639$ cat b.txt
1
2
31
2
31
2
3

If you want the output of your searched pattern in color then use the command given below

**egrep** is the same as grep -E. fgrep is the same as grep -F. rgrep is the same as grep -r. egrep is an acronym for "Extended Global Regular Expressions Print". It is a program which scans a specified file line by line, returning lines that contain a pattern matching a given regular expression.

egrep -c "ORA-[0-9][0-9][0-9][0-9][0-9]*" ${output_file_name}

Pasted from <http://www.cs.columbia.edu/~tal/3261/fall07/handout/egrep_mini-tutorial.htm>

## Sort:

| | |
|---|---|
| **-n** | Sort numerically (example: 10 will sort after 2), ignore blanks and tabs. |
| **-r** | Reverse the order of sort. |
| **-f** | Sort upper- and lowercase together. |
| **+x** | [redacted] |

$ls -l | grep "Aug" | sort +4n

Sort 5th column : - sort -nr **-k 5**
Sort –u --this sort the column and remove duplicates
uniq -c --count of duplicates
uniq -d --print only duplicates

## Less command:

/ for forward search
? For backward search
n - for next match in forward
N- for next match in backward

## More , less in unix

Unlike "more", less allows for both forward and backward navigation so you can look for any search string at any point in the document.
https://helpdeskgeek.com/linux-tips/more-less-command-linux-unix/

## Process in Unix:

Foreground process: ls -ltr
Background process : ls -ltr &

Switch background process to foreground.

- Jobs - this list down all the background job
- fg n - this move background to foreground.

## Job and process

https://www.daniweb.com/programming/software-development/threads/159920/difference-btw-a-process-and-a-job

Background and suspended processes are usually manipulated via **job number (job ID)**. This number is different from the process ID and is used because it is shorter.
In addition, a job can consist of multiple processes running in a series or at the same time, in parallel. Using the job ID is easier than tracking individual processes.

The "job" is any process that is running in a background mode. Job can have multiple processed running..

Ps(process status) -ef (full) --- there is no memory utilization column in ps -ef

## Ps - elf (detail listing)

UID PID PPID "CPU utilization"
UID PID PPID C STIME TTY TIME CMD
UID- user id
PID- process id
PPID- parent process id
C - cpu utilization
STIME - process start time
TTY - terminal type
TIME - cpu time
CMD - name of command.

## Hierarchal process in UINX:

https://www.unixtutorial.org/2008/05/how-to-show-a-processes-tree-in-unix/
Pstree <processid> /<username> [pstree <username> shows all running process with this particular user]
Ps -aef --forest

**sshd** (SSH Daemon) is the daemon program for ssh(1). Together these programs replace rlogin and rsh, and provide secure encrypted communications between two untrusted hosts over an insecure network.

## Unix Nohup:

Run a Command or Shell-Script Even after You Logout
When you execute a Unix job in the background ( using &, bg command), and logout from the session, your process will get killed. You can avoid this using several methods — executing the job with nohup, or making it as batch job using at, batch or cron command.

## Kill

The Kill command in unix or linux operating system is used to send a signal to the specified process or group. If we dont specify any signal, then the kill command passes the SIGTERM signal. We mostly use the kill command for terminating or killing a process. However we can also use the kill command for running a stopped process.
kill [-s signal] pid
kill -l

The command kill sends the specified signal to the specified process or process group. **If no signal is specified, the TERM signal is sent. The TERM signal will kill processes which**

**do not catch this signal**. For other processes, it may be necessary to use the KILL (9) signal, since this signal cannot be caught.

| Number | Signal Name | Description |
|---|---|---|
| 0 | SIGNULL | Used to check access to the process id |
| 1 | SIGHUP | **Hup** signal. Terminates the process. |
| 2 | SIGNINT | Interrupt signal. Terminating the process |
| 3 | SIGQUIT | Quit signal. Terminate process with core dump |
| 9 | SIGKILL | Forcibly killing a process |
| 24 | SIGSTOP | Pausing the process |
| 26 | SIGCONT | Runs a stopped process |
| 17 | SIGCHLD | |

Pasted from <http://www.folkstalk.com/2012/07/kill-command-examples-in-unix-linux.html>

**Zombie and Orphan Processes**

Normally, when a child process is killed, the parent process is told via a SIGCHLD (number 17) signal. Then the parent can do some other task or restart a new child as needed. However, sometimes the parent process is killed before its child is killed. In this case, the "parent of all processes," init process, becomes the new PPID (parent process ID). Sometime these processes are called orphan process.

When a process is killed, a ps listing may still show the process with a Z state. This is a zombie, or defunct, process. The process is dead and not being used. These processes are different from orphan processes. They are the processes that has completed execution but still has an entry in the process table.

**Daemon Processes (nohup background process running with root user permission)**

Daemons are system-related background processes that often run with the permissions of root and services requests from other processes.

A daemon process has no controlling terminal. It cannot open /dev/tty. If you do a "ps -ef" and look at the tty field, all daemons will have a ? for the tty.

```
lon3302xus:08:53:51:~ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0  2015 ?        00:00:05 init [3]
root         2     1  0  2015 ?        00:00:35 [migration/0]
```

More clearly, a daemon is just a process that runs in the background, usually waiting for something to happen that it is capable of working with, like a printer daemon is waiting for print commands.

**If you have a program which needs to do long processing then its worth to make it a daemon and run it in background.**

**Top command:**

http://www.dba-oracle.com/t_linux_oracle_monitor_with_top.htm

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND

PID -- Process Id : This is a unique number used to identify the process.

User : The username of whoever launched the process.

PR -- Priority : The priority of the process. Processes with higher priority will be favored by the kernel and given more CPU time than processes with lower priority. Oddly enough, the lower this value, the higher the actual priority; the highest priority on *nix is -20 and the lowest is 20.

NI -- Nice value : nice is a way of setting your process' priority. See here for more details.
VIRT -- Virtual Memory Size (KiB) : The total amount of virtual memory used by the process.
RES -- Resident Memory Size (KiB) : The non-swapped physical memory a task has used.
SHR -- Shared Memory Size (KiB) : The amount of shared memory available to a task, not all of which is typically resident. It simply reflects memory that could be potentially shared with other processes.
S -- Process Status : The status of the task which can be one of:

'D' = uninterruptible sleep
'R' = running
'S' = sleeping
'T' = traced or stopped
'Z' = zombie

%CPU -- CPU Usage : The percentage of your CPU that is being used by the process. By default, top displays this as a percentage of a single CPU. On multi-core systems, you can have percentages that are greater than 100%. For example, if 3 cores are at 60% use, top will show a CPU use of 180%. See here for more information. You can toggle this behavior by hitting Shifti while top is running to show the overall percentage of available CPUs in use.
%MEM -- Memory Usage (RES) : A task's currently used share of available **physical memory** (RAM).
TIME+ -- CPU Time, hundredths : Total CPU time the task has used since it started.
COMMAND -- Command Name or Command Line : To see the full command line that launched the process,

**Swap** space in **Linux** is used when the amount of physical **memory** (**RAM**) is full. If the system needs more **memory** resources and the **RAM** is full, inactive pages in **memory**are moved to the **swap** space. While **swap** space can help machines with a small amount of **RAM**, it should not be considered a replacement for more **RAM**.

start top with the -c flag : top -c.
To see specific process using top - **top -cp**
**top -a - sort output by memory utilization**
**top -d to increase/decrease delay time**

Ping:
2 packets transmitted, 22 received, 0% packet loss, time 21017ms

**Fpt:**
file transfer protocol
Ftp <machinename>

**telnet:**
Utility to connect to remote host.
Command is "telnet <hostname>"

**Finger:**
The finger command displays information about users on a given host. The host can be either local or remote.
bash-3.00$ finger

| Login | Name | Tty | Idle | Login Time | Office | Office Phone |
|---|---|---|---|---|---|---|
| guptvag | Vishal Gupta | pts/0 | | Oct 21 14:34 (28.151.178.85) | | |
| sinclga | Gary Sinclair | pts/1 | 7d | Oct 14 09:00 (lonmw38893.fm.rbsgrp.net) | | |

**Other user related commands -** w, whoami, users, groups

vastool list netgroups - command to list netgroups

## Shell:

It's an interface to UNIX system.
It gathers input from you and executes programs based on that input. When a program finishes executing, it displays that program's output.
There are different flavors of shells, just as there are different flavors of operating systems
The Bourne shell is usually installed as /bin/sh on most versions of UNIX.

Shell type:
**1.) Bourne shell : sh, bash, ksh are examples. the default prompt is the $ character.**
Bourne shell ( sh)
Korn shell ( ksh)
Bourne Again shell ( bash)
POSIX shell ( sh)

**2.) C shell: the default prompt is the % character.**
C:
TENEX/TOPS C shell ( tcsh)

To get process id of shell : echo $$
Command to get current shell: echo $SHELL
Exit status: $?

| | |
|---|---|
| **$0** | The filename of the current script. |
| **$n** | These variables correspond to the arguments with which a script was invoked. Here n is a positive decimal number corresponding to the position of an argument (the first argument is $1, the second argument is $2, and so on). |
| **$#** | The number of arguments supplied to a script. |
| **$*** | All the arguments are double quoted. If a script receives two arguments, $* is equivalent to $1 $2. |
| **$@** | All the arguments are individually double quoted. If a script receives two arguments, $@ is equivalent to $1 $2. |
| **$?** | The exit status of the last command executed. |
| **$$** | [redacted]. For shell scripts, this is the process ID under which they are executing. |
| **$!** | The process number of the last background command. |

Input to a script : read command
Readonly variables: readonly <var name>

Arithmetic operation
https://www.tutorialspoint.com/unix/unix-basic-operators.htm

Substitution -
Command substitution -
UP=`date ; uptime`
echo "Uptime is $UP"
Variable substitution -

| .No. | Form & Description |
|---|---|
| 1 | **${var}**<br>Substitute the value of *var.* |
| 2 | **${var:-word}**<br>If *var* is null or unset, *word* is substituted for **var**. The value of *var* does not change. |
| 3 | **${var:=word}**<br>If *var* is null or unset, *var* is set to the value of **word**. |
| 4 | **${var:?message}**<br>If *var* is null or unset, *message* is printed to standard error. This checks that variables are set correctly. |
| 5 | **${var:+word}**<br>If *var* is set, *word* is substituted for var. The value of *var* does not change. |

Pasted from <https://www.tutorialspoint.com/unix/unix-shell-substitutions.htm>

## Rename -

We often use "mv" command to rename a single file in Linux. However, renaming multiple or group of files quickly makes it very difficult task in a terminal.

Linux comes with a very powerful built-in tool called rename. The rename command is used to rename multiple or group of files, rename files to lowercase, rename files to uppercase and overwrite files using perl expressions.

rename -v 's/\.php$/\.html/' *.php

https://www.tecmint.com/rename-multiple-files-in-linux/

## EXPORT - used to make a variable as environment variable

There are instances or situations where we would like to define a variable and it should be accessed in all the shells or processes invoked by the original shell. This can be achieved by the **export** command in ksh/sh as shown below.

```
export FILE="output.txt"
#echo $FILE
output.txt
#ksh
#echo $FILE
output.txt
#
```

This FILE variable is now an environment variable. An environment variable is the one which can be accessed across all the shells or processes initiated from the original shell of the environment. So, in ksh/sh, a variable can be made an environment variable using the export command.
http://www.theunixschool.com/2010/04/what-is-difference-between-export-set.html

**Array in unix:**

array_name=(vishal1 gupta)
echo ${array_name[0]}
Length of array - echo ${[redacted]array_name[@]}

Arithmetic operation:
echo `expr 2 / 2`

echo -e "Value of a is $a \n"
Enables interpretation of \n

- You can access all the items in an array in one of the following ways –
  ${array_name[@,*]}

**If condistion in unix:**

```
if [ expression ]
then
  Statement(s) to be executed if expression is true
fi

if [ expression ]
then
  Statement(s) to be executed if expression is true
else
  Statement(s) to be executed if expression is not true
fi

for FILE in $HOME/.bash*
do
  echo $FILE
Done

for file in /home/guptvag/*; do echo $file; done
for line in {1..5}; do  echo $line; done
```

**While loop to print line:**

[redacted]

Meta characters:
* ? [ ] ' " \ $ ; & ( ) | ^ < > new-line space tab

**Escape characters:**

| | |
|---|---|
| **Backslash** | Any character immediately following the backslash loses its special meaning. |

echo hello \\\\ world
Prints - hello \\ world

Quoting In unix:

| Quoting | Description |
|---|---|
| [redacted] | [redacted] |
| **Double quote** | Most special characters between these quotes lose their special meaning with these exceptions: |

| | |
|---|---|
| | • $<br>• `<br>• \$<br>• \'<br>• \"<br>• \\ |
| **Backslash** | Any character immediately following the backslash loses its special meaning. |
| **Back Quote** | Anything in between back quotes would be treated as a command and would be executed(`date`) |

bash-3.00$ echo `uptime`
16:34:42 up 355 days, 1:30, 1 user, load average: 0.84, 0.81, 0.81

## Shell input and output redirections:

Output redirection: >
Input redirection : <

command1 > out.txt 2> err.txt
command1 > everything.txt 2>&1

Redirect output and view the output.
cat file.txt | tee err.txt
command1 |& tee log.txt

**tee - display the output along with saves the content in file.**
**ls| tee file.txt1**

https://www.cyberciti.biz/faq/linux-redirect-error-output-to-file/
http://sc.tamu.edu/help/general/unix/redirection.html

## WC: output

- Number of line
- Number of word
- Size of file (byte)
- Name of the file

```
$ wc -l users
2 users
$

$ wc -l < users
2
$
```

## Here document:

```
$wc -l << EOF
  This is a simple lookup program
      for good (and bad) restaurants
      in Cape Town.
EOF
```

3
$

In the first case, wc knows that it is reading its input from the file users. In the second case, it only knows that it is reading its input from standard input so it does not display file name.

## Function in Unix:

```
#!/bin/sh
# Define your function here
Hello () {
  echo "Hello World"
}

# Invoke your function
Hello
```

Return value from function $?. This  Capture value returnd by last command

a=${fun 2}

uname -a shows command shows the kernel version and other things.

## Memory state:

cd /proc/
Vmstat -s  - prints all virtual memory statistics

Free -m (m for MB, G for GB)
Df -kh
Du -sh (du -kh, du -mh)
http://www.binarytides.com/linux-command-check-memory-usage/

free -m -g
df -kh
du -sh
vmstat
cd /proc/meminfo

## UNIX networking

Netstat - Prints network connections, routing tables, interface statistics, masquerade connections, and multicast memberships, port

Netstat -lntu

-l – prints only listening sockets
-n – shows port number
-t – enables listing of tcp ports
-u – enables listing of udp ports

Cat /etc/services | grep -I smpt  **--- list down all the running services**

```
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 localhost.localdomain  localhost.localdomain  ESTABLISHED
```

## SED in Unix:

http://www.folkstalk.com/2012/01/sed-command-in-unix-examples.html

Sed is a Stream Editor used for modifying the files in unix (or linux). Whenever you want to make changes to the file automatically, sed comes in handy to do this. This doesn't saves the data
sed 's/unix/linux/n' file.txt here n is the occurrence of string. For global replacement use 'g'
If you want to save changes
sed 's/xxxxxxActive/Active/g' meminfo3002.txt > meminfo3002.txt

sed [redacted] 's/xxxxxxActive/Active/g' meminfo3002.txt - make changes in existing file

The "c" command to sed tells it to change the line.
>sed '/unix/ c "Change line"' file.txt
"Change line"
learn operating system.
"Change line"

sed 's/^\(.\)\(.\)\(.\)/\3\2\1/g' awk1.txt

Sed-e use to run multiple commands in UNIX
sed 's/unix/linux/' file.txt| sed 's/os/system/'
sed -e 's/unix/linux/' -e 's/os/system/' file.txt

Sed as grep:
sed -n '/searchpatter/ p' meminfo3002.txt

Sed 'p' file.txt
Prints the line twice

Delete in sed
sed '2 d' meminfo3002.txx , sed '1,2 d' meminfo3002.txt

Sed -n '2p:10p' filename.txt

sed '/unix/ i "Add a new line"' file.txt
a- add line after the searched line
i - add line befor the searched line
c - replaces the searched line

sed 'y/ul/UL/' file.txt
The sed command can be used to convert the lower case letters to upper case letters by using the transform "y" option.

Also we can use sed like sed 's:HighTotal:High:g' meminfo3002.txt. Here : is used in place of /

Delete blank line from file
sed '/^$/d' <filename>
Use i option to ignore case while searching
sed 's/abc/xyz/[redacted]g' <filename>

The characters **a through z** and the characters **A through Z**, constitute one such class of characters that has the keyword **[[:alpha:]]**
cat /etc/syslog.conf | sed -n '/^[[:alpha:]]/p'

**[[:digit:]]**

Numbers [0-9]

5555551212
5555551213
sed -e 's/^[[:digit:]][[:digit:]][[:digit:]]/(&)/g' phone.txt
(555)5551212
(555)5551213

Pasted from <https://www.tutorialspoint.com/unix/unix-regular-expressions.htm>

**Read write in SED:**
sed '1,2 r file.txt1' awk1.txt
sed -n '1,2 w file.txt2' awk1.txt
http://www.theunixschool.com/2011/04/sed-read-from-file-or-write-into-file.html

## File system

**Mount** - show all the mounts on Unix server
Example - <filename>:<path>on <path on host> type nfs (rw,nosuid,addr=<ip address>)

**NFS (network file system)**
NFS stands for Network File System, a file system developed by Sun Microsystems, Inc. It is a client/server system that allows users to access files across a network and treat them as if they resided in a local file directory. For example, if you were using a computer linked to a second computer via NFS, you could access files on the second computer as if they resided in a directory on the first computer. This is accomplished through the processes of exporting (the process by which an NFS server provides remote clients with access to its files) and mounting (the process by which file systems are made available to the operating system and the user). The NFS protocol is designed to be independent of the computer, operating system, network architecture, and transport protocol. This means that systems using the NFS service may be manufactured by different vendors, use different operating systems, and be connected to networks with different architectures. These differences are transparent to the NFS application, and thus, the user.

## Find in UNIX:

http://www.folkstalk.com/2011/12/101-examples-of-using-find-command-in.html

find -maxdepth 1 -name "sum.java"
find for a file in the current directory only

How to print the files in the subdirectories between level 1 and 4?
find -mindepth 2 -maxdepth 5 -name "sum.java"

How to find the files which are created between two files.
find . -cnewer f1 -and ! -cnewer f2

find . -type f -exec ls -s {} \; | sort -nr | head -1
ls -s = list file on basis of size.

Find . -not -name "xy*"
Find . -not -mtime -1

Find command to find file created in particular day
find ./ -type f **-daystart** -ctime **9** -exec ls -ltr {} \;

find . -type f -**newermt** "2018-05-01 10:42"

**Difference between xargs and exec:**
This is where -exec breaks down and xargs shows its superiority. When you use -exec to do the work you run *a separate instance* of the called program for each element of input. So if find comes up with 10,000 results, you run exec 10,000 times. With xargs, you build up the input into bundles and run them through the command as few times as possible, *which is often just once*. When dealing with hundreds or thousands of elements this is a **big** win for xargs.
Pasted from <https://danielmiessler.com/blog/linux-xargs-vs-exec/>

Find command to rename files
find -name "sed*" -exec mv {} {}_vis \;
Here {} is used for substitution

find the files which are modified after the modification of a give file
find -newer "sum.java"
anewer -access after modification of give file
cnewer - changed after modification of give file.

Files accessed(a - atime, amin), modified(m - mtime, mmin) and changed(c - ctime,cmin) with time.

1 find /home/ -type f -size 6579c -exec ls {} \;

As units you can use:
b – for 512-byte blocks (this is the default if no suffix is used)
c – for bytes
w – for two-byte words
k – for Kilobytes (units of 1024 bytes)
M – for Megabytes (units of 1048576 bytes)
G – for Gigabytes (units of 1073741824 bytes)

Df -kh (disk free in human readable format)

## Difference in files

diff cash.sh cash1.sh - shows difference between 2 files.
vmdiff shows difference in file in vm

## Cut using delimiter:

cut -c1,2 meminfo3002.txt - cut content of file. - this wil cut 1 and 2 character of file (position)
cut -c6-15 file.txt - this will cut file from character 6 to 11 in range. (range)

cut -d, -f 1 file1

Space delimiter in cut
cut -d" " -f 1 file.txt

http://www.theunixschool.com/2011/05/cut-cut-files-with-delimiter.html

Cut last char from string

- echo 123456 | rev | cut -c -3 | rev
- echo 123456 | tail -c 4

## How will you connect to a database server from linux?

We can use isql utility that comes with open client driver as follows:

isql –S serverName –U username –P password
jdbc:oracle:thin:@<hostname>:<port name>:<dbname>", DBUserName
${ORACLE_HOME}/bin/[redacted] -s ${DB_USER}/${DB_PASS}@${DB_SID}

```
$ cat << VIS
> kdkdk
> jsjdj
> VIS
kdkdk
jsjdj
```

```
 cat file.txt |while read line
> do
> for word in `echo $line`; do echo $word; done
> done
```

## Date:

date --date='1 days' +%y%m%d
http://www.thegeekstuff.com/2013/05/date-command-examples/
date +%j - display day of year

Set date and time using -s option.
$ date -s "Sun May 20 21:00:00 PDT 2013"
Sun May 20 21:00:00 PDT 2013

## AWK:

Awk is one of the most powerful tools in Unix used for processing the rows and columns in a file. Awk has built in string functions and associative arrays. Awk supports most of the operators, conditional blocks, and loops available in C language.

Search pattern in AWK

Awk '{/<pattern1>/, /<\/ pattern1>/' file name

Print file name in Awk
awk '{print FILENAME}' *.*

http://www.folkstalk.com/2011/12/good-examples-of-awk-command-in-unix.html
awk 'BEGIN {start_action} {action} END {stop_action}' filename
ls -ltr | awk 'BEGIN{for(i=1;i<5;i++)print i*i;}'
ls -ltr | awk 'BEGIN{}{if($5 == "4096")print $5;}'

NR - print number of row.
NF - print number of column.

awk 'END {print NR}' input_file
This will display the total number of lines in the file

Awk to covert to upper case:
awk '{print toupper($0)}' file.txt
http://linuxcommando.blogspot.in/2008/05/how-to-convert-text-files-to-all-upper.html

**String functions in Awk:**

Some of the string functions in awk are:

index(string,search) - awk '{print index($5,<put string to be searched>)}'
length(string)
split(string,array,separator)
substr(string,position)
substr(string,position,max)
tolower(string)
toupper(string)

**Awk examples:**

```
for file in `ls`
do
awk -F "<tradeId>" '{print $2"|"$3"|$4"|"$5}' $file | awk -F "</tradeId>" '{print $1}' >>
/home/guptvag/trade_list1.txt
done

for file in `ls`
do
sed 's/<tradeId>/|/g' $file | sed 's/<\/tradeId>/|/g' | awk -F '|' '{print $2"|"$4"|"$6"|"$8"|"$10"|"$12}'
>> /home/guptvag/trade_final.txt
done

for file in `ls anubis-web.log.2017-05-2*`
> do grep -n "ERROR PROCESSING INTERNAL TRANSACTION:" $file | awk -F "ERROR
PROCESSING INTERNAL TRANSACTION:" '{print $2}'
> done
```

**Tr (transform command):**

Tr stands for translate or transliterate. The tr utility in unix or linux system is used to translate, delete or squeeze characters. The syntax of tr command is

```
> echo "linux dedicated server" | tr "[:lower:]" "[:upper:]"
LINUX DEDICATED SERVER
> echo "linux dedicated server" | tr "[a-z]" "[A-Z]"
LINUX DEDICATED SERVER
```

Pasted from <http://www.folkstalk.com/2012/10/translate-tr-command-examples-in-unix.html>

**FS:**
So far, we have seen the fields separted by a space character. By default Awk assumes that fields in a file are separted by space characters. If the fields in the file are separted by any other character, we can use the FS variable to tell about the delimiter.
awk 'BEGIN {FS=":"} {print $2}' input_file
OR
awk -F: '{print $2} input_file
Pasted from <http://www.folkstalk.com/2011/12/good-examples-of-awk-command-in-unix.html>

**OFS -** Output field separator variable:
By default whenever we printed the fields using the print statement the fields are displayed with space character as delimiter. For example
awk 'BEGIN {OFS=":"} {print $4,$5}' input_file
ls -ltr | awk 'BEGIN{OFS=":"}{print $1,$2}'

**Find in Awk:**
http://www.theunixschool.com/2012/05/awk-match-pattern-in-file-in-linux.html

**EVAL:**
$ COMMAND="ls -ltr"
$ eval $COMMAND

http://www.folkstalk.com/2012/10/executes-eval-arguments-as-shell.html

## Cron jobs:

Crontab: http://www.thegeekstuff.com/2009/06/15-practical-crontab-examples

Minute hour "day of month" month "day of week" command

| Keyword | Equivalent |
|---|---|
| @yearly | 0 0 1 1 * |
| @daily | 0 0 * * * |
| @hourly | 0 * * * * |
| @reboot | Run at startup |

**Cron Job every weekday during working hours**

minute
hour
day of month
month
day of week

This example checks the status of the database every weekday (i.e excluding Sat and Sun) during the working hours 9 a.m – 6 p.m.
00 09-18 * * 1-5 /home/ramesh/bin/check-db-status

- **00** – 0th Minute (Top of the hour)
- **09-18** – 9 am, 10 am,11 am, 12 am, 1 pm, 2 pm, 3 pm, 4 pm, 5 pm, 6 pm
- * – Every day
- * – Every month
- **1-5** -Mon, Tue, Wed, Thu and Fri (Every Weekday)

List currently logged cron jobs
crontab -l

Cron job running every 10 min:
*/10 * * * * /home/ramesh/check-disk-space

## Inode
http://www.grymoire.com/Unix/Inodes.html

"ls -i" lists the inode of a file

- We are used to thinking about a directory containing files. This is really an illusion. Directories do not contain files. The data of the files is not stored in the directory.
- A directory is really just a file. It's a special file with special rules (you can't just type "cp /dev/null directory" to erase it. It's got special bits to make sure a mere mortal can't mess it up. Because if a file system gets corrupted, then you can say goodbye to your data. On older UNIX systems,

you actually could "read" the contents, using 'cat .', of a directory. But let me get back to that in a second...

- A Unix file is "stored" in two different parts of the disk - the **data blocks and the inodes**. (I won't get into superblocks and other esoteric information.) The data blocks contain the "contents" of the file. The information about the file is stored elsewhere - in the inode.
- Both the inodes and data blocks are stored in a "filesystem" which is how a disk partition is organized. But these inodes are strange and confusing. Let me give you an introduction.
- "ls -i" lists the inode of a file

- Size of file (In bytes)
- Device ID (Device containing the file)
- User ID (of the owner)
- Group ID
- File Modes (how owner, group or others could access the file)
- Extended Attributes (like ACL)
- Files access, change or modification time stamps
- Link Count (no of hard links pointing to the inode ... remember, no soft links are counted here)
- Pointer to the disk block that stores the content.
- File type (whether file, directory or special block device)
- Block size of the filesystem
- No. of blocks file us using.

## Sticky bit:

http://www.thegeekstuff.com/2013/02/sticky-bit/

A Sticky bit is a permission bit that is set on a file or a directory that lets only the owner of the file/directory or the root user to delete or rename the file. No other user is given privileges to delete the file created by some other user.

# chmod +t allAccess/ - to add sticky bit from directory

# chmod -t allAccess/ - to remove sticky bit from directory

# ls -ld allAccess/

drwxrwxrwt 2 himanshu himanshu 4096 Oct 24 16:19 allAccess/

When a program is executed, it takes time to load the program into memory before the user can actually start using it. If a program, for example an editor is used frequently by users the start-up time delay was an overhead back then. To improve this time delay, the sticky bit was introduced. The OS checked that if sticky bit on an executable is ON, then the text segment of the executable was kept in the swap space. This made it easy to load back the executable into RAM when the program was run again thus minimizing the time delay.

ls -ld - this is used to list down directory

## Hard link and soft link:

ln file1 file2 - hardlink (file level)

ln -s file1 file2 - soft link (file as well as directory level)

If we do ls -ltr, the second column shows the number of link available for the file

rwxrw-r-- 1 root root 2048 Jan 13 07:11 afile.exe

- file permissions,
- number of links,
- owner name,
- owner group,

- file size,
- time of last modification, and
- file/directory name

**Soft links can span across Filesystems whereas hard links can be created only within a file system. Why so?**

Hard link refers to its file using the inode number. Inode numbers are specific within a file system. This means a file "f1" in filesystem "X" can have the same inode number as of a file "f2" in filesystem "Y". This is very much possible. For this reason, hard links cannot be shared across file systems.

However, the same is not true for soft links. Soft links refer to the files using the file name as seen in the earlier case. Hence, a soft links can be accessed across the file systems.

**Does soft links and hard links work only at the file level? Does it work at the directory level as well?**

Hard links work only at the file level. However, soft links works at the directory level as well.

**This particular use of soft links, to create links at the directory level, serves us a shortcut for many frequently used directories.**

Soft link have different I nodes. By default soft link created file/directory have full permission (777)

Hard link - inode number is same
Soft link - inode number is differnt

http://www.theunixschool.com/2012/03/soft-links-hard-links-all-about-inodes.html

## Tar and untar: - tape archive

In Unix, the name of the tar command is short for tape archiving, the storing of entire file systems onto magnetic tape, which is one use for the command. However, a more common use for tar is to simply combine a few files into a single file, for easy storage and distribution.

Replace inputfile1 and inputfile2 with the files and/or directories you want to combine. You can use any name in place of file.tar, though you should keep the .tar extension. If you don't use the f option, tar assumes you really do want to create a tape archive instead of joining up a number of files. The v option tells tar to be verbose, which reports all files as they are added.

https://kb.iu.edu/d/acfi
Gzip, gunzip

tar -cvf file.tar inputfile1 inputfile2 - to combine files
tar -xvf file.tar inputfile1 inputfile2 - to separate files

tar -cvzf file.tar.gz inputfile1 inputfile2 - to combine and compress files
tar -xvzf file.tar.gz inputfile1 inputfile2 - to separate and un-compress files

Gzip - to zip file
Gunzip to unzip

While command to read line from file:

```
#!/bin/bash
file=/etc/resolv.conf
while IFS= read -r line
do
    # echo line is stored in $line
    echo $line
done < "$file"
```

## SSH command :

SSH client utility in unix or linux server is used to logging into a remote host and execute commands on the remote machine. The rlogin and rsh commands can also be used to login into the remote machine. However these are not secure. The ssh command provides a secure connection between two hosts over a insecure network.

Using SSH public-key authentication to connect to a remote system is a robust, more secure alternative to logging in with an account password or passphrase. SSH public-key authentication relies on asymmetric cryptographic algorithms that generate a pair of separate keys (i.e., a key pair), one "private" and the other "public". You keep the private key a secret and store it on the computer you use to connect to the remote system. Conceivably, you can share the public key with anyone without compromising the private key; you store it on the remote system in a.ssh/authorized_keys directory.

Generate private and pubic key:
https://kb.iu.edu/d/aews

- ssh -keygen -t rsa/dsa - dsa and rsa are algorithms.
- It will prompt for file name ; input file name
- It will ask for password
- Once input is provided; public/private key will be placed at location **~/.ssh**

scp filename user@remote-host:/var/tmp/

Ssh user @hostname
http://www.folkstalk.com/2012/07/ssh-command-examples-unix-linux.html

TCP/IP protocol:
http://www.computerhope.com/jargon/t/tcpip.htm

## Database connectivity:

http://www.folkstalk.com/2012/06/connect-to-oracle-database-in-unix.html

```
${ORACLE_HOME}/bin/sqlplus -s ${DB_USER}/${DB_PASS}@${DB_SID} <<EOF
whenever sqlerror exit -1
SET SQLBLANKLINES ON
set trimspool on
set markup html on
set line 1000
set feed off
set pages 1000
break on date skip page
    spool ${Exc_rep}/MTR_Exception_Report
    @Query.sql
    spool off
set markup html off
quit
exit
EOF
```

Run script in debug mode:
1. Set -x
2. -x <Scriptname>

**Tasklist** is used to list running processes in Windows.

**Remove duplicate in Unix:**

uniq -c --count of duplicates

uniq -d --print only duplicates

http://www.theunixschool.com/2012/09/howto-remove-delete-duplicate-records.html
sort file | uniq
sort -u file
awk '!a[$0]++' file

! a [ $0 ] ++
first

a [ $0 ]
we look at the value of a[$0] (array a with whole input line ( $0 ) as key).

If it does not exist ( ! is negation in test will eval to true)

! a [ $0 ]
we print the input line $0 (default action).

Also, we add one ( ++ ) to a[$0] , so next time !a[$0] will evaluate to false.

Pasted from <http://126kr.com/article/69oq0961xlm>

**seq 10** will print number 1 to 10

Yes - this command will print endless y's

**traceroute**