Login and Logout

Switch User (SU) command is used to login to a user account, used to switch to superuser or any other user

$ su - user1

Password:

su - : reads the user's shell initialization files, also changes to the user's home directory

```
$ su - user1
Password:
$ pwd
/home/user1
```

su : does not read the user's shell initializing files, does not change over to the user's home directory

```
$ su user2
Password:
$ pwd
pwd: cannot access parent directories [Permission denied]
```

Logging out – Logging out can be done in any of he three ways

1) **Ctrl+d**
2) **exit**
3) **logout**

PWD **– Print Working Directory (also known as Current Working Directory),** displays the current directory.

```
$ pwd

/home/user1

$ cd dir1

$ pwd

/home/user1/dir1

$ cd ..

$ pwd

/home/user1
```

Date command is used to view the current date

```
$ date

Thu Oct 14 02:35:27 PDT 2010

date "+%D"

09/13/10
```

+%D – date in mm/dd/yy format

+%H – hour in 24 Hr format

+%M—displays only minute

+%S – displays only seconds

+%T – displays timestamp

+%a – displays weekday in three lettered format

+%b – displays month in three lettered format

+%d – displays day of month

+%y – displays year in yy format

+%Y – displays year in YYYY format

date "+%d-%b-%Y" – displays date in dd-mon-yyyy format

Who – displays information about all the users currently logged onto the system. The user name, terminal number and the date and time that each user logged onto the system.

```
who
root     :0           Oct 12 18:20
root     pts/0        Mar 23 11:47 (:1.0)
```

whoami – prints the currents user information, there is a difference between “who am I” and “whoami”

```
$ whoami
user1
$ pwd
/export/home/user1
$ su
password:
$ whoami
root
$ pwd
/export/home/user1
```

who am i - displays the login name of the original user

File Types

- Regular or ordinary files
- Directories
- Symbolic links
- Device files

```
drwxr-xr-x 2 adm adm 512 Apr 3 10:42 acct

lrwxrwxrwx 1 root root 14 Apr 3 11:05 aliases -> /mail/aliases

-rw-r--r-- 1 root bin 50 Apr 3 10:45 auto_home

-rw-r--r-- 1 root bin 113 Apr 3 10:45 auto_master

brw------- 1 root sys 136, 0 Apr 3 11:11 dad@0,0:a

crw------- 1 root sys 136, 0 Apr 3 11:11 dad@0,0:a,raw

brw------- 1 root sys 136, 1 Apr 4 11:06 dad@0,0:b

crw------- 1 root sys 136, 1 Apr 3 11:11 dad@0,0:b,raw
```

- : Regular files

d : Directories

l : Symbolic links

b : Block-special device files

c : Character-special device files

Creating a file :

```
touch file1

vi file2

cat > file3
```

cat command is used to display the contents of a file. Used without arguments it takes input from standard input <Ctrl d> is used to terminate input.

Creating a directory

```
mkdir dir1

mkdir –p dir2/dir3/dir4
```

-p flag is used to create directories recursively

ls – list directory contents

-l use a long listing format

-r reverse order while sorting

-R list subdirectories recursively

-t sort by modification time

-h print sizes in human readable format (e.g., 1K 234M 2G)

-d list directory entries instead of contents

-p displays a "/" beside a directory to show it's a directory

-a to list hidden files

cd – Change directory, navigating across directories

```
$ cd dir1

$ pwd

/home/dir1
```

Hidden files

Hidden files have a period (.) before their name

```
$ touch .hidden_file
$ mkdir .hidden_dir
$ ls –la
rw-r--r--  1 user1 group1  403 Mar 11 02:30 .hidden_file
drwxr-xr-x 3 user1 group1 4096 Mar 14 01:26 .hidden_dir
```

cp – copy a file or directory

```
$ cp file1 copy_file1
$ ls -l
-rw-r--r--  1 user1 group1 316 Mar 20 03:26 copy_file1
-rw-r--r--  1 user1 group1 316 Mar 20 01:59 file1
```

Moving / Renaming files or directories

```
$ mv file1 file_1
$ ls -l
-rw-r--r--  1 user1 group1 316 Mar 20 03:26 copy_file1
-rw-r--r--  1 user1 group1 316 Mar 20 01:59 file_1

$ mv dir1 dir_1
$ ls –l
drwxr-xr-x  2 user1 group1 4096 Mar 20 04:23 dir_1
drwxr-xr-x  2 user1 group1 4096 Mar 20 04:26 dir2
```

Deleting / removing a file or directory

rm – command used to remove a file/directory

```
$ rm –i file1

rm: remove regular empty file `file1 '? Y

$ rm –f file1

or

$ rm file1


$ rm –Rf dir2

Or

$ rmdir dir2 – directory dir2 should be empty for this to succeed
```

Wild Card Characters

These are used to replace one or a pattern of characters in working with files or directories

? - Single character

*** - Multiple character**

[ ] - Range of Characters

```
$ ls -l filer?

-rw-r--r--  1 user1 group1 0 Mar 29 23:48 filers

$ ls -l f????

-rw-r--r--  1 user1 group1 0 Mar 20 05:01 file1

-rwxr-xr-x  1 user1 group1 0 Mar 20 05:01 file2
```

```
$ ls -l file*
-rw-r--r--  1 user1 group1 0 Mar 20 05:01 file1
-rwxr-xr-x  1 user1 group1 0 Mar 20 05:01 file2

$ ls -l file*[1-3]
-rw-r--r--  1 user1 group1 0 Mar 20 05:01 file1
-rwxr-xr-x  1 user1 group1 0 Mar 20 05:01 file2
-rw-r--r--  1 user1 group1 0 Mar 29 23:49 file3
-rw-r--r--  1 user1 group1 0 Mar 29 23:49 filers1
```

Filters

pg - Displaying files or command outputs in view mode, Used to display data one page (screenful) at a time.

more - most predominantly used, an alternative to 'pg'

```
$ more file_name.ext
```

<space bar> - scroll by line

<enter> - scroll by page

VI – Editor

Modes : There are 3 modes of a VI editor as listed below

1) Write mode / Insert mode / Append mode / Text mode

2) Command mode

3) Command line mode / Last line mode

You have to be in one of the above three modes to enter commands into the editor, the command can be of any form like inserting text into a file, navigating across different lines in a file, page up/down, delete a letter/word or a complete line, copy/paste texts etc. The below commands require you to be in the Command mode of the editor. Hit [Esc] <command>

Commands

a - Append text after the current character

A - Append text at the end of the current line

i - Insert text before the current character (position of the cursor)

I - Insert text at the beginning of the line

o - Open up a line under the current line or the position of the cursor

O - Open up a line above the current line or the position of the cursor

x - Delete the current character

X - Delete the character before the current character

yy / YY - Yanks (Copies) the entire line and store in the buffer

5yy - Yank 5 lines of text and store in the buffer

p - Paste the deleted line under the current line

P - Paste the deleted line above the current line

r - Replace just one character

R - Replace the whole line

u /U - undo the last change to the file

dd / D - deletes the entire line

dw - deletes a word

/acme - searches the entire file for the word called “acme” and proceeds further

or backtracks with n/N commands from top to bottom

?acme - searches the entire file for the work called “acme” and proceeds further

or backtracks with n/N commands from bottom to top

G - jumps to the last line of the file

$ - moves the cursor to the end of the current line

0 or ^ - moves the cursor to the beginning of the current line

Following are some of the commands in the Command Line / Last line mode.

Hit [Esc] : <command>

<u>Commands</u>

: w - write to file

: wq - write to file and quit

: q - quit file

: se nu - set number to each line of file

: se nonu - removes numbers if they are numbered using : se nu editor command

:%s/search/replace/g - searches for the pattern "search" and replaces it with the pattern "replace" globally

:g/search/s//replace/g - Same as above, global search and replace

:1 - the cursor is placed at the beginning of the 1st line of the file

:15 - the cursor is placed at the beginning of the 15th line of the file

Navigating in a file using VI commands : Hit [Esc] <command>

<u>Commands</u>

Ctrl + d - Page down by half page (half window)

Ctrl + u - Page up by half page (half window)

Ctrl + f - page down by full page (page forward)

Ctrl + b - page up by full page (page backward)

Use of arrow keys inside a VI file can be misleading since VI has its own navigational keys which also work only in the command mode.

h - move right

j - move downward

k - move upward

l - move left

Word Count

Counting the number of lines/words and characters in a file

```
$ wc file3
 4  25 122 file3
```

```
wc -c file3
122 file3        No. of bytes/characters in the file
```

```
wc -w file3
25 file3              No. of words in the file
```

```
wc -l file3
4 file3               No. of lines in the file
```

Inodes

A unique number to a file or a directory

```
$ ls -li

total 24

4947970 -rw-r--r--  1 user1 group1    0 Mar 20 01:42 dir1

4947975 drwxr-xr-x  2 user1 group1 4096 Mar 20 01:43 dir2
```

Links – links are short cuts to files or directories

Types of Links

- **Hard links**
- **Soft or Symbolic links**

Hard links

```
$ ln file1 file2

$ ls -li
```

total 0

```
1282 -rw-r--r-- 2 root other 0 Apr 7 15:26 file1

1282 -rw-r--r-- 2 root other 0 Apr 7 15:26 file2
```

Soft links

```
$ ln -s file1 soft_link1

$ ls -li

-rw-r--r--  1 user1 group1  316 Mar 20 01:59 file1

lrwxrwxrwx  1 user1 group1    5 Mar 20 02:59 soft_link1 ->
file1
```

```
$ ls -li

total 24

4947970 -rw-r--r--  1 user1 group1    0 Mar 20 01:42 dir1

4947975 drwxr-xr-x  2 user1 group1 4096 Mar 20 03:00 dir2

4947981 -rw-r--r--  1 user1 group1  316 Mar 20 01:59 file1

4898831 lrwxrwxrwx  1 user1 group1    5 Mar 20 02:59 soft_link1 ->
file1

4898832 lrwxrwxrwx  1 user1 group1   10 Mar 20 03:00 soft_link2 ->
dir2/file6

4947973 -rw-r--r--  2 user1 group1   26 Mar 20 03:01 dir4

4947973 -rw-r--r--  2 user1 group1   26 Mar 20 03:01 hard_link
```

Echo – is used to output statements/variable values

```
$ echo "hello"

hello

$ echo "123... "

123...
```

Read – is to accept inputs from the user

```
$ read a

123
```

The above statement means variable “a” holds the number “123” as a=123

The “echo” command is also used to print the values assigned to variables

```
$ echo $a
```

Redirection

“>” is the redirection operator when used with files while “>>” is the redirection append operator

```
$ echo "this is file2" > file2

$ more file2

this is file2

$ echo "this is 2nd line in file2" >> file2

$ more file2

this is file2

this is 2nd line in file2
```

Environmental Variables

These are the default variables most of which are automatically set, while the rest are included in the profile file of the user account. env displays all the environment variables

```
$ env
```

Global Variables

These are initialized with an export keyword, also included in the profile file

```
$ export a=10
```

Aliases

Alias is a user understandable name given to one / combination of commands

```
$ alias ll='ls -l'
```

```
$ ll
total 8
drwxr-----  2 user1 group1 4096 Mar 20 05:01 dir1
drwxr-xr-x  3 user1 group1 4096 Mar 20 05:01 dir2
-rw-r--r--  1 user1 group1    0 Mar 20 05:01 file1
-rwxr-xr-x  1 user1 group1    0 Mar 20 05:01 file2
-rw-r--r--  1 user1 group1    0 Mar 29 23:49 file3

$ alias
alias l='ls'
alias ll='ls -l'
$ unalias ll
```

Profile File

A profile file is the first script that gets executed as soon as you login to a unix host

Type of Shell	Name of profile file
Bourne (sh)	.profile
Bourne again (bash)	.bash_profile
Korn (ksh)	.profile
C (csh)	.login
TC (tcsh)	.login

Grave Operator

Used to store the standard the output of a command in an enviroment variable. (`)

```
$ d=`date`
$ echo $d
Thu Oct 14 02:35:27 PDT 2010
$ d=`pwd`
$ echo $d
/home/user1
```

expr – Expr (command) command is used for numeric computation.
The operators + (add), -(subtract), *(multiply), /(divide), (remainder) are allowed. Calculation are performed in order of normal numeric precedence.

```
      $ a=4
      $ b=5

$ expr $a + $b
9

Or
$ c=`expr $a \+ $b`
$ echo $c
9
```

Logical operators

-eq : evaluates if two numbers are equal

-ne : not equal

-gt : greater than

-lt : less than

-le : less than or equal to

-ge : greater than or equal to

! : negates the value of another operator

-o / || : OR operation

-a / && : AND operation

Conditional constructs

if..then..else..fi

```
if [ condition ]
then
     command(s) 1
else
     command(s) 2
fi


a=2
if  [ "$a" = 2 ]
then
```

```
echo "this is 2 "
else
echo "this is not 2 "
fi
```

if..then.elif..then..else..fi

```
if [ condition 1 ]
then

     command(s) 1
elif [ condition 2 ]
then
     command(s) 2
else
     command(s) 3
fi

a=3

if [ "$a" -ne 5 ]
then
echo "this is not 5"
elif [ "$a" -lt 5 ]
then
echo "this is less than 5"
else
```

```
echo "this is greater than 5"
fi
```

for..do..done

```
for  variable in value1 value2 value3...valueN
do
     command(s)
done

for i in 1 2 3
do
     echo $i
done
```

while..do..done

```
while [ condition ]
do
     command(s)
done

i=0
while [ "$i" -lt 10 ]
do
     echo "$i"

     i=`expr $i + 1`
done
```

until..do..done

```
until [ condition ]
do
      command(s)
done


i=10
until [ "$i" -lt 1 ]
do
      echo "$i"
       i=`expr $i - 1`
done
```

case..esac

```
case value in

      choice1)
            command(s)            ;;
      choice2)
            command(s)            ;;
      choice3)
            command(s)            ;;
             *)
            command(s)            ;;
esac
```

```
$ a=4
$ b=5
$ c=5
$ d=4
$ if [ "$a" -lt "$b" -a "$c" -gt "$d" ]
then
     echo "true"
else
     echo "false"
fi
true

$ if [[ "$a" -lt "$b" && "$c" -gt "$d" ]]
then
     echo "true"
else
     echo "false"
fi
true
```

Executing a script – you execute a script by calling the script with the shell binary you intend to use

```
ksh scipt.ksh
ksh script
```

Comments in scripts

 # – Single line comments

<<keyword – Multi line comments

keyword

```
$ more one
#echo "this is 1"
echo "this is 2"
echo "this is 3"
echo "this is 4"
echo "this is 5"

$ ksh one
this is 2
this is 3
this is 4
this is 5

$ more one
echo "this is 1"
<<comment
echo "this is 2"
echo "this is 3"
echo "this is 4"
comment
echo "this is 5"
```

```
$ ksh one

this is 1

this is 5
```

#!/bin/ksh

This tells the interpreter what type of shell commands to expect further, Also, this will initialize (spawn) a new process with that shell as the default shell

Passing arguments to a shell script

```
$ ksh script1 param1

# cat script1

echo "the first parameter is $1"


# ksh script1 param1

the first parameter is param1


# ksh script1 1 2 3 4 5

the first parameter is 1

the second parameter is 2

the third parameter is 3

the fourth parameter is 4

the fifth parameter is 5
```

$# The number of positional parameters

$0 The name of the command/script being executed

$@ The list of positional parameters

$* same as $@, except when enclosed in double quotes

$? The success of last executed command

```
# cat script1
echo "the first parameter is $1"
echo "the second parameter is $2"
echo "the third parameter is $3"
echo "number of parameters passed are : $#"
echo "My name is : $0"
```

```
# ksh script1 one two three
the first parameter is one
the second parameter is two
the third parameter is three
number of parameters passed are : 3
My name is : script1
```

Shift operator - Using shift $1 becomes the source string and other arguments are shifted. $2 is shifted to $1, $3to $2 and so on

```
while [ "$#" -gt 0 ]
do
echo "this is $1"
shift
sleep 1
done
```