CSE333 LAB

Linux Commands & Shell Programming

12.10.2021

Zuhal Altuntaş
zuhal.altuntas@marmara.edu.tr

Fall 2021

[Slides by Dr. Sanem Arslan Yılmaz]

Lab Information

- Teaching Assistant:
- Zuhal Altuntaş zuhal.altuntas@marmara.edu.tr
  -
- Lab Sections : (Online)
  - Section I : Tuesday 14:00-16:00
  - Section II : Wednesday 14:00-16:00

- If you have any problem, please contact!
  - Office Hours: Friday 11:30-12:30
  - You can also make an assignment via email.

- Please subscribe and follow the mail group!
  - cse333list@googlegroups.com

Overview

- Course web page:

https://mimoza.marmara.edu.tr/~zuhal.altuntas/

- Lab content is not similar with lecture content.
  - 70% attendance is mandatory!
  - Attend to all lectures/labs on time !!!

- There will be 3 Programming Projects
  - Get a Linux distrubition as soon as possible! (Fedora, Ubuntu, etc.)
  - The weights of programming projects may vary.
  - *No late homework will be accepted.*

Policy on Projects:

- All projects will be done within a group.
- You will select your group and the group will not be changed throughout the semester.
- It is not acceptable of a partner team to work with other teams.
- It is NOT acceptable to copy solutions from other students.
- It is NOT acceptable to copy (or start your) solutions from Web.
- In case of any forms of cheating or copying, the penalties will be severe.
- Both Giver and Receiver are equally culpable and suffer equal penalties.

Lab Content:

- Overview on Linux and Shell Programming
- Process Management in Linux
- Interprocess Communication Techniques in Linux
- POSIX Threads and Thread Synchronization
- File System in Linux & Advanced File Handling

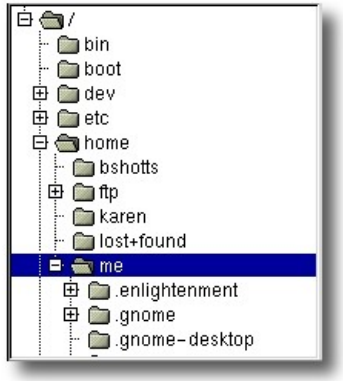
Linux Commands

- What is the «Shell»?
  - A program that takes commands from the keyboard and gives them to the operating system to perform.
  - A program called bash (Bourne Again SHell) acts as the shell program.

- What is the «Terminal»?
  - It’s a program called a *terminal emulator*.
  - This is a program that opens a window and lets you interact with the shell.

File System Organization

- The files on a Linux system are arranged in a *hierarchical directory structure*.
- The first directory (called folders in other systems) in the file system is called the *root directory*.



- One important difference between the legacy operating system and Linux is that:
  - Linux does not employ the concept of drive letters.
  - While drive letters split the file system into a series of different trees (one for each drive), Linux always has a single tree.
  - Different storage devices may contain different branches of the tree, but there is always a single tree.

Linux Commands

- pwd - (Print Working Directory)
  - The directory you are standing in is called the *working directory*.
  - To find the name of the working directory, use the pwd command.
  - When you first log on to a Linux system, the working directory is set to your *home directory*.
    - /home/your_user_name

Linux Commands

- cd - (Change Directory)
  - To change your working directory you use the cd command.
  - Type cd followed by the *pathname* of the desired working directory.
  - *Syntax:*
  - cd *pathname*

Linux Commands

- cd - (Change Directory)
  - To change your working directory you use the cd command.
  - Type cd followed by the *pathname* of the desired working directory.
  - *Syntax:*
  - cd *pathname*

  - *Absolute pathnames*: begins with the root directory and follows the tree branch by branch until the path to the desired directory or file is completed.
  - [sanem@localhost ~]$ cd /home/sanem/Desktop
    -

Linux Commands

- cd - (Change Directory)
  - To change your working directory you use the cd command.
  - Type cd followed by the *pathname* of the desired working directory.
  - *Syntax:*
  - cd *pathname*

  - *Absolute pathnames*: begins with the root directory and follows the tree branch by branch until the path to the desired directory or file is completed.
  - [sanem@localhost ~]$ cd /home/sanem/Desktop
    -
  - *Relative pathnames*: starts from the working directory and leads to its destination.
    - [sanem@localhost Desktop]$ cd New_Dir

Linux Commands

- cd - (Change Directory)
  - To change your working directory you use the cd command.
  - Type cd followed by the *pathname* of the desired working directory.
  - *Syntax:*
  - cd *pathname*
  - *Absolute pathnames*: begins with the root directory and follows the tree branch by branch until the path to the desired directory or file is completed.
  - [sanem@localhost ~]$ cd /home/sanem/Desktop
    -
  - *Relative pathnames*: starts from the working directory and leads to its destination.
    - [sanem@localhost Desktop]$ cd New_Dir

Special notations:

. – current directory

.. – parent directory

/ – root

Linux Commands

- ls - (List)
  - To list the files in the working directory, use the ls command.

Most commands operate like this:

command -options arguments

Command	**Result**
ls	List the files in the working directory
ls /bin	List the files in the /bin directory (or any other directory you care to specify)
ls -l	List the files in the working directory in long format
ls -l /etc /bin	List the files in the /bin directory and the /etc directory in long format
ls -al	List all files (including hidden files) in the working directory in long format

A Closer Look At Long Format

- If you use the -l option with ls you will get a file listing that contains a wealth of information about the files being listed.

```
-rw-------   1 bshotts  bshotts       576 Apr 17  1998 weather.txt
drwxr-xr-x   6 bshotts  bshotts      1024 Oct  9  1999 web_page
-rw-rw-r--   1 bshotts  bshotts    276480 Feb 11 20:41 web_site.tar
-rw-------   1 bshotts  bshotts      5743 Dec 16  1998 xmas_file.txt

----------     -------  -------  -------- ------------ -------------
    |             |        |         |         |             |
    |             |        |         |         |        File Name
    |             |        |         |         |
    |             |        |         |         +---  Modification Time
    |             |        |         |
    |             |        |         +-------------   Size (in bytes)
    |             |        |
    |             |        +-----------------------       Group
    |             |
    |             +-------------------------------       Owner
    |
    +------------------------------------------------   File Permissions
```

Manipulating Files

- cp - (Copy)
  - Copy files and directories
- mv - (Move)
  - Move or rename files and directories
- rm- (Remove)
  - Remove files and directories
- mkdir- (Make Directory)
  - Create directories
- rmdir- (Remove Directory)
  - Remove directories

Wildcards

- Wildcards allow you to select filenames based on patterns of characters.

* - Matches any characters

? - Matches any single character

[characters] - Matches any character that is a member of the set characters.

[!characters] - Matches any character that is not a member of the set characters

Pattern	**Matches**
*	All filenames
a*	All filenames that begin with the character "a"
b*.txt	All filenames that begin with the character "b" and end with the characters ".txt"
Data???	Any filename that begins with the characters "Data" followed by exactly 3 more characters
[abc]*	Any filename that begins with "a" or "b" or "c" followed by any other characters

cp - Copy

Syntax:
cp Source Destination

Command	**Result**
cp *file1* *file2*	Copies the contents of *file1* into *file2*. If *file2* does not exist, it is created; **otherwise, *file2* is silently overwritten with the contents of *file1*.**
cp -i *file1* *file2*	Like above however, since the "-i" (interactive) option is specified, if *file2* exists, the user is prompted before it is overwritten with the contents of *file1*.
cp *file1* *dir1*	Copy the contents of *file1* (into a file named *file1*) inside of directory *dir1*.
cp -R *dir1* *dir2*	Copy the contents of the directory *dir1*. If directory *dir2* does not exist, it is created. Otherwise, it creates a directory named *dir1* within directory *dir2*.

mv - Move

Syntax:
mv Source Destination

Command	**Result**
mv *file1* *file2*	If *file2* does not exist, then *file1* is renamed *file2*. **If *file2* exists, its contents are silently replaced with the contents of *file1*.**
mv -i *file1* *file2*	Like above however, since the "-i" (interactive) option is specified, if *file2* exists, the user is prompted before it is overwritten with the contents of *file1*.
mv *file1* *file2* *file3* *dir1*	The files *file1, file2, file3* are moved to directory *dir1*. If *dir1* does not exist, mv will exit with an error.
mv *dir1* *dir2*	If *dir2* does not exist, then *dir1* is renamed *dir2*. If *dir2* exists, the directory *dir1* is moved within directory *dir2*.

rm - Remove

Command	**Result**
rm *file1*	Delete *file1*
rm *file1* *file2*	Delete *file1* and *file2*.
rm -i *file1* *file2*	Like above however, since the "-i" (interactive) option is specified, the user is prompted before each file is deleted.
rm -r *dir1* *dir2*	Directories *dir1* and *dir2* are deleted along with all of their contents.

rm - Remove

Be careful with rm!

Command	**Result**
rm *file1*	Delete *file1*
rm *file1* *file2*	Delete *file1* and *file2*.
rm -i *file1* *file2*	Like above however, since the "-i" (interactive) option is specified, the user is prompted before each file is deleted.
rm -r *dir1* *dir2*	Directories *dir1* and *dir2* are deleted along with all of their contents.

rm - Remove Be careful with rm!

Command	**Result**
rm *file1*	Delete *file1*
rm *file1* *file2*	Delete *file1* and *file2*.
rm -i *file1* *file2*	Like above however, since the "-i" (interactive) option is specified, the user is prompted before each file is deleted.
rm -r *dir1* *dir2*	Directories *dir1* and *dir2* are deleted along with all of their contents.

- mkdir- (Make Directory)
  - mkdir dirName

rm - Remove

Be careful with rm!

Command	**Result**
rm *file1*	Delete *file1*
rm *file1* *file2*	Delete *file1* and *file2*.
rm -i *file1* *file2*	Like above however, since the "-i" (interactive) option is specified, the user is prompted before each file is deleted.
rm -r *dir1* *dir2*	Directories *dir1* and *dir2* are deleted along with all of their contents.

- mkdir- (Make Directory)
  - mkdir dirName
- rmdir- (Remove Directory)
  - rmdir dirName

Using Commands With Wildcards..

Command	**Result**
cp *.txt text_files	
mv my_dir ../*.bak my_new_dir	
rm *~	

Using Commands With Wildcards..

Command	**Result**
cp *.txt text_files	Copy all files in the current working directory with names ending with the characters ".txt" to an existing directory named *text_files*.
mv my_dir ../*.bak my_new_dir	
rm *~	

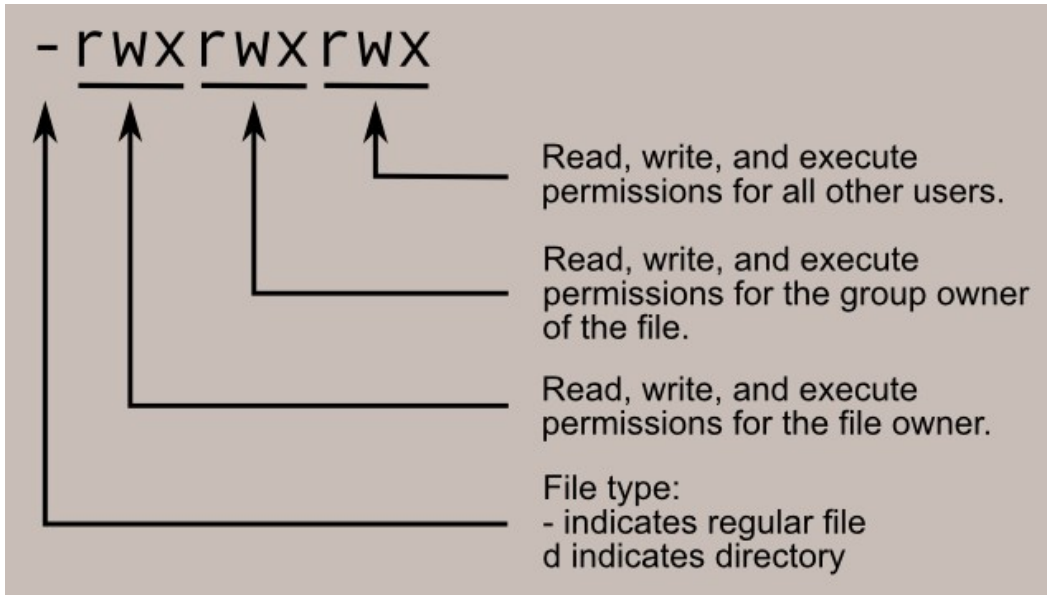
Using Commands With Wildcards..

Command	**Result**
cp *.txt text_files	Copy all files in the current working directory with names ending with the characters ".txt" to an existing directory named *text_files* .
mv my_dir ../*.bak my_new_dir	Move the subdirectory *my_dir* and all the files ending in ".bak" in the current working directory's parent directory to an existing directory named *my_new_dir* .
rm *~	

Using Commands With Wildcards..

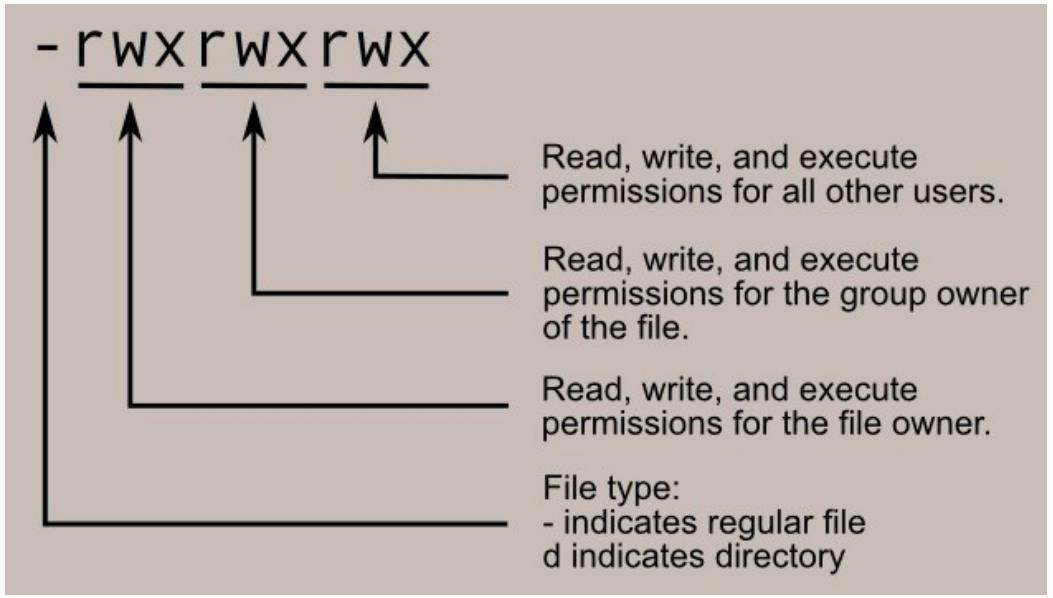
Command	**Result**
cp *.txt text_files	Copy all files in the current working directory with names ending with the characters ".txt" to an existing directory named *text_files* .
mv my_dir ../*.bak my_new_dir	Move the subdirectory *my_dir* and all the files ending in ".bak" in the current working directory's parent directory to an existing directory named *my_new_dir* .
rm *~	Delete all files in the current working directory that end with the character "~".

File Permissions & chmod



chmod - change the permissions of a file.
Syntax:
chmod *Permissions* *FileName*

File Permissions & chmod

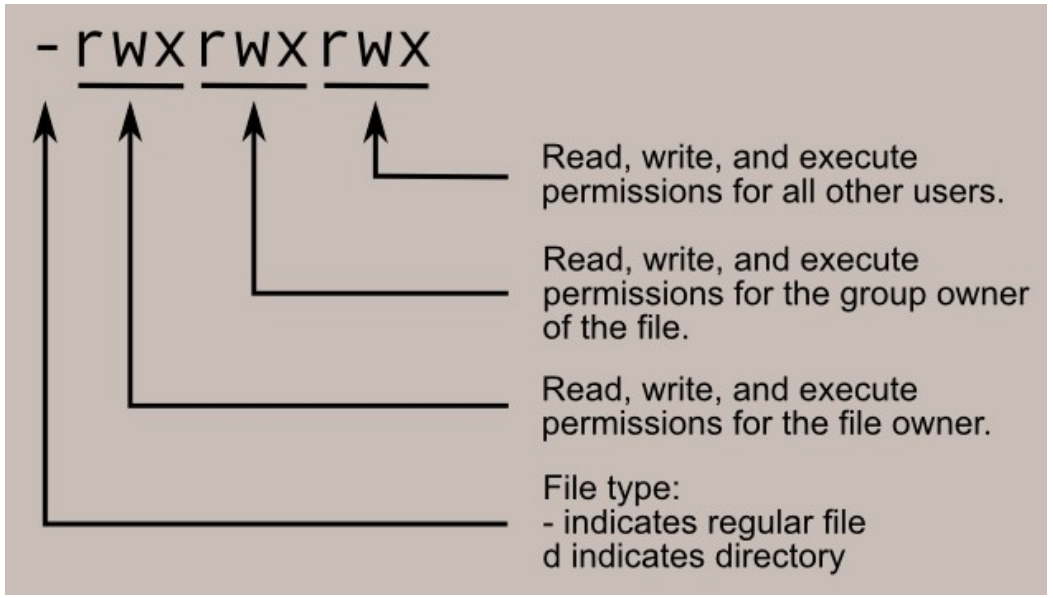


chmod - change the permissions of a file.
Syntax:
chmod *Permissions* *FileName*

```
rwx rwx rwx = 111 111 111
rw-  rw- rw-  = 110 110 110
rwx  ---  ---   = 111 000 000

rwx = 111 in binary = 7
rw- = 110 in binary = 6
r-x = 101 in binary = 5
r-- = 100 in binary = 4
```

File Permissions & chmod



chmod - change the permissions of a file.
Syntax:
chmod *Permissions* *FileName*

```
rwx rwx rwx = 111 111 111
rw-  rw- rw-  = 110 110 110
rwx  ---  ---   = 111 000 000

rwx = 111 in binary = 7
rw- = 110 in binary = 6
r-x = 101 in binary = 5
r-- = 100 in binary = 4
```

chmod 765 a.txt

Some useful commands..

- **`man`** - Display an on-line command reference
  - man *ls*
- **`clear`** - Clear the terminal screen.
- **`find`** - Search for files in a directory hierarchy
  - find *where-to-look criteria what-to-do*
  - *find / -name *.jpg*

I/O Redirection

- Standard Output
  - To redirect standard output to a file, the ">" character is used.
    - ls > file_list.txt
  - Each time the command above is repeated, file_list.txt is overwritten from the beginning with the output of the command ls.

I/O Redirection

- Standard Output
  - To redirect standard output to a file, the ">" character is used.
    - ls > file_list.txt
  - Each time the command above is repeated, file_list.txt is overwritten from the beginning with the output of the command ls.
  - If you want the new results to be *appended* to the file instead, use ">>".
    - ls >> file_list.txt

I/O Redirection

- Standard Output
  - To redirect standard output to a file, the ">" character is used.
    - ls > file_list.txt
  - Each time the command above is repeated, file_list.txt is overwritten from the beginning with the output of the command ls.
  - If you want the new results to be *appended* to the file instead, use ">>".
    - ls >> file_list.txt

- Standard Input
  - To redirect standard input from a file instead of the keyboard, the "<" character is used.
    - sort < file_list.txt
    - **`sort`** command is used for sorting the contents of file_list.txt.

I/O Redirection

- Standard Output
  - To redirect standard output to a file, the ">" character is used.
    - ls > file_list.txt
  - Each time the command above is repeated, file_list.txt is overwritten from the beginning with the output of the command ls.
  - If you want the new results to be *appended* to the file instead, use ">>".
    - ls >> file_list.txt
- Standard Input
  - To redirect standard input from a file instead of the keyboard, the "<" character is used.
    - sort < file_list.txt
    - **`sort`** command is used for sorting the contents of file_list.txt.
- We can redirect both standart input and outputs :
  - sort < file_list.txt > sorted_file_list.txt

Pipelines

- The standard output of one *command* is fed into the standard input of another with *pipelines.*
  - ls -l | sort
  - 

Pipelines

- The standard output of one *command* is fed into the standard input of another with *pipelines.*
  - ls -l | sort → ls -l > file_list.txt
  - sort < file_list.txt

Pipelines

- The standard output of one *command* is fed into the standard input of another with *pipelines.*
  - ls -l | sort → ls -l > file_list.txt
  - sort < file_list.txt
  - ls -l | sort | wc -l

wc - print the number of bytes, words, and lines in files. -l option counts the lines!

Shell Programming

- What Are Shell Scripts?
  - A shell script is a file containing a series of commands.
- To successfully write a shell script, you have to do:
  - Open an editor like *gedit.*
  - Write down any commands to your script.
  - Give permissions to execute it.

Example 1

```
#
# My first shell script
#
clear
echo "Knowledge is
Power"
```

echo command simply prints its arguments on the display.

Example 1

```
#
# My first shell script
#
clear
echo "Knowledge is
Power"
```

echo command simply prints its arguments on the display.

Give permissions as:

```
$ chmod   755    Example1.sh
OR
$ chmod   +x    Example1.sh
```

Example 1

```
#
# My first shell script
#
clear
echo "Knowledge is
Power"
```

echo command simply prints its arguments on the display.

Give permissions as:

```
$ chmod   755    Example1.sh
OR
$ chmod   +x    Example1.sh
```

Execute script as:

```
$ ./Example1.sh
```

Example 2

```
#
#
# Script to print user information who currently login , current date & time
#
clear
echo "Hello $USER"
echo "Today is  ";date
echo "Number of user login : " ; who | wc -l
echo "Calendar"
cal
exit 0
```

Example 2

```
#
#
# Script to print user information who currently login , current date & time
#
clear
echo "Hello $USER"
echo "Today is  ";date
echo "Number of user login : " ; who | wc -l
echo "Calendar"
cal
exit 0
```

Execute script as:

```
$  chmod 755 Example2.sh
$  ./Example2.sh
```

Variables in Shell

In Linux (Shell), there are two types of variable:,

1. **System variables** - Created and maintained by Linux itself. This type of variable defined in CAPITAL LETTERS.
2. **User defined variables (UDV)** - Created and maintained by user. This type of variable defined in lower letters.

Variables in Shell

In Linux (Shell), there are two types of variable:,

1. **System variables** - Created and maintained by Linux itself. This type of variable defined in CAPITAL LETTERS.
2. **User defined variables (UDV)** - Created and maintained by user. This type of variable defined in lower letters.

System Variable	**Meaning**
BASH=/bin/bash	Our shell name
HOME=/home/std	Our home directory
PWD=/home/std/Common	Our current working directory
SHELL=/bin/bash	Our shell name
USERNAME=std	User name who is currently login to this PC

Variables in Shell

In Linux (Shell), there are two types of variable:,

1. **System variables** - Created and maintained by Linux itself. This type of variable defined in CAPITAL LETTERS.
2. **User defined variables (UDV)** - Created and maintained by user. This type of variable defined in lower letters.

System Variable	**Meaning**
BASH=/bin/bash	Our shell name
HOME=/home/std	Our home directory
PWD=/home/std/Common	Our current working directory
SHELL=/bin/bash	Our shell name
USERNAME=std	User name who is currently login to this PC

- You can print any of the above variables contains as follows:
- $ echo $USERNAME
  $ echo $HOME

User Defined Variables (UDV)

To define UDV use following syntax

Syntax:

- variable name=value

Example:

```
$ no=10    # this is ok
$ 10=no    # Error, NOT Ok, Value must be on right side of = sign.
```

To define variable called 'vech' having value Bus

```
$ vech=Bus
```

To define variable called n having value 10

```
$ n=10
```

Rules for Naming variable name

1. Variable name must begin with Alphanumeric character or underscore character (_), followed by one or more Alphanumeric character.
2. Don't put spaces on either side of the equal sign when assigning value to variable.
3. Variables are case-sensitive, just like filename in Linux.
4. You can define NULL variable as follows:

```
$ vech=""
```

How to print or access value of UDV

- Define variable *vech* and *n* as follows:

```
$ vech=Bus
$ n=10
```

- To print contains of variables:

```
$ echo $vech
$ echo $n
```

Example 3

```
#
#
# Script to test MY knowledge about variables!
#
myname=Vivek
myos = TroubleOS
myno=5
echo "My name is $myname "
echo "My OS is $myos "
echo "My number is myno, can you see this number?"
```

Example 3

```
#
#
# Script to test MY knowledge about variables!
#
myname=Vivek
myos = TroubleOS
myno=5
echo "My name is $myname "
echo "My OS is $myos "
echo "My number is myno, can you see this number?"
```

Execute script as:

```
$  chmod 755 Example3.sh
$  ./Example3.sh
```

Shell Arithmetic

Syntax:

- expr op1 math-operator op2

- *Examples:*

```
$ expr 1 + 3
$ expr 2 – 1
$ expr 10 / 2
$ expr 20 % 3
$ expr 10 \* 3
$ echo `expr 6 + 3`      # Use before and after expr keyword
```

` (back quote) sign

not the (single quote i.e. ') sign.

Exit Status

- By default in Linux if particular command/shell script is executed, it return two type of values which is used to see whether command or shell script executed is successful or not.
  1. If return *value is zero* (0), command is successful.
  2. If return *value is nonzero*, command is not successful or some sort of error executing command/shell script.

Exit Status

- By default in Linux if particular command/shell script is executed, it return two type of values which is used to see whether command or shell script executed is successful or not.
  1. If return *value is zero* (0), command is successful.
  2. If return *value is nonzero*, command is not successful or some sort of error executing command/shell script.

- How to find out exit status of command or shell script?
  - Use **$?** special variable of shell.

Exit Status

- By default in Linux if particular command/shell script is executed, it return two type of values which is used to see whether command or shell script executed is successful or not.
  1. If return *value is zero* (0), command is successful.
  2. If return *value is nonzero*, command is not successful or some sort of error executing command/shell script.

- How to find out exit status of command or shell script?
  - Use **$?** special variable of shell.

- Examples:
- **$ ls**
  $ echo $? **#** It will print 0 to indicate command is successful.
  $ rm unknownfile
  $ echo $? **#** It will print nonzero value to indicate error.

The read Statement

- Use to get input (data from user) from keyboard and store (data) to variable.
- *Syntax:*

read variable1, variable2,...variableN

The read Statement

- Use to get input (data from user) from keyboard and store (data) to variable.
- *Syntax:*

read variable1, variable2,...variableN

Example4:

```
#
# Script to read your name from key-board
#
echo "Your first name please: "
read fname
echo "Hello $fname, Lets be friend!"
```

The read Statement

- Use to get input (data from user) from keyboard and store (data) to variable.
- *Syntax:*

read variable1, variable2,...variableN

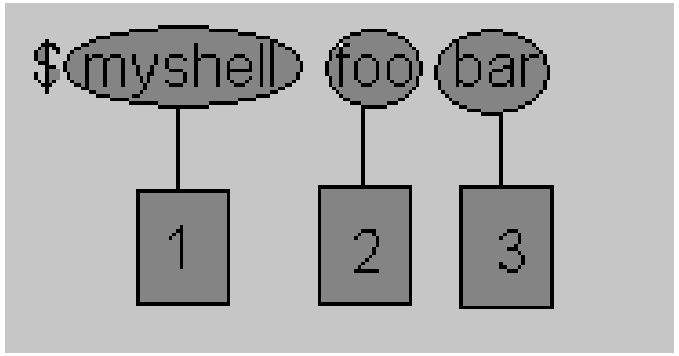
Example4:

```
#
# Script to read your name from key-board
#
echo "Your first name please: "
read fname
echo "Hello $fname, Lets be friend!"
```

Execute script as:

```
$  chmod 755 Example4.sh
$  ./Example4.sh
```

Command Line Arguments



Shell Script name i.e. myshell (myshell is $0)
First command line argument passed to myshell i.e. foo (foo is $1)
Second command line argument passed to myshell i.e. bar (bar it is $2)

Shell Script Name ($0),
No. of Arguments (i.e. $#),
Actual argument (i.e. $1,$2 etc),
$* -> `$1,$2...$9`

Example 5

```
#
# Script that demos, command line args
#
echo "Total number of command line argument are $#"
echo "$0 is script name"
echo "$1 is first argument"
echo "$2 is second argument"
echo "All of them are :- $* or $@"
```

Example 5

```
#
# Script that demos, command line args
#
echo "Total number of command line argument are $#"
echo "$0 is script name"
echo "$1 is first argument"
echo "$2 is second argument"
echo "All of them are :- $* or $@"
```

Execute script as:

```
$  chmod 755 Example5.sh
$  ./Example5.sh Hello World
```

Conditions in Shell

- **if Condition**

Syntax:

```
if condition
then
    command1 if condition is true or if exit status of condition is 0 (zero)
    ...
    ...
fi
```

Conditions in Shell

- **if Condition**

Syntax:

```
if condition
then
    command1 if condition is true or if exit status of condition is 0 (zero)
    ...
    ...
fi
```

- **if…else…fi Condition**

Syntax:

```
if condition
then
    condition is zero (true - 0)
    execute all commands up to else statement
else
    if condition is not true then
    execute all commands up to fi
fi
```

Conditions in Shell

- **if Condition**

Syntax:

```
if condition
then
    command1 if condition is true or if exit status of condition is 0 (zero)
    ...
    ...
fi
```

- **if…else…fi Condition**

Syntax:

```
if condition
then
    condition is zero (true - 0)
    execute all commands up to else statement
else
    if condition is not true then
    execute all commands up to fi
fi
```

- **test command or [expr]**
- *Syntax:*

  test expression OR [expression]

Comparion in Shell

For Mathematics, use following operator in Shell Script

Mathematical Operator in Shell Script	**Meaning**	**For test statement with if command**	**For [expr] statement with if command**
-eq	is equal to	if test 5 -eq 6	if [5 -eq 6]
-ne	is not equal to	if test 5 -ne 6	if [5 -ne 6]
-lt	is less than	if test 5 -lt 6	if [5 -lt 6]
-le	is less than or equal to	if test 5 -le 6	if [5 -le 6]
-gt	is greater than	if test 5 -gt 6	if [5 -gt 6]
-ge	is greater than or equal to	if test 5 -ge 6	if [5 -ge 6]

Comparion in Shell

For Mathematics, use following operator in Shell Script

Mathematical Operator in Shell Script	**Meaning**	**For test statement with if command**	**For [expr] statement with if command**
-eq	is equal to	if test 5 -eq 6	if [5 -eq 6]
-ne	is not equal to	if test 5 -ne 6	if [5 -ne 6]
-lt	is less than	if test 5 -lt 6	if [5 -lt 6]
-le	is less than or equal to	if test 5 -le 6	if [5 -le 6]
-gt	is greater than	if test 5 -gt 6	if [5 -gt 6]
-ge	is greater than or equal to	if test 5 -ge 6	if [5 -ge 6]

For string Comparisons use

Operator	**Meaning**
string1 = string2	string1 is equal to string2
string1 != string2	string1 is NOT equal to string2
-n string1	string1 is NOT NULL and does exist
-z string1	string1 is NULL and does exist

Example 6:

```
#
# Script to see whether argument is positive or negative
#
if [ $# -eq 0 ]
then
	echo "$0 : You must give/supply one integers"
	exit 1
fi
if test $1 -gt 0
then
	echo "$1 number is positive"
else
	echo "$1 number is negative"
fi
```

Example 6:

```
#
# Script to see whether argument is positive or negative
#
if [ $# -eq 0 ]
then
	echo "$0 : You must give/supply one integers"
	exit 1
fi
if test $1 -gt 0
then
	echo "$1 number is positive"
else
	echo "$1 number is negative"
fi
```

Execute script as:

```
$  chmod 755 Example6.sh
$  ./Example6.sh  10
$  ./Example6.sh  -5
$  ./Example6.sh  Hello
```

Conditions in Shell

- **Multilevel if-then-else**

Syntax:

```
if condition
then
    condition is zero (true - 0)
    execute all commands up to elif statement
elif condition1
    condition1 is zero (true - 0)
    execute all commands up to else statement
else
    None of the conditions are true
    execute all commands up to fi
fi
```

Example 7:

```
#!/bin/sh
# Script to test if..elif...else
#
if [ $1 -gt 0 ];
then
  echo "$1 is positive"
elif [ $1 -lt 0 ]
then
  echo "$1 is negative"
elif [ $1 -eq 0 ]
then
  echo "$1 is zero"
else
  echo "Opps! $1 is not number, give number"
fi
```

Example 7:

```
#!/bin/sh
# Script to test if..elif...else
#
if [ $1 -gt 0 ];
then
  echo "$1 is positive"
elif [ $1 -lt 0 ]
then
  echo "$1 is negative"
elif [ $1 -eq 0 ]
then
  echo "$1 is zero"
else
  echo "Opps! $1 is not number, give number"
fi
```

Execute script as:

```
$  chmod 755 Example7.sh
$  ./Example7.sh  10
$  ./Example7.sh  -5
$  ./Example7.sh  0
$  ./Example7.sh  Hello
```

Loops in Shell Scripts

- for Loop

Syntax:

```
for { variable name } in { list }
do
    execute one for each item in the list until the list is not finished
    (And repeat all statement between do and done)
done
```

Loops in Shell Scripts

- for Loop

Syntax:

for { variable name } in { list }
do
execute one for each item in the list until the list is not finished
(And repeat all statement between do and done)
done

Example 8

```
#
# Script to test for loop
#
for i in 1 2 3 4 5
do
echo "Welcome $i times"
done
```

Loops in Shell Scripts

- Even you can use following syntax:

Syntax:

```
for (( expr1; expr2; expr3 ))
do
    repeat all statements between do and
    done until expr2 is TRUE
done
```

Loops in Shell Scripts

- Even you can use following syntax:

Syntax:

```
for (( expr1; expr2; expr3 ))
do
    repeat all statements between do and
    done until expr2 is TRUE
done
```

Example 9

```
#
# Script to test for loop
#
for ((  i = 0 ;  i <= 5;  i++  ))
do
  echo "Welcome $i times"
done
```

Loops in Shell Scripts

- while Loop

Syntax:

```
while [ condition ]
do
    command1
    command2
    ....
    ....
done
```

Loops in Shell Scripts

- while Loop

Syntax:

```
while [ condition ]
do
    command1
    command2
    ....
    ....
done
```

Example 10:

```
#
#Script to test while statement
#
#
if [ $# -eq 0 ]
then
   echo "Error - Number missing form command line argument"
   echo "Syntax : $0 number"
   echo " Use to print multiplication table for given number"
exit 1
fi
n=$1
i=1
while [ $i -le 10 ]
do
  echo "$n * $i = `expr $i \* $n` "
  i=`expr $i + 1`
done
```

The case Statement

Syntax:

```
case $variable-name in
    pattern1)    command
        ...
        ...
        command  ;;
    pattern2)    command
            ...
            ...
            command  ;;
    patternN)    command
            ...
            ...
            command  ;;
    *)           command
            ...
            ...
            command  ;;
esac
```

Example 11:

```
#
#
# Script to test case statement
#
#

if [ -z $1 ]
then
  rental="*** Unknown vehicle ***"
elif [ -n $1 ]
then
# otherwise make first arg as rental
  rental=$1
fi
case $rental in
   "car") echo "For $rental Rs.20 per k/m";;
   "van") echo "For $rental Rs.10 per k/m";;
   "jeep") echo "For $rental Rs.5 per k/m";;
   "bicycle") echo "For $rental 20 paisa per k/m";;
   *) echo "Sorry, I can not gat a $rental for you";;
esac
```

Example 11:

```
#
#
# Script to test case statement
#
#

if [ -z $1 ]
then
  rental="*** Unknown vehicle ***"
elif [ -n $1 ]
then
# otherwise make first arg as rental
  rental=$1
fi
case $rental in
   "car") echo "For $rental Rs.20 per k/m";;
   "van") echo "For $rental Rs.10 per k/m";;
   "jeep") echo "For $rental Rs.5 per k/m";;
   "bicycle") echo "For $rental 20 paisa per k/m";;
   *) echo "Sorry, I can not gat a $rental for you";;
esac
```

Execute script as:

```
$  chmod 755 Example11.sh
$  ./Example11.sh  car
$  ./Example11.sh  van
$  ./Example11.sh  jeep
$  ./Example11.sh  Maserati
```

More Examples:

- Menu.sh
- ArabicToRoman.sh