Shell basics Introduction

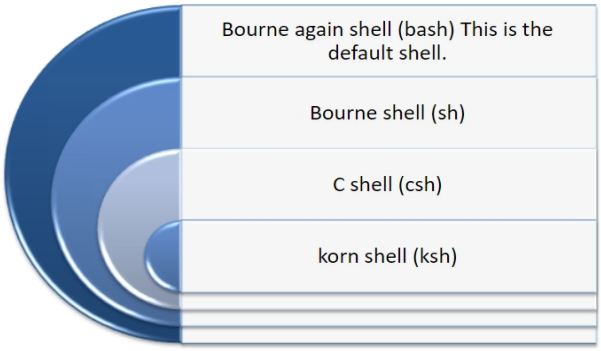
TEKNOTURF

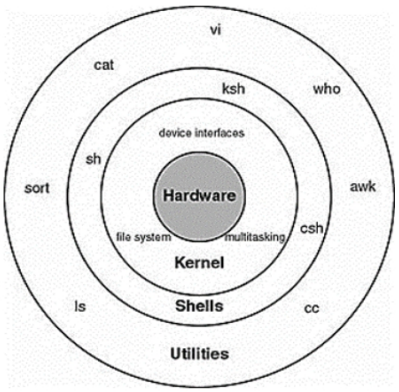
- Shell supports execution of scripts
- When a group of commands have to be executed regularly, they should be written in a file, which is the shell script or shell program and the script will be executed by the shell
- A shell program runs in interpretive mode. It is not compiled to a separate executable file. Each statement is loaded into memory when it is to be executed
- For the programmer's understandability, the script file may have extension as "sh"
  - For a good programming practice, the extension is followed
    - E.g testfile.sh

Shell

TEKNOTURF

Types of Shell





Working of Shell

TEKNOTURF

Comments

Single line comment

is used to comment a line in the script

Example: # echo "hai"

Multiple line comment

<< comment
- statements
- comment

Example: <<comment
- echo "this is"
- echo "test code"
- comment

Basic elements in Shell programming:

command grouping:

;

- executing multiple commands in a single line
- Example:

```
tsc@oracle:~
[tsc@oracle ~]$ date ; who
Tue Jul 14 11:56:54 IST 2009
tsc      pts/1        Jul 14 11:56 (192.168.40.146)
[tsc@oracle ~]$
```

This executes the date command 1st and then the who command.

Basic elements in Shell programming:

{ }

- used for grouping commands and executes in the current shell. The last statement in every group should be terminated with a semicolon
- There should be a blank space after the open braces and before the close braces
- This is used when one of the statements in the group needs to update the current environment

- Example:
  - $ { cd mydir ; pwd; }
    - /home/teknoturf/staff/tsc/mydir
- After executing this command, the user is placed in the mydir directory

Basic elements in Shell programming:

( )

- used for grouping commands in a single line and executing the grouped command in the sub-shell(child process)

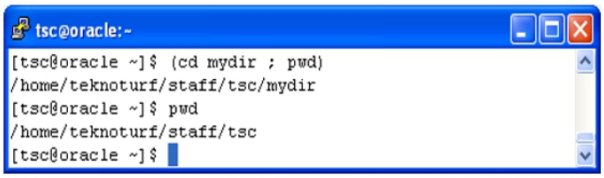
example:

- Without parenthesis, these commands execute in the same shell

```
tsc@oracle:~/mydir
[tsc@oracle ~]$ cd mydir ; pwd
/home/teknoturf/staff/tsc/mydir
[tsc@oracle mydir]$ pwd
/home/teknoturf/staff/tsc/mydir
[tsc@oracle mydir]$
```

Basic elements in Shell programming:

With parenthesis, these commands execute in the sub-shell. So once executed, the sub-shell(child process) is destroyed.



Basic elements in Shell programming:

Conditional execution:

- These are for logical testing

&& :

- This will test for "and" condition
- The second command will be executed only when the 1st command is successful

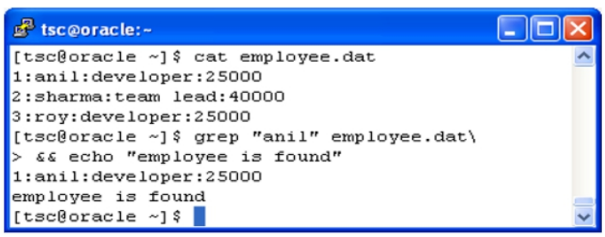
Syntax:

- command1 && command2

Basic elements in Shell programming:

&& example:

check whether the employee "anil" is present and if present, display his information



Basic elements in Shell programming:

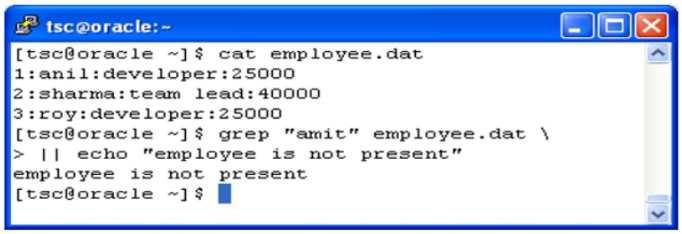
Conditional execution:

||

- This will test “or” condition
- The second command will be executed only when the 1st command fails
- **Syntax:**
  - **command1 || command2**

Basic elements in Shell programming:

Logical OR (||) example:



Basic elements in Shell programming:

Variables:

- Variables provide the ability to store and manipulate information within a shell program.
- The data is treated as string.

Types of variable:

- System or environment variable
- User defined variable

Basic elements in Shell programming:

System Variables

- These are standard variables which are always accessible
- These variables are declared in the environment area
- The shell provides the value for these variables
- These variables are used by the system to govern the environment
- The user is allowed to change the values of these variables

Basic elements in Shell programming:

Various system variables are

PS1, PS2	PATH	HOME	LOGNAME	SHELL	TERM
→ system prompts	→ path which the shell must search in order to execute any command or file	→ stores the default working directory of the user	→ login name	→ name of the default working shell	→ defines the name of the terminal on which the user is working

Basic elements in Shell programming:

user-defined variable:

- These are created, used and maintained by the user in shell programming

export:

- User defined variables are for the current shell
- To access the user defined variable in the child process, these variables should be placed in the environment area
- This is done by the command export

Example 1:

```
staff@MQSVR:~
[staff@MQSVR ~]$ age=32
[staff@MQSVR ~]$ echo $age
32
[staff@MQSVR ~]$ bash
[staff@MQSVR ~]$ echo $age

[staff@MQSVR ~]$
```

Example 2:

```
staff@MQSVR:~
[staff@MQSVR ~]$ gender=F
[staff@MQSVR ~]$ echo $gender
F
[staff@MQSVR ~]$ export gender
[staff@MQSVR ~]$ bash
[staff@MQSVR ~]$ echo $gender
F
[staff@MQSVR ~]$
```

Basic elements in Shell programming:

Assigning value to the variable:

= is used to assign value to the variable

Example 1: **$ name=tsc**

- name is the variable name
- tsc is the value
- no space between the variable name, assignment operator and the value.

Example 2: **$ name="Teknoturf school of computing"**

- Value should be enclosed within Double quotes when there is more than 1 word.

Basic elements in Shell programming:

Listing of variables:

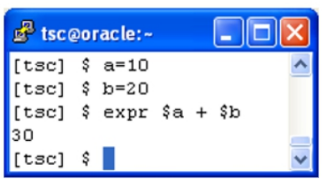
- $ env → displays all the system or environment variables
- $ set → displays all the variables available in the current shell
- $ echo $<variable-name> → displays the value of the specific variable

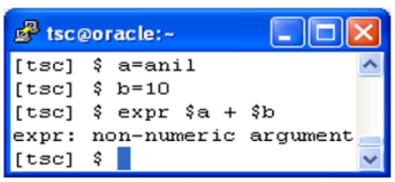
Basic elements in Shell programming:

Evaluate expression:

- To carry out arithmetic operations, **expr** command is used.
- **expr** can do only the integer operations.

Example:





Basic elements in Shell programming:

Test (or) [ ]

Used to check whether the statement is true or false

test returns true or false status

It works in 3 ways

- Numeric comparison
- String comparison
- Checks file's attributes

$? is used to display the status returned by the previous command

Operators should be surrounded by spaces

Basic elements in Shell programming:

Numeric comparison

OPERATOR	MEANING
-eq	Equal to
-ne	Not equal to
-gt	Greater than
-ge	Greater than or equal to
-lt	Less than
-le	Less than or equal to

Basic elements in Shell programming:

Numeric comparison Example:

$ x=5 ; y=10

$ test $x -eq $y ; echo $?

output : 1 (for not equal)

or

$ [$x -eq $y] ; echo $?

output : 1 (for not equal)

When the test condition returns true, $? prints 0

Basic elements in Shell programming:

TEKNOTURF

String comparison

OPERATOR	MEANING
S1=S2	Checks for equality
S1!=S2	Not equal
-n string	String is not null
-z string	String is null

Example:

$ employee1=Tom ; employee2=Amit

$ test $employee1 = $employee2 ; echo $?

output : 1 (represents not equal)

File Comparison

TEKNOTURF

operator	Meaning
•-e file	True if file exists.
•-f *file*	*file* exists and is a regular file
•-r *file*	*file* exists and is readable
•-w file	file exists and is writable
•-x file	file exists and is executable
•-d file	file exists and is a directory
•-s file	file exists and the size is greater than 0
•file1 -nt file2	file1 is newer than file2
•file2 -ot file2	file1 is older than file2
•file1 -ef file2	file1 is linked to file2

•(-nt , -ot , -ef are available only in bash and korn shell)

Basic elements in Shell programming:

TEKNOTURF

File comparison example:

- $ touch testfile
- $ ls -l testfile

  -rw-r--r-- 1 tsc staff 0 Jul 15 15:10 testfile
- $ [-f testfile] ; echo $?

  0 (it is an ordinary file)
- $ [-x testfile] ; echo $?

  1 (it is not executable)

Meta-character

A meta character is a character that has a special meaning to the Shell.

Type	Metacharacter
Filename substitution	? * [ ] [!]
I/O Redirection	> >> < <<
Process Execution	; ( ) & && ||
Quoting metacharacters	\ " " ' '
positional parameters	$1 to $9
Special characters	$0 $* $@ $# $! $$ $-

Metacharacter continued

Filename substitution metacharacter

- * → it is a wild card character, which represents none or any number of characters
- $ ls a* lists all files beginning with character 'a'
- ? → stands for 1 character
  - $ ls ?? lists all files whose file names are 2 character long
- [ ] any one character from the enclosed list
  - $ ls [kdgp]* lists all files whose 1st character is k , d , g or p
- [!] any one character except those enclosed in the list
  - $ ls [!d – m]* lists all files whose 1st character is anything other than the alphabet in the range d to m

Quoting Metacharacter

double quote:

- They allow *all* shell interpretations to take place inside them.
- Example:
  - $ name=tom
  - $ echo “The name is $name”
  - output : The name is tom

Quoting Metacharacter

single quote:

- anything enclosed in single quotes remains unchanged.
- Example:
  - $ name=tom
  - $ echo 'The name is $name'
  - output: The name is $name

Quoting Metacharacter

back quotes

- To execute command and replace the output in place of the command
- Example:
  - $ echo "Today is date"
    Output : Today is date
  - $ echo "Today is `date`"
    Output : Today is Wed Jul 15 13:18:30 IST 2009

Shell Redirection

- Any process that starts executing in the shell has 3 files which are given by the shell
- These files are called streams
- streams are
  - standard input – represents the input which is normally the keyboard
  - standard output – represents the output which is normally the monitor
  - standard error – represents the error which is normally the monitor
- These streams are assigned with numbers which are called file descriptors

Redirection

File Descriptor:

- a file descriptor is an abstract key for accessing a file
- a file descriptor is an integer value
- There are 3 standard file descriptors which every process should have

integer value	name
0	standard input(stdin)
1	standard output(stdout)
2	standard error(stderr)

Redirection

Redirection is the switching of a *standard stream* of data so that it comes from a source other than its default source or goes to some destination other than its default destination

Redirection is used to change the default stream to any other stream

Example: redirected to or from a file

Types of redirection

- Input redirection
- Output redirection
- Error redirection

Redirection

Input redirection:

- Input redirection makes the command to get the input from other devices apart from the standard input
- The symbol for input redirection is **<**

example:

- with standard input stream:

```
$ bc
2 + 3
5
[ctrl+d]
```

Redirection

TEKNOTURF

with input redirection: to avoid user interaction

- $ bc < inputfile (or) bc 0 < inputfile
  5
- note: the content of inputfile is 2 + 3
  0 is the file descriptor and is optional

How it works:

- On seeing the < , the shell opens the file, inputfile for reading
- unplugs the standard input stream from its default and assigns it to inputfile
- bc reads the contents from the file, manipulates it and displays the output to the screen

Redirection

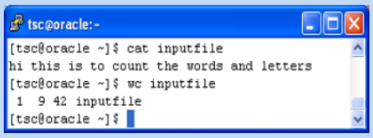
TEKNOTURF

Output redirection:

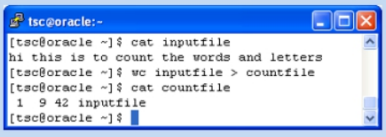
- The output of the command is normally displayed in the monitor which is the standard output stream
- The *output redirection operator* (>) can be used to redirect standard output from the monitor to any other device.

Example

with standard redirection



with output redirection



- 1> can also be used. 1 is the file descriptor for output stream
- The output instead of displaying in the monitor is being redirected to countfile

TEKNOTURF

How it works:

- On seeing >, the shell opens a new file named countfile if it doesn't exist or it overwrites the content, if the file already exists
- Unplugs the default standard output stream and assigns it to countfile
- command gets executed and the output is written to the countfile

Output redirection with append:

- symbol >>
- Shell opens the file and moves the cursor to the end of the file. No overwriting occurs

Shell Redirection

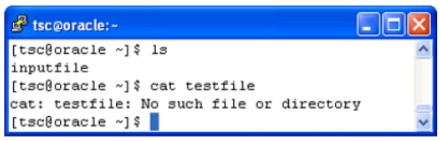
TEKNOTURF

Error redirection:

- The default standard error stream is the monitor.
- standard error can be redirected from its default to any other device by using the *standard error redirection operator*, 2>.
- 2> overwrites the file content and 2>> appends to the already existing file
- Both output and error redirection works the same.

Example:

with standard error



with error redirection