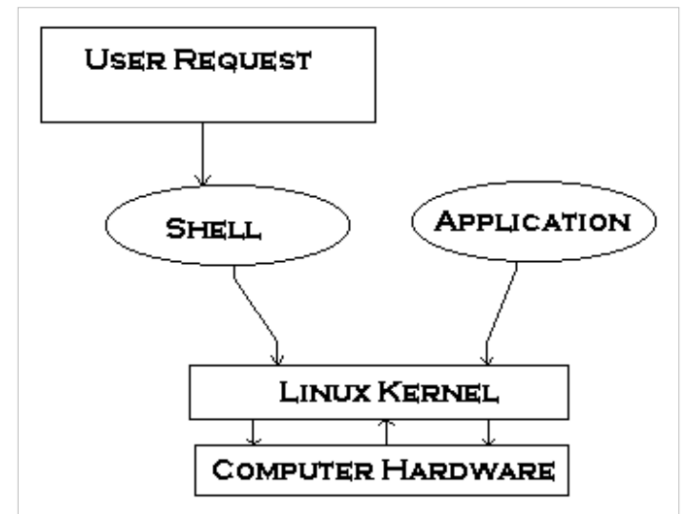
Bash Shell

Agenda

- What is a shell? A shell script?
- Introduction to `bash`
- Running Commands
- Shell Structures
- Applied Shell Programming

Introduction to Shell Programming

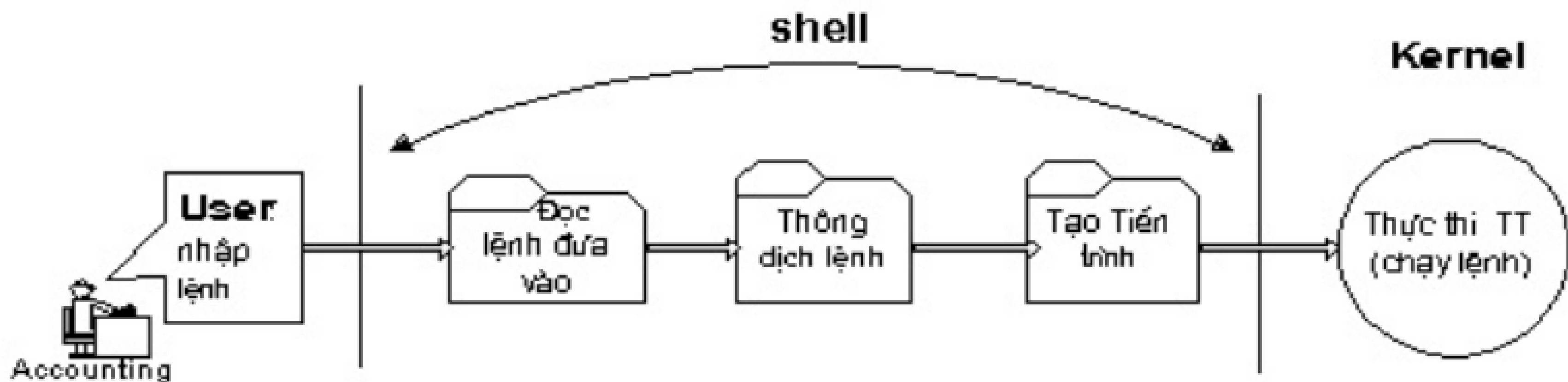
- Computer understand the language of 0's and 1's called binary language.
- In early days of computing, instruction are provided using binary language, which is difficult for all of us, to read and write. So in Os there is special program called Shell. Shell accepts your instruction or commands in English (mostly) and if its a valid command, it is passed to kernel.



Introduction to Shell Programming

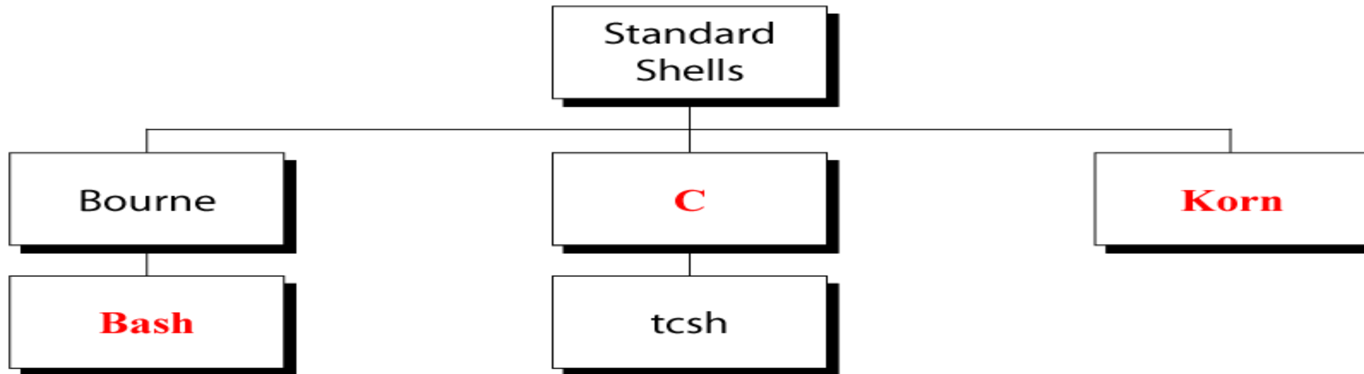
Shell is a user program or it's a environment provided for user interaction.

Shell is an command language interpreter that executes commands read from the standard input device (keyboard) or from a file.



UNIX

UNIX Command Interpreters



Linux has a variety of different shells:

Bourne shell (sh), C shell (csh), Korn shell (ksh), TC shell (tcsh), Bourne Again shell (bash).

Certainly the most popular shell is "bash". Bash is an sh-compatible shell

To find all available shells in your system type following command:

$ cat /etc/shells

UNIX

Programming or Scripting ?

No	Compiler	Interpreter
1	Compiler Takes **Entire** program as input	Interpreter Takes **Single** instruction as input .
2	Intermediate Object Code is **Generated**	**No** Intermediate Object Code is **Generated**
3	Conditional Control Statements are Executes **faster**	Conditional Control Statements are Executes **slower**
4	**Memory Requirement : More** (Since Object Code is Generated)	**Memory Requirement** is **Less**
5	Program need not be **compiled** every time	Every time higher level program is converted into lower level program
6	**Errors** are displayed after **entire program** is checked	**Errors** are displayed for **every instruction** interpreted (if any)

UNIX

What is Shell Script

Normally shells are interactive. It means shell accept command from you (via keyboard) and execute them.

But if you use command one by one (sequence of 'n' number of commands) , the you can store this sequence of command to text file and tell the shell to execute this text file instead of entering the commands. This is know as shell script.

Shell script defined as:
"Shell Script is series of command written in plain text file. Shell script is just like batch file is MS-DOS but have more power than the MS-DOS batch file."

Shell Program Structure

- A shell program contains high-level programming language features:
  - –Variables for storing data
  - –Decision-making control (e.g. if and case statements)
  - –Looping abilities (e.g. for and while loops)
  - –Function calls for modularity

UNIX

Steps to Create Shell Programs

- Specify shell to execute program
  - Script must begin with #! to identify shell to be executed

Examples:

```
#! /bin/sh            (defaults to bash)
#! /bin/bash
#! /bin/csh
#! /usr/bin/tcsh
```

- Make the shell program executable
  - Use the “chmod” command to make the program /script file executable

UNIX

How to write shell script

(1) Use any editor like vi or pico, gedit to write shell script.

(2) After writing shell script set execute permission for your script as follows

syntax: chmod permission your-script-name

$ chmod +x your-script-name OR $ chmod 755 your-script-name

This will set read write execute (7) permission for owner, for group and other permission is read and execute only (5).

(3) Execute your script as syntax:

bash your-script-name

sh your-script-name

./your-script-name

$ bash bar

$ sh bar

$ ./bar

- A Text File
- With Instructions
- Executable

UNIX

Basic Shell Programming

- A script is a file that contains
- shell commands
- data structure: variables
- control structure: sequence, decision, loop

Input
- prompting user
- command line arguments

Decision:
- if-then-else
- case

Repetition
- do-while, repeat-until
- for
- select

Functions

UNIX

What is a Shell Script? What To Do

```
pico hello.sh
#!/bin/sh
echo 'Hello, world'

% chmod +x hello.sh
% ./hello.sh

Hello, world
```

UNIX

What is a Shell Script? Executable

```
pico hello.sh
#!/bin/sh
echo ‘Hello, world’

% chmod +x hello.sh
% ./hello.sh

Hello, world
```

UNIX

What is a Shell Script? Running it

```
pico hello.sh
#!/bin/sh
echo 'Hello, world'

% chmod +x hello.sh
% ./hello.sh

Hello, world
```

UNIX

FORMATTING SHELL PROGRAMS

- Comments
- Start comment lines with a pound sign (#)
- Include comments to describe sections of your program
- Help you understand your program when you look at it later

UNIX

Single and Double Quote

- When assigning character data containing spaces or special characters, the data must be enclosed in either single or double quotes.

- Using double quotes to show a string of characters will allow any variables in the quotes to be resolved

```
$ var=“test string”
$ newvar=“Value of var is $var”
$ echo $newvar
Value of var is test string
```

- Using single quotes to show a string of characters will not allow variable resolution

```
$ var=’test string’
$ newvar=’Value of var is $var’
$ echo $newvar
Value of var is $var
```

UNIX

Command Substitution

- The backquote "`" is different from the single quote "´". It is used for command substitution: `command`

```
$ LIST=`ls`
$ echo $LIST
hello.sh read.sh
```

- We can perform the command substitution by means of $(command)

```
$ LIST=$(ls)
$ echo $LIST
hello.sh read.sh

$ rm $( find / -name "*.tmp" )

$ cat  backup.sh
#!/bin/bash
OF=myscript_directory_$(date +%Y%m%d).tar.gz
tar -czf $OF /usr/local
```

UNIX

The export command

- The export command puts a variable into the environment so it will be accessible to child processes. For instance:

```
$ x=hello
$ bash              # Run a child shell.
$ echo $x           # Nothing in x.
$ exit              # Return to parent.
$ export x
$ bash
$ echo $x
hello               # It's there.
```

- If the child modifies x, it will not modify the parent’s original value. Verify this by changing x in the following way:

```
$ x=ciao
$ exit
$ echo $x
hello
```

The Environment

- The Unix system is controlled by a number of shell variables that are separately set by the system some during boot sequence, and some after logging in. These variables are called system variables or environment variables.
- The set statement displays the complete list of all these variables. Built-in variable names are defined in uppercase.

The Environment

- The PATH : is a variable that instructs the shell about the route it should follow to locate any executable command.

- The HOME : when you log in, UNIX normally places you in a directory named after your login name.

- The SHELL: determines the type of shell that a user sees on logging in.

- .bash_profile : the script executed during login time. Every time you make changes to it, you should log out and log in again.

- The .bash_profile must be located in your home directory, and it is executed after /etc/profile, the universal profile for all users. Universal environment settings are kept by the administrator in /etc/profile so that they are available to all users.

UNIX

The Environment

- ALIASES : it allows you to assign short-hand names for commands you may be using quite frequently. This is done with the alias statement. Consider following ex.
  - $ alias l='ls -l'

- Aliases are listed when the alias statement is used without argument.

- The alias feature also allows you to incorporate positional parameters as variables in an alias.

For ex.

  - $ alias showdir='cd $1 ; ls -l'

- When you want to see the contents of the directory /home/arm

UNIX

Environmental Variables

- There are two types of variables:

- Local variables
- Environmental variables

- Environmental variables are set by the system and can usually be found by using the env command. Environmental variables hold special values. For instance:

```
$ echo $SHELL
/bin/bash
$ echo $PATH
/usr/local/bin:/bin:/usr/bin
```

- Environmental variables are defined in /etc/profile, /etc/profile.d/ and ~/.bash_profile. These files are the initialization files and they are read when bash shell is invoked.

- When a login shell exits, bash reads ~/.bash_logout

- The startup is more complex; for example, if bash is used interactively, then /etc/bashrc or ~/.bashrc are read. See the man page for more details.

UNIX

Environmental Variables

- HOME: The default argument (home directory) for cd.
- PATH: The search path for commands. **It is a colon-separated list of directories that are searched when you type a command.**
- LOGNAME: contains the user name
- HOSTNAME: contains the computer name.
- Usually, we type in the commands in the following way:

```
$ ./command
```

- By setting PATH=$PATH:. our working directory is included in the search path for commands, and we simply type:

```
$ command
```

- If we type in

```
$ mkdir ~/bin
```

- and we include the following lines in the ~/.bash_profile:

```
PATH=$PATH:$HOME/bin
export PATH
```

- we obtain that the directory /home/userid/bin is included in the search path for commands.

Read command

- The read command allows you to prompt for input and store it in a variable.

- Example:

```
#!/bin/bash
echo -n “Enter name of file to delete: ”
read file
echo “Type 'y' to remove it, 'n' to change your mind ... ”
rm -i $file
echo "That was YOUR decision!”
```

- Line 2 prompts for a string that is read in line 3. Line 4 uses the interactive remove (rm -i) to ask the user for confirmation.

UNIX

Variables

- We can use variables as in any programming languages. Their values are always stored as strings, but there are mathematical operators in the shell language that will convert variables to numbers for calculations.

- We have no need to declare a variable, just assigning a value to its reference will create it.

- Example

```
#!/bin/bash
STR="Hello World!"
echo $STR
```

- Line 2 creates a variable called STR and assigns the string "Hello World!" to it. Then the value of this variable is retrieved by putting the '$' in at the beginning.

UNIX

Warning !

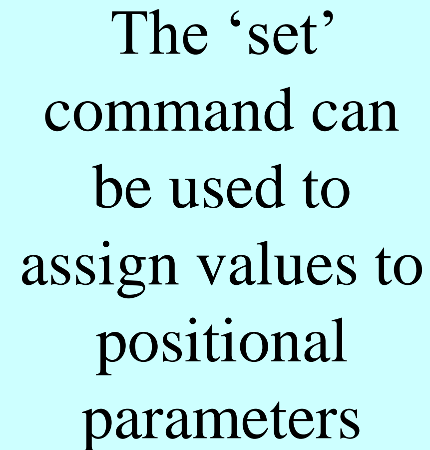
- The shell programming language does not type-cast its variables. This means that a variable can hold number data or character data.

```
count=0
count=Sunday
```

- Switching the TYPE of a variable can lead to confusion for the writer of the script or someone trying to modify it, so it is recommended to use a variable for only a single TYPE of data in a script.

Examples: Command Line Arguments

```
% set tim bill ann fred
      $1  $2   $3  $4
% echo $*
tim bill ann fred
% echo $#
4
% echo $1
tim
% echo $3 $4
ann fred
```



UNIX

Arithmetic Evaluation

- The let statement can be used to do mathematical functions:

```
$ let X=10+2*7
$ echo $X
24
$ let Y=X+2*4
$ echo $Y
32
```

- An arithmetic expression can be evaluated by $[expression] or $((expression))

```
$ echo “$((123+20))”
143
$ VALORE=$[123+20]
$ echo “$[123*$VALORE]”
17589
```

UNIX

Arithmetic Evaluation

- Available operators: +, -, /, *, %
- Example

```
$ cat arithmetic.sh
#!/bin/bash
echo -n "Enter the first number: "; read x
echo -n "Enter the second number: "; read y
add=$(($x + $y))
sub=$(($x - $y))
mul=$(($x * $y))
div=$(($x / $y))
mod=$(($x % $y))

# print out the answers:
echo "Sum: $add"
echo "Difference: $sub"
echo "Product: $mul"
echo "Quotient: $div"
echo "Remainder: $mod"
```

UNIX

Conditional Statements

- Conditionals let us decide whether to perform an action or not, this decision is taken by evaluating an expression. The most basic form is:

```
if [ expression ];
then
        statements
elif [ expression ];
then
        statements
else
        statements
fi
```

- the elif (else if) and else sections are optional

- Put spaces after [and before], and around the operators and operands.

UNIX

Expressions

- An expression can be: String comparison, Numeric comparison, File operators and Logical operators and it is represented by [expression]:

- String Comparisons:

=	compare if two strings are equal
!=	compare if two strings are not equal
-n	evaluate if string length is greater than zero
-z	evaluate if string length is equal to zero

- Examples:

[s1 = s2]	(true if s1 same as s2, else false)
[s1 != s2]	(true if s1 not same as s2, else false)
[s1]	(true if s1 is not empty, else false)
[-n s1]	(true if s1 has a length greater then 0, else false)
[-z s2]	(true if s2 has a length of 0, otherwise false)

UNIX

Expressions

- Number Comparisons:

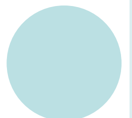
-eq	compare if two numbers are equal
-ge	compare if one number is greater than or equal to a number
-le	compare if one number is less than or equal to a number
-ne	compare if two numbers are not equal
-gt	compare if one number is greater than another number
-lt	compare if one number is less than another number

- Examples:

[n1 -eq n2]	(true if n1 same as n2, else false)
[n1 -ge n2]	(true if n1greater then or equal to n2, else false)
[n1 -le n2]	(true if n1 less then or equal to n2, else false)
[n1 -ne n2]	(true if n1 is not same as n2, else false)
[n1 -gt n2]	(true if n1 greater then n2, else false)
[n1 -lt n2]	(true if n1 less then n2, else false)

Relational Operators

Meaning	Numeric	String
Greater than	-gt	
Greater than or equal	-ge	
Less than	-lt	
Less than or equal	-le	
Equal	-eg	= or ==
Not equal	-ne	!=
str1 is less than str2		str1 < str2
str1 is greater str2		str1 > str2
String length is greater than zero		-n str
String length is zero		-z str



Examples

```
$ cat user.sh
 #!/bin/bash
  echo -n “Enter your login name: "
  read name
  if [ “$name” = “$USER” ];
  then
          echo “Hello, $name. How are you today ?”
  else
          echo “You are not $USER, so who are you ?”
  fi
```

```
$ cat number.sh
#!/bin/bash
  echo -n “Enter a number 1 < x < 10: "
  read num
  if [ “$num” -lt 10 ];     then
          if [ “$num” -gt 1 ]; then
                  echo “$num*$num=$(($num*$num))”
          else
                  echo “Wrong insertion !”
          fi
  else
          echo “Wrong insertion !”
  fi
```