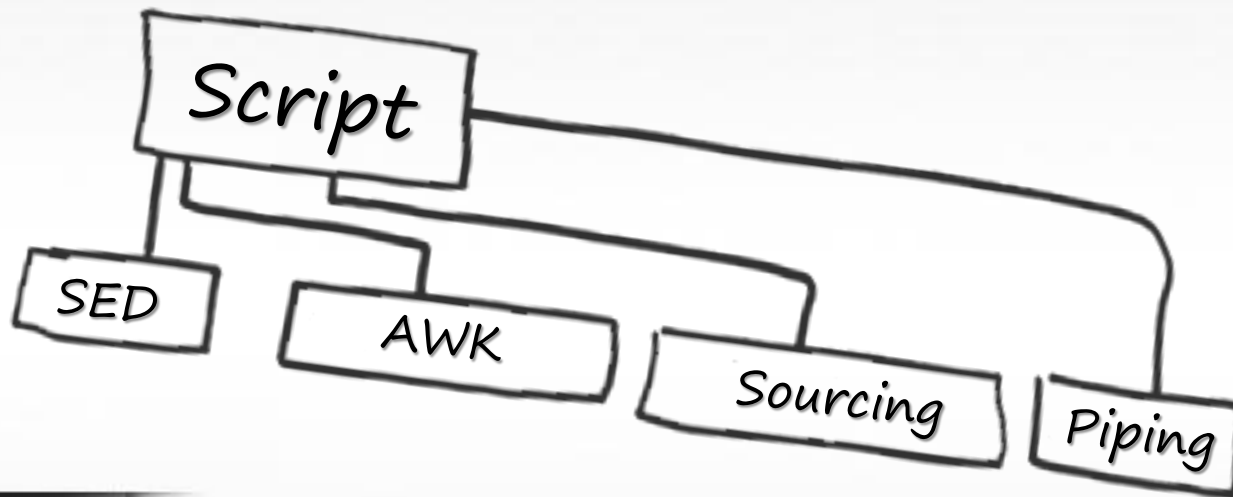


Shell Scripting



Course Materials



You can access the course materials via this link

<http://goo.gl/fbyDAo>

Day 2 Contents



- Using / Avoid using Shell Script
- Standard Shells
- Built-in commands
- Variables
- Flow Control

When do you use shell script?



- When you want to extract information from a lot of data
- It supports the user by allowing tools for
 - Data selection
 - Data combination
 - Decision and rules.
- It automates repetitive tasks
- It so simple
- Doesn't need a compiler
- It is portable

When do you AVOID Shell script



- This task will be done once.
- It is a complex task and need user interactive.
- It requires different software tools or different hardware environment.

Standard Shells



- Bourne Shell (sh)

Most system Administration's scripts are Bourne shell scripts

- C Shell (csh)

Command line history, aliasing, and job control

- sh shell is more simpler and faster
- Korn shell (ksh)

Editing history, aliasing, functions, regular expression wildcards, job control and special debugging features

Built-in Commands



- cd
- export
- umask
- exit
- break
- continue
- test
- for
- if

Built-in Commands



- newgrp
 - read
 - set
 - until
 - while
-
- To know if a command is built-in or not use the whence -v/type command

Creating a shell Script



- A shell program is a combination of UNIX commands, programming constructions and comments.
- To execute the script use `chmod` command to turn on the execute permission.

Creating a shell Script



- The first line

```
# ! /usr/bin/bash
```

- Comments

```
# calculating x
```

Example



```
$ vi hi1
#!/usr/bin/ksh
# this is my first korn shell script
```

```
print hi there
```

```
$ chmod +x hi1
```

```
$ ./hi1
hi there
```

Using sourcing

```
$ . ./hi1
hi there
```

Variables



- Type of Variables:
 - Local Variables
 - Environment Variables
 - Predefined Variables

Local Variables



- Local variables are given values that are known only to the shell in which they are created.
- Variables names must begin with an alphabetic or underscore character.

Examples



```
$ state=cal
$ echo $state
    cal
$ name="Sherine Bahader"
$ echo $name
    Sherine Bahader
$ x=
$ echo $x

$ echo ${state}ifornia
    california
```

Arithmetic



- ksh support integers

```
typeset -i variable
```

```
$ typeset -i num
```

```
$ num=5+5
```

```
$ echo $num
```

```
10
```

Examples

```
$ num=5 + 5  
ksh: +: not found
```

```
$ num=4*6  
$ echo $num  
24
```

```
$ num="4 * 6"  
$ echo $num  
24
```

```
$ num=6.789  
$ echo $num  
6
```

```
$ num=hello  
$ echo $num  
0
```



Examples

```
$ i=5
$ let i=i+1
$ print $i
6
```

```
$ let "i = i + 2"
$ print $i
8
```

```
$ let "i+=1"
$ print $i
Output:
9
```

```
$ i=9
$ ((i = i * 6))
$ print $i
54
```



Environment Variables



- To set a variable

```
VAR=value
```

```
export VAR=value
```

- To unset a variable

```
unset VAR
```

- To display all variables

```
set
```

```
env
```

```
export
```

- To display values stored in variables

```
echo $VAR
```

```
print $var
```


Environment Variables



- Environment variables are available to the shell in which they are created and any sub-shells.

PATH

HOME

PS1

LOGNAME

PS2

... •

Environment Variables



```
PS1="$LOGNAME@`uname -n`: \$ "
```

```
echo $PS1
```

```
user1@host1: $
```

```
echo $PATH
```

```
/usr/dt/bin:/usr/openwin/bin:/usr/bin:/usr/ucb
```

```
PATH=$PATH:~
```

```
echo $PATH
```

```
/usr/bin:/bin
```

Environment Variables



- Quoting is a process that instructs the shell to mask, or ignore, the special meaning of metacharacters.
- Single forward quotation instruct the shell to ignore all metacharacters.
- Double quotation instruct the shell to ignore all metacharacters except \$ ' \
- A backslash (\) character prevents the shell from interpreting the next character as a metacharacter.

Environment Variables



```
echo '$SHELL'
$SHELL
```

```
echo "$SHELL"
/bin/ksh
```

```
echo "\$SHELL"
$SHELL
```

```
echo "Today's date is `date`"
Today's date is Tue May 2 14:10:05 MDT 2002
```

```
echo "The user is currently in the $(pwd) directory."
The user is currently in the /home/user1 directory.
```

Predefined Variables



- Predefined variables are variables known to the shell and their values are assigned by the shell.

- \$#

Number of arguments

- \$*

List of all arguments

- \$0

Script name

- \$1, \$2, . . .

First argument, second argument,...

- \$?

Return code of the last command

Examples



```
$ print The name of the script $0
$ print The first argument $1
$ print The second argument $2
$ print the number of arguments $#
$ oldarg=$*
# reset predefined variables
$ set Ahmed Mohamed Adel
$ print all arguments are $*
$ print the number of arguments $#
$ print $oldarg
# predefined variables are unassigned
$ set --
$ print Good-bye for now, $1
$ set $oldarg
$ print $*
```

Reading User Input



```
#!/usr/bin/ksh
print "Are you happy ?"
read answer
print "$answer is the right response."

print "What is your full name?"
read first middle last
print "hello $first"

print "where do you work?"
read
print I guess $REPLY keeps you busy !

read place?"where do you live?"
print Welcome to $place, $first $last
```

Conditional Constructs and Flow control



- Conditional commands allow you to perform some tasks based on whether or not a condition succeeds or fails
 - `if`
 - `if/else`
 - `if/elif/else`
- The shell expects a command after an `if` and the exit status of the command is used to evaluate the condition.
- To evaluate an expression use the `test` command or double brackets.

Testing and logical operations



- String Testing

`string1=string2`

string1 is equal string2

`string1!=string2`

string1 is not equal string2

`string`

string is not null

`-z string`

Length of string is zero

`-n string`

Length of string is nonzero

Testing and logical operations (cont.)



- Integer Testing

`int1 -eq int2`

`int1 -ne int2`

`int1 -gt int2`

`int1 -ge int2`

`int1 -lt int2`

`int1 -le int2`

- equal to
- not equal to
- greater than
- greater and equal
- less than
- less and equal

Testing and logical operations (cont.)



- !
- -a
- -o
- -f filename
- -h filename
- -r filename
- -w filename
- -x filename
- not operator
- and operator
- or operator
- file existence
- symbolic link
- readable
- writable
- executable

Examples



```
$ test -r fname
```

```
$ test "islam" = "islama"
```

```
$ test 5 -gt 3
```

```
$ test "sbahader" = "sbahader" -a 5 -gt 3
```

The `if` command



```
if command
then
    ... commands ...
fi

or

if [ expression ]
then
    ... commands ...
fi

if command
then
    ... commands ...
    if command
    then
        ... commands ...
    fi
fi
```

Examples



```
$ if test -f file1
  > then
  > cat file1
> fi
```

```
$ if [ -f file1 ]
  > then
  > cat file1
> fi
```

```
$ print "Are you ok?"
$ read answer
$ if [ $answer = Y -o $answer = y ]
  > then
  > print "Glad to hear that 😊"
➤ fi
```

Thanks 😊

SBAHADER@GMAIL.COM

