

Shell Scripting

NOW



Bash Shell

0:0 0

ԱՌՋՈՂՈՎՐԴՅԱՆ
ԱՄԱՐՏԻՎՈՒՅԹ
ԱՎԱԼՈՒՅՈՒՆ

ՕՐԵԱԿՈՒ



Day 2 Contents

- case
- Loops
- select
- shift
- break
- continue
- Arrays
- Functions
- Debugging

The case command

```
case variable in  
value1)  
    Command(s)  
    ;;  
value2)  
    Command(s)  
    ;;  
* )  
    Command(s)  
    ;;  
esac
```

ksh Sub Patterns

- **?(pattern(s))**
 - Match one or zero occurrence of any of the patterns
- ***(pattern(s))**
 - Match zero or more occurrence of any of the patterns
- **@(pattern(s))**
 - Match exactly one occurrence of any of the patterns
- **+(pattern(s))**
 - Match one or more occurrence of any of the patterns
- **!(pattern(s))**
 - Match all strings except any of the patterns

Example

```
#!/bin/ksh
case $var in
+ ([a-z] ) ) echo " lower case "
;;;
+ ([A-Z] ) echo " upper case "
;;;
+ ([0-9] ) echo " integer "
;;
esac
```

Looping Commands

- The while command
- The until command
- The for command

The while command

```
while command  
do  
... command ...  
done
```

- Examples

```
num=0  
while [ $num -lt 10 ]  
do  
    echo $num  
    let num=$num+1  
done
```

Examples (cont.)

```
echo what is my name
```

```
read ans
```

```
while [ $ans != "sherine" ]
```

```
do
```

```
    echo Wrong Answer!
```

```
    echo Try Again
```

```
    read ans
```

```
done
```

```
echo You Got It ☺
```

The until command

```
until command  
do  
    command(s)  
done
```

- Example

```
hour=1  
until [ $hour -gt 24 ]  
do  
    case $hour in  
        [0-9] | 1[0-1]) echo good morning ;;  
        12) echo lunch time ;;  
        1[3-7]) echo work time ;;  
        *) echo Good Night ;;  
    esac  
    let hour=$hour+1  
done
```

The **for** command

- It is used to execute commands a finite number of times on a list of items (files/usernames)

```
for variable in word list  
do  
    ... commands ...  
done
```

The **for** command (cont.)

```
for pal in mona ahmed maha  
do  
    echo hi $pal  
done  
  
for person in `cat mylist`  
do  
    mailx $person < letter  
    echo mail to $person was sent  
done
```

The select command and Menus

- The select loop is an easy way for creating menus.
- The PS3prompt is used to prompt the user for an input.
- The input should be one of the numbers in the menu list.
- The input is stored in the special variable RELPY.
- The casecommand is used with the select command to make it possible for the user to make a select from the menu.

Examples

```
select choice in Ahmed Adel Tamer do case
$choice in Ahmed) print Ahmed is good boy
;; Adel) print Adel is the best ;; Tamer)
print Tamer is a bad boy ;; *) print
$REPLY is not one of the choices. ;;
esac
done
```

Examples

Output: 1) Ahmed 2) Adel
3) Tamer #? **1** Ahmed is a
good boy 1) Ahmed 2) Adel
3) Tamer #?**5** 5 is not one of
the choices 1) Ahmed 2) Adel
3) Tamer

Examples (cont.)

```
print Choose from the following:  
select choice in Ahmed Adel Tamer  
do  
case $choice in  
Ahmed) print Ahmed is good boy  
        break;;  
Adel) print Adel is the best  
        break;;  
Tamer) print Tamer is a bad boy  
        break;;  
*) print $REPLY is not one of the choices.  
    print Try again  
;;  
esac  
done
```

The shift command

- It shifts the parameter list to the left a specified number of times.
- The shiftwithout arguments shifts the parameter list once to the left.
- Once the list is shifted, the parameter is removed permanently.
- It is often used in while loops.

Examples

```
While (( $#>0 ))
do
print $*
shift
done
:wq doit

$ ./doit a b c d e
a b c d e
b c d e
c d e
d e
e
```

The break command

- The `break` command is used to force immediate exit from the loop, but not from the program.
- Example

```
while true
do
echo "Are you ready to move on?"
read answer
if [ [ $answer =[Yy]* ] ] # the new test command
then
break
else
...commands..
done
echo "Here are you?"
```

The continue command

- The `continue` command is used to starts back at the top of the loop
- Example

```
for name in `cat names`  
do  
if [ $name = sherine ]  
then  
continue  
else  
mailx $name <memo  
fi  
done
```

Example for Nested Loops

```
while true
do
    for user in Ahmed Tamer Samy
    do
        if [[ $user = [Tt]* ]]
        then
            echo A Hi from Tamer
            continue
        fi
    while true
        do
            if [[ $user = [S]* ]]
            then
                echo A Hi from Samy
                break 3
            fi
            echo A Hi from Ahmed
            continue 2
            done
        done
    done
echo Out of the Loop
```

Substitutions

```
$ echo $(date)  
Thu Jan 9 13:38:21 EET 2003  
$ var=$(date)  
$ print var  
var  
$ echo $var  
Thu Jan 9 13:38:49 EET 2003  
$ cat x  
abc  
$ var=`cat x`  
$ echo $var  
abc
```

Arrays

- korn/bash shells are one-dimensional arrays that contain up to 1024 elements consisting of words or integers.
- Index starts with zero.
- Each element can be set and unset individual.
- Values do not have to be set in any particular order.

Examples

- To set the value of array element

```
$array[0]=ahmed  
$array[1]=ali  
$array[2]=mohamed  
$array= ("ahmed" "ali" "mohamed")
```

- To print the values of the array elements

```
$echo ${array[0]}  
ahmed  
$echo ${array[1]}  
ali  
$echo ${array[2]}  
mohamed
```

Examples (cont.)

- To declare a 2 integer elements array

```
$typeset -i ele[2]
$ele[0]=50
$ele[1]=happy
ksh:happy:bad number

$ele[1]=6
```

- To display all the elements in the array

```
$print ${ele[*]}
50 6
$print ${ele[@]}
50 6
```

- To display the number of elements in the array

```
$print ${#ele[@]}
2
```

Functions

- Functions are used to modularize programs.
- A Function is a collection of commands that can be executed simply by entering the function's name.
- ksh executes aliases, built-in commands, functions and then executables.
- Functions must be defined before it is used.

Functions (cont.)

- Functions are run in the current environment ; it shares variables.
Local variables can be declared in the function using typeset.
- Functions can be exported to sub-shells
-
- To list functions and definitions use the alias functions.
Functions can be recursive
-

Functions (cont.)

- Format

```
function function-name { commands; commands; }
```

- Example

```
function pr  
{print $0 must take arg; exit 1; }
```

Example

```
#! /usr/bin/bash
function increment {
    typeset sum
    (( sum = $1 + 1 ))
    return $sum
}
echo The sum is
increment 5
echo $?
echo $sum
```

Handling Errors and Debuggers

- If you need to know the way your scripts runs , and the values of such variables during execution ; you can use either one of 2 options:
 - Write `set -x` inside your script before the part you want to debug,
 - and `set +x` at the end of the part.

Run the script using command `bash -x script-name` so that it is equal to `set -x` but for the whole script.
- Using these ways let the shell write each command before executing it and also but the values of the variables. So you can test the sequence of the execution and check if variable is expanded correctly.



Thank you