Unix Shell Programming

Chapter 5: Looping

Roshan Chitrakar, PhD
roshanchi@gmail.com

for

- The for command is used to execute a set of commands a specified number of times.

```
for var in word1 word2 ... wordn

do

        command

        command

        ...

done
```

```
for i in 1 2 3

do

        echo $i

done
```

for (contd.)

- the shell permits filename substitution in the list of words in the for

```
for file in memo[1-4]
do
        run $file
done
```

```
for file in *
do
        run $file
done
```

```
files=$(cat filelist)


for file in $files
do
        run $file
done
```

for with arguments

```
for arg in $*
do
        echo $arg
done
```

```
for arg in "$@"
do
        echo $arg
done
```

for without lists

In absence of the keyword ***in***, the shell automatically sequences through all the arguments typed on the command line.

```
for arg
do
        echo $arg
done
```

while

- $command_t$ is executed and its exit status tested. If it's zero, the commands enclosed between the do and done are executed.

```
while command_t

do

        command

        command

        ...

done
```

```
while [ "$i" -le 5 ]

do

        echo $i

        i=$((i + 1))

done
```

```
while [ "$#" -ne 0 ]

do

      echo "$1"

      shift

done
```

until

- The until command is similar to the while, only it continues execution as long as the command that follows the until returns a nonzero exit status.

```
until who | grep "^$user " > /dev/null
do
        sleep 60
done
echo "$user has logged on"
```

```
until command_t
do
        command
        command
        ...
done
```

break and ***continue***

```
for file
do
        ...
        while [ "$count" -lt 10 ]
        do
                ...
                if [ -n "$error" ]
                then
                        break 2
                fi
                ...
        done
        ...
done
```

- To just exit from the loop, you can use the ***break*** command. ***break*** can be followed by a number n, which means that it will break two innermost loops.
- The continue command causes the remaining commands in the loop to be skipped. Execution of the loop then continues as normal.

```
for file
do
        if [ ! -e "$file" ]
        then
                echo "$file not found!"
                continue
        fi
        ...
done
```

loop options

- Executing a loop in the background
  - ***done &***
- I/O redirection on a loop
  - ***done >*** *output*
  - ***done 2>*** *errors*
- Piping data into and out of a loop
  - ***done |*** *command*
- Typing a loop on one Line
  - Put a semicolon after the last item in the list and one after each command in the loop e.g.
  - ***$ for i in 1 2 3 4; do echo $i; done***

getopts

- allows a program to take options in any order on the command line.
- Syntax: ***getopts options variable***
- If getopts finds a matching option in the arguments, the letter is stored inside the specified variable and returns a zero exit status.
- It stores a question mark, if it doesn't find a matching option.
- Example: -
  - ***getopts air option***
  - ***$ foo -a -i -r or foo -air***
- The getopts command also handles the case where an option must be followed by an argument. You write a colon character after the option letter on the getopts command line e.g.
  - ***getopts mt: option***