Lesson 5: Using Conditional Statements

5.1 Using **if** then **fi**

Using **if** ... then ... **fi**

- Used to perform actions based on a specific expression

```
if expression
then
    command 1
    command 2
fi
```

- **else** can be included to define what happens if expression is not true

```
if expression
then
    command 1
else
    command 2
fi
```

Using **if** ... then ... **fi**

- **elif** is used to nest a new if statement within the **if** statement
- While **if** has to be closed with a **fi**, **elif** does not need a separate **fi**

```
if expression
then
        command 1
elif expression 2
then
        command 2
fi
```

if ... then ... **fi** Example

```
#!/bin/bash
if [ -d $1 ]
then
        echo $1 is a directory
elif [ -f $1 ]
then
        echo $1 is a file
else
        echo $1 is not a file, nor a directory
fi
```

```
#!/bin/bash
if [ -d $1 ]
then
        echo $1 is a directory
elif [ -f $1 ]
then
        echo $1 is a file
else
        echo $1 is not a file, nor a director
fi
```

```
[root@server1 bin]# vim ifdir
[root@server1 bin]# vim ifdir
[root@server1 bin]# ifdir /etc/passwd
/etc/passwd is a file
[root@server1 bin]# ifdir /etc
/etc is a directory
[root@server1 bin]# ifdir /abcd
/abcd is not a file, nor a director
[root@server1 bin]# ifdir
is a directory
[root@server1 bin]#
```

Solution of the above the example script

Using && and ||

- **&&** and **||** are the logical AND and OR
- Use them as a short notation for **if** ... then ... **fi** constructions
- When using **&&**, the second command is executed only if the first returns an exit code zero
  - **[-z $1] && echo $1 is not defined**
- When using **||**, the second command is executed only if the first command does not return an exit code 0
  - **[-f $1] || echo $1 is not a file**

&& and || Example

```
#!/bin/bash
[ -z $1 ] && echo no argument provided && exit 2
[ -f $1 ] && echo $1 is a file && exit 0
[ -d $1 ] && echo $1 is a directory && exit 0
```

```
#!/bin/bash
#if [ -d $1 ]
#then
#       echo $1 is a directory
#elif [ -f $1 ]
#then
#       echo $1 is a file
#else
#       echo $1 is not a file, nor a directory
#fi

####
# alternative notation
####

[ -d $1 ] && echo $1 is a directory
[ -f $1 ] && echo $1 is a file || echo $1 is not a file, nor a directory
~
```

Using for

- **for** statements are useful to evaluate a range or series

```
for i in something
do
        command 1
        command 2
done
```

- **for i in `cat /etc/hosts`; do echo $i; done**
- **for i in {1..5}; do echo $i; done**
- It is common to use a variable **i** in a **for** loop, but any other variable can be used instead

` backtick help us to use command substitution but $ dollar and ( ) braces also help the same

```
melissa
bill
steve
larrie
linus
~
```

```
[root@server1 bin]# vim users
[root@server1 bin]# for i in `cat users`; do echo $i; done
melissa
bill
steve
larrie
linus
[root@server1 bin]# for i in `cat users`; do echo useradd $i; done
useradd melissa
useradd bill
useradd steve
useradd larrie
useradd linus
[root@server1 bin]# for i in {200..210}; do ping -c 1 192.168.122.$i; done
PING 192.168.122.200 (192.168.122.200) 56(84) bytes of data.
64 bytes from 192.168.122.200: icmp_seq=1 ttl=64 time=0.323 ms

--- 192.168.122.200 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.323/0.323/0.323/0.000 ms
PING 192.168.122.201 (192.168.122.201) 56(84) bytes of data.
From 192.168.122.210 icmp_seq=1 Destination Host Unreachable

--- 192.168.122.201 ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms

PING 192.168.122.202 (192.168.122.202) 56(84) bytes of data.
```

That is the best case

```
[root@server1 bin]# for i in {200..210}; do ping -c 1 192.168.122.$i > /dev/null && echo 192.168.122.$i is available; done
192.168.122.200 is available
```

Example with **for**

```
#!/bin/bash
# script that counts files
echo which directory do you want to count?
read DIR
cd $DIR
COUNTER=0

for i in *
do
        COUNTER=$((COUNTER+1))
        echo I have counted $COUNTERfiles in this directory
done
```

* refers to interprets to the bash shell to all file in a current directory

5.4 Using **case**

Using **case**

- **case** is used if specific values are expected
- The most common example is in the legacy system V / Upstart init scripts in /etc/init.d

```
case $VAR in
        yes)
        echo ok;;
        no|nee)
        echo too bad
        ;;
        *)
        echo try again
        ;;
esac
```

```
[root@server1 bin]# cd /etc/init.d
[root@server1 init.d]# ls
functions  netconsole  network  README
[root@server1 init.d]# vim network
```

```
# See how we were called.
case "$1" in
  start)
        [ "$EUID" != "0" ] && exit 4
        rc=0
        # IPv6 hook (pre IPv4 start)
        if [ -x /etc/sysconfig/network-scripts/init.ipv6-global ]; then
                /etc/sysconfig/network-scripts/init.ipv6-global start pre
        fi

        apply_sysctl

        # bring up loopback interface
        action $"Bringing up loopback interface: " ./ifup ifcfg-lo

        case "$VLAN" in
          yes)
            if [ ! -d /proc/net/vlan ] && ! modprobe 8021q >/dev/null 2>&1 ; then
```

5.5 Using while and until

Using while and until

- **while** is used to execute commands as long as a condition is true
- **until** is used to execute commands as long as a condition is false

```
while|until condition
do
     command
done
```

This is the infinite loop

```
[root@server1 ~]# while true; do true; done &
[1] 2820
[root@server1 ~]#
```

```
top - 12:43:42 up 13 min,  2 users,  load average: 0.45, 0.16, 0.08
Tasks: 298 total,   3 running, 295 sleeping,   0 stopped,   0 zombie
%Cpu(s):  2.5 us,  1.2 sy,  0.0 ni, 96.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem:   1010856 total,   306028 used,   704828 free,     1080 buffers
KiB Swap:   839676 total,        0 used,   839676 free.   111204 cached Mem

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
 2820 root      20   0  116112   1264    160 R 87.8  0.1   0:21.43 bash
 2821 root      20   0  123812   1660   1068 R  5.5  0.2   0:00.02 top
    1 root      20   0   53780   7600   2512 S  0.0  0.8   0:01.79 systemd
    2 root      20   0       0      0      0 S  0.0  0.0   0:00.10 kthreadd
    3 root      20   0       0      0      0 S  0.0  0.0   0:00.01 ksoftirqd/0
    5 root       0 -20       0      0      0 S  0.0  0.0   0:00.00 kworker/0:0H
    7 root      rt   0       0      0      0 S  0.0  0.0   0:00.00 migration/0
    8 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcu_bh
    9 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcuob/0
   10 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcuob/1
   11 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcuob/2
   12 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcuob/3
   13 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcuob/4
   14 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcuob/5
   15 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcuob/6
   16 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcuob/7
   17 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcuob/8
   18 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcuob/9
   19 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcuob/10
   20 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcuob/11
   21 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcuob/12
   22 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcuob/13
   23 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcuob/14
   24 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcuob/15
   25 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcuob/16
   26 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcuob/17
   27 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcuob/18
```

Example with **while**

```
#!/bin/bash
COUNTER=0

while true
do
        sleep 1
        COUNTER=$(( COUNTER + 1 ))
        echo $COUNTER seconds have passed since starting this script
done
```

```
[root@server1 bin]# vim counter
[root@server1 bin]# counter
1 seconds have passed since starting this script
2 seconds have passed since starting this script
3 seconds have passed since starting this script
4 seconds have passed since starting this script
5 seconds have passed since starting this script
6 seconds have passed since starting this script
^C
[root@server1 bin]#
```

Example with **until**

```
#!/bin/bash

until users | grep $1 > /dev/null
do
        echo $1 is not logged in yet
        sleep 5
done

echo $1 has just logged in
mail –s "$1 has just logged in" root < .
```

```
[root@server1 bin]# mail -s hello root
message itself
.
EOT
[root@server1 bin]#
```

In the above interface it open the new subshell which help us to know about the . EOT but in bash it does not recognize well so that's why it will redirect to the end < .

First we need to run the users script

```
[root@server1 bin]# untilusers user
user is not logged in yet
user is not logged in yet
user is not logged in yet
user is not logged in yet
user is not logged in yet
user is not logged in yet
user is not logged in yet
user is not logged in yet
user has just logged in
Null message body; hope that's ok
You have new mail in /var/spool/mail/root
[root@server1 bin]# mail
Heirloom Mail version 12.5 7/5/10.  Type ? for help.
"/var/spool/mail/root": 3 messages 3 new
>N  1 user@localhost.examp  Wed Jul 29 08:32 183/8619  "[abrt] full crash report"
 N  2 root                  Fri Jan 15 12:47  18/598   "hello"
 N  3 root                  Fri Jan 15 12:49  18/615   "user has just logged in"
&
```

After running the script when I logging the user it will mssg on the screen as above and send a mail

Exercise 5

- A customer has exported a long list of LDAP user names. These usernames are stored in the file ldapusers. In this file, every user has a name in the format cn=lisa,dc=example,dc=com. Write a script that extracts the username only (lisa) from all of these lines and write those to a new file. Based on this new file, create a local user account on your Linux box.
- Note: while testing it's not a really smart idea to create the user accounts directly. Find a solution that proves that the script works, without polluting your system with many usernames.

```
#!/bin/bash
# extract the user names
for i in $(cat ldapusers)
do
        USER=${i%%,*}
        USER=${USER#*=}
        echo $USER >> users
done

# show that user creation will work
for j in $(cat users)
do
        echo useradd $j
done

exit 0
```