IF , while and for loop statements

By Mohsen Mahdavifar

گردآوری شده توسط محسن مهدوی فر

Laitec.ir

(Mahdavifar@laitec.ir)

If Statement

با استفاده از IF میتوان عبارات شرطی نوشت .شما می توانید عملیات های متفاوتی با توجه به شرایط مشخص شده انجام دهید . حلقه شرطی IF می توانید این قابلیت را برای سیستم ایجاد کنید که بتواند در شرایط متفاوت تصمیم گیری کند .

نحوه نگارش دستور IF

```
if   [ conditional-expression ]
                then
               action1
               action2
                  .
                  .
                  .
               else
                  .
                  .
                  .
               fi
```

شرط if با fi تمام خواهد شد .

بررسی یک مثال ساده :

```
Mahdavifar=Mohsen
if [ $Mahdavifar = Mohsen ]
then
echo "Linux"
fi
```

در مثال بالا متغیری به نام Mahdavifar تعریف شده است که مقدار آن برابر Mohsen قرار داده شده است دستور If به بررسی مقدار Mahdavifar می پردازد اگر مقدار آن برابر Mohsen بود عبارت Linux را چاپ می کند در غیر این صورت به دلیل عدم وجود else عملیاتی برای ما انجام نخواهد داد .

نکته : بین [(براکت) و عبارت بعدی باید یک فاصله وجود داشته باشد در غیر این صورت ارور خواهید گرفت .

نکته ۲ : بین دو طرف = نیز فاصله ای وجود داشته باشد و متغیر ها به = نچسبیده باشند .

نحوه نگارش اشتباه دستور IF

```
if [$Mahdavifar = Mohsen ]
```

عدم وجود فاصله بین [و متغیر Mahdavifar

```
if [$Mahdavifar = Mohsen]
```

عدم وجود فاصله بین [و متغیر Mahdavifarو مقدار Mohsen

```
if [ $Mahdavifar= Mohsen ]
```

عدم وجود فاصله بین = و متغیر Mahdavifar

```
if [ $Mahdavifar=Mohsen ]
```

عدم وجود فاصله بین = و متغیر Mahdavifar و مقدار Mohsen

```
if [$Mahdavifar=Mohsen]
```

عدم وجود فاصله بین متغیر ها و] و =

در مثال قبل بعد از هر با اجرای برنامه عبارت Linux چاپ خواهد شد به دلیل اینکه مقدار Mahdavifar همیشه برابر Mohsen بوده و شرط If همیشه برقرار بوده و سپس دستورات echo اجرا خواهد شد .

اگر میخواهید در صورت درست نبودن شرط if مجموعه دستوراتی را اجرا کنید میتوانید از else استفاده کنید .

مثال

```
Mahdavifar=19
if [ $Mahdavifar == 20 ]
then
echo "value is 20"
else
echo "value is not 20"
fi
```

در مثال بالا مقدار متغیر Mahdavifar برابر 19 قرار گرفته است . شرط IF در صورتی درست است که مقدار Mahdavifar برابر 20 باشد که در اینجا شرط اشتباه بوده و مجموعه دستورات مربوط به else اجرا خواهد شد .

اگر چندین شرط برای چک کردن دارید میتوانید از else و elif استفاده کنید که از else if مشتق شده است .

به مثال زیر دقت کنید

```
Mahdavifar=6

if [ $Mahdavifar == 5 ]
then
  echo " value is five"
elif [ $Mahdavifar == 6 ]
then
  echo "value is six"
else
echo "none of the above"
fi
```

در مثال بالا مقدار متغیر Mahdavifar اگر برابر 5 بود عبارت value is five و اگر برابر 6 بود value is six و اگر هیچ کدام از مقادیر 5 و 6 نبود عبارت none of the above نمایش داده خواهد شد .

شما میتوانید به تعداد دلخواه شرط برای چک شدن در IF داشته باشید با استفاده از Keyword گفته شده در مثال قبل elif میتوانید به هر تعداد که نیاز دارید شرط تعریف کنید

در مثال زیر چندیدن شرط توسط elif چک شده است .

```
Laitec=5
if [ $Laitec == 5 ]
        then
echo "Laitec is five"
elif [ $Laitec == 6 ]
        then
 echo "Laitec is six"
elif [ $Laitec == 7 ]
        then
echo "Laitec is seven"
elif [ $Laitec == 8 ]
        then
echo "Laitec is eight"
elif [ $Laitec == 9 ]
        then
echo "Laitec is nine"
        else
echo "none of the above"
        fi
```

یکی از راههای دیگر نوشتن چنین دستوری استفاده از دستور Case می باشد که به صورت زیر است

نحوه نگارش case

```
case  $variable-name  in
pattern1)
command1
...
....
commandN
;;
pattern2)
command1
...
....
commandN
;;
patternN)
command1
...
....
```

```
commandN
;;
*)
esac
```

برای تبدیل مثال قبل و نوشتن آن با دستور Case به صورت زیر عمل میکنیم

```
case "$Laitec" in
5)
echo " Laitec is five"
;;
6)
echo " Laitec is six"
;;
7)
echo " Laitec is seven"
;;
8)
echo " Laitec is eight"
;;
9)
echo " Laitec is nine"
;;
*)
echo "none of the above"
esac
```

دستورات if معمولا داخل حلقه های for و while نیز مورد استفاده قرار می گیرد .

به طور نمونه به مثال زیر توجه کنید

```
Num=1
while [ $Num -le 9 ]
do
sleep 1
(( Num++ ))
if [ $Num == 5 ]
then
continue
fi
echo "$Num "
done
echo Finished
```

در مثال فوق مقدار Num برابر ۱ قرار گرفته است حلقه while تا زمانی که مقدار Num کوچکتر از ۹ باشد انجام خواهد شد . با دستور sleep 1 یک ثانیه صبر خواهد کرد مقدار Num در خط بعدی ۱ مقدار افزایش خواهد یافت . برای افزایش مقدار Num از دستور رو به رو نیز میتوانیم استفاده کنیم

```
Num=$ (($num+1) )
```

در هنگامی که شرط دستور `if` برقرار گردد یعنی هنگامی که مقدار **Num** برابر 5 قرار گیرد دستور `continue` اجرا خواهد شد . دستور فوق باعث می شود که پرش به تکرار بعدی حلقه انجام شود بدین معنی که برای مقدار 5 مجموعه دستورات `while` انجام نخواهد شد و مقدار **Num** بدون انجام عملیاتی یک مقدار افزایش خواهد یافت .

خروجی تکه برنامه فوق به صورت زیر خواهد بو د

```
2
3
4
6
7
8
9
19
Finished
```

WHILE loops

ما می توانیم مجموعه از کدهایی را که تا یک زمان خاص نیاز به تکرار دارند را با استفاده از دستور **while** اجرا کنیم

به مثال ساده زیر توجه کنید

```
#!/bin/bash
Liatec=1
while [ $Liatec -le 9 ]
do
echo "$Liatec "
sleep 1
(( Liatec++ ))
done
```

مادامی که مقدار **Liatec** کمتر از ۹ باشد پس از ۱ ثانیه صبر مقدار **Laitec** ۱ واحد افزایش و چاپ خواهد شد.

ما میتوانیم توسط دستور **while** نیز حلقه های بی نهایت ایجاد کنیم

به مثال زیر توجه فرمایید

```
#!/bin/bash
count=1
while :
do
```

```
echo "$count
   sleep 1
(( count++ ))
   done
```

ما میتوانیم از چندین شرط نیز برای حلقه **while** مان در نظر بگیریم نحوه نگارش برای بررسی چندین شرط به صورت زیر است

AND :

```
[ EXPR1 -a EXPR2 ]
```

OR

```
[ EXPR1 -o EXPR2 ]
```

به مثال زیر توجه فرمایید

```
Laitec=1
Test=0
while [ "$Laitec" -le "9" -a  "$Test" == "0" ]
do
echo "$Laitec"
sleep 1
(( Laitec++ ))
if [ "$Laitec" == 5 ]
then
Test=1
fi
done
```

خروجی تکه کد بالا اعداد ۱ تا ۴ خواهد بود در تکه کد بالا ابتدا مقدار Laitec برابر 1 قرار داده شد و مقدار Test برابر ۰ دستورات حلقه while تا زمانی انجام خواهند شد که مقدار Laitec کمتر از 9 بوده و مقدار Test نیز برابر ۰ باشد در هر بار انجام شدن دستورات مقدار متغیر Laitec نمایش داده می شود و ۱ ثانیه صبر کرده برنامه و متغیر Laitec ۱ مقدار افزایش می دهد . زمانی که مقدار Laitec به ۵ رسید مقدار متغیر Test ۱ برابر ۱ قرار می گیرد و شرط حلقه while نقض شه و حلقه به انتها خواهد رسید .

با استفاده از **while** نیز میتوان خط به خط از یک فایل خواند .

در مثال زیر خط به خط فایل **/etc/passwd** خوانده شده و چاپ خواهد شد .

```
while read line
do
echo $line
sleep 1
done < /etc/passwd
```

در مثال فوق خطوط **/etc/passwd** خط به خط خوانده شده و نمایش داده خواهد شد .

Line در مثال فوق یک متغیر بوده که میتواند هر اسمی داشته باشد . در مثال زیر متغیر **line** تغییر اسم داده شده است.

```
while read i
do
echo $i
sleep 1
done < /etc/passwd
```

در انتهای حلقه نیز فایلی که میخواهیم آن را خط به خط بخوانیم معرفی شده است

به استفاده از عملگر < فایل مورد نظر را معرفی میکنیم .

دستور فوق را میتوانیم به صورت زیر نیز بنویسیم

```
Cat /etc/passwd | while read i
do
echo $i
sleep 1
done
```

برای خروج از حلقه از دستور **break** استفاده میکنیم

به مثال زیر توجه فرمایید :

```
Laitec=1
while [ $Laitec -le 9 ]
do
echo "$Laitec "
sleep 1
(( Laitec ++ ))
if [ $Laitec == 5 ]
then
break
fi
done
echo Finished
```

شرط حلقه بالا مادامی که مقدار متغیر Laitec کوچکتر از 9 است صحیح بوده و مجموعه دستورات موجود در حلقه اجرا خواهد شد ولی به محض اینکه مقدار متغیر Laitec برابر 5 شود برنامه از حلقه خارج شده و عبارت finished چاپ خواهد شد .

دستور break باعث می شود که برنامه از while خارج شده و ادامه برنامه بعد از حلقه while اجرا شود .

فرق دستور continue و break در این است که هنگام اجرا شدن دستور continue برنامه از loopخارج نخواهد شد و فقط باعث می شود که برای مقداری که در حال حاضر متغیر دارد مجموعه دستورات درون loop اجرا نگردد و مقدار متغیر حلقه به مقدار بعدی تغییر کند .

به مثال زیر توجه کنید :

```
Laitec=1
while [ $Laitec -le 9 ]
do
sleep 1
(( Laitec ++ ))
if [ $Laitec == 5 ]
then
continue
fi
echo "$Laitec "
done
echo Finish
```

FOR loops

یکی از راههای بسیار مفید برای اجرای دستورات دنباله دار استفاده از اسکریپت است .

دستور for یکی از دستور های مفید برای ایجاد حلقه است .

نحوه نگارش دستور

```
for VARIABLE in 1 2 3 4 5 .. N
do
command1
command2
commandN
done
```

به مثال زیر توجه فرمایید

```
#!/bin/bash
for Laitec in 1 2 3 4 5 6 7 8 9
do
echo "$Laitec "
done
```

تکه کد بسیار ساده بالا باعث می شود متغیر Laitec از ۱ تا ۹ تغییر کرده و هر بار مقدار آن چاپ شود . خروجی دستور بالا چاپ اعداد ۱ تا ۹ است .

یکی از کاربردهای بسیار مفید حلقه for برای انجام عملیات خاصی بر روی تعدادی فایل می باشد .

در نظر بگیرید که میخواهید تمام فایلهای bzip2 را uncompress کنید میتوانید از دستور for استفاده کنید . به مثال زیر دقت شود .

```
#!/bin/bash
FILES=/home/Mahdavifar/*.bz2
for file in $FILES
do
bunzip2 $file
done
```

در مثال بالا متغیر FILE تمامی فایلهای bz2 موجود در دایرکتوری /home/Mahdavifar است که دارای پسوند bz2 هستند .

مقدار file نیز متغیر حلقه است که هر بار نام یکی از فایلهای موجود در دایرکتوری /home/Mahdavifar را که به bz2 ختم می شوند را به خود اختصاص می دهند که با هر بار انجام دستورات حلقه مقدار آن تغییر می کند .

ما می توانیم در حلقه for از range هم استفاده کنیم .

مثال زیر یکی از ساده ترین مثالهای range می باشد .

```
#!/bin/bash
for i in {1..9}
do
echo "$i"
done
```

در مثال بالا مقدار i از 1 تا 9 تغییر کرده و چاپ خواهد شد .

عبارت "{1..9}" برابر این است که ۱ تا ۹ را تک تک در شرط حلقه بنویسیم

ما می توانیم هنگامی که از range در حلقه for استفاده میکنیم مقدار افزایش متغیر یا گام افزایش متغیر حلقه را نیز تعیین کنیم.

به مثال زیر دقت شود

```
#!/bin/bash
for count in {1..10..3}
do
echo "$count"
done
```

در مثال فوق گام حلقه ۳ بوده و در هر بار اجرای حلقه مقدار count ۳ واحد پرش دارد .

خروجی دستور بالا به صورت زیر خواهد بود .

```
1
4
7
10
```

همانطور که می توان با استفاده از حلقه **while** دستوراتی را به ازای هر خط از فایل اجرا کرد همین امر را نیز میتوان با استفاده از حلقه **for**نیز انجام داد .

مثال زیر هر خط از فایل **/etc/passwd** را خوانده چاپ کرده و پس از ۱ ثانیه به سراغ خط بعدی خواهد رفت

```
for i in $(cat /etc/passwd)
do echo $i
sleep 1
done
```

یکی از نحوه های نگارش دیگر حلقه **for** به صورت **c** است

```
for (( expr1; expr2; expr3 ))
do
command1
command2
..
done
```

به مثال زیر توجه شود

```
for ((i=1, j=10; i <= 5 ; i++, j=j+5))
                    do
            echo "Number $i: $j"

                    done
```

خروجی دستور بالا به صورت زیر خواهد بود

```
Number 1: 10
Number 2: 15
Number 3: 20
Number 4: 25
Number 5: 30
```

مثال دیگری از حلقه `c like for`

```
for (( c=1; c<=5; c++ ))
do
   echo "You are Num: $c Welcome To Laitec"
done
```

خروجی دستور بالا به صورت زیر خواهد بود

```
You are Num: 1 Welcome To Laitec
You are Num: 2 Welcome To Laitec
You are Num: 3 Welcome To Laitec
You are Num: 4 Welcome To Laitec
You are Num: 5 Welcome To Laitec
```

چگونه یک حلقه for بی نهایت c like تعریف کنیم ؟

```
for (( ; ; ))
do
   echo "infinite loops [ hit CTRL+C to stop]"
done
```

ما میتوانیم توسط if وجود یک فایل دایکتوری و مواردی از این قبیل را نیز بررسی کنیم

به جدول زیر توجه کنید

1. File-based conditions:

Condition	True if	Example/explanation
[-a existingfile]	file 'existingfile' exists.	if [-a tmp.tmp]; then rm -f tmp.tmp # *Make sure we're not bothered by an old temporary file* fi
[-b blockspecialfile]	file 'blockspecialfile' exists and is block special.	Block special files are special kernel files found in /dev, mainly used for ATA devices like hard disks, cd-roms and floppy disks. if [-b /dev/fd0]; then dd if=floppy.img of=/dev/fd0 # *Write an image to a floppy* fi
[-c characterspecialfile]	file 'characterspecialfile' exists and is character special.	Character special files are special kernel files found in /dev, used for all

		kinds of purposes (audio hardware, tty's, but also /dev/null). if [-c /dev/dsp]; then cat raw.wav > /dev/dsp # *This actually works for certain raw wav files* fi
[-d directory]	file 'directory' exists and is a directory.	In UNIX-style, directories are a special kind of file. if [-d ~/.kde]; then echo "You seem to be a kde user." fi
[-e existingfile]	file 'existingfile' exists.	(same as -a, see that entry for an example)
[-f regularfile]	file 'regularfile' exists and is a regular file.	A regular file is neither a block or character special file nor a directory. if [-f ~/.bashrc]; then source ~/.bashrc fi
[-g sgidfile]	file 'sgidfile' exists and is set-group-ID.	When the SGID-bit is set on a directory, all files created in that directory will inherit the group of the directory. if [-g .]; then echo "Created files are inheriting the group '$(ls -ld . | awk '{ print $4 }')' from the working directory." fi
[-G fileownedbyeffectivegroup]	file 'fileownedbyeffectivegroup' exists and is owned by the effective group ID.	The effective group id is the primary group id of the executing user. if [! -G file]; then # *An exclamation mark inverts the outcome of the condition following it* chgrp $(id -g) file # *Change the group if it's not the effective one*

		fi
[-h symboliclink]	file 'symboliclink' exists and is a symbolic link.	if [-h $pathtofile]; then pathtofile=$(readlink -e $pathtofile) # *Make sure $pathtofile contains the actual file and not a symlink to it* fi

[-k stickyfile]	file 'stickyfile' exists and has its sticky bit set.	The sticky bit has got quite a history, but is now used to prevent world-writable directories from having their contents deletable by anyone. if [! -k /tmp]; then # *An exclamation mark inverts the outcome of the condition following it* echo "Warning! Anyone can delete and/or rename your files in /tmp!" fi
[-L symboliclink]	file 'symboliclink' exists and is a symbolic link.	(same as -h, see that entry for an example)
[-N modifiedsincelastread]	file 'modifiedsincelastread' exists and was modified after the last read.	if [-N /etc/crontab]; then killall -HUP crond # *SIGHUP makes crond reread all crontabs* fi
[-O fileownedbyeffectiveuser]	file 'fileownedbyeffectiveuser' exists and is owned by the user executing the script.	if [-O file]; then chmod 600 file # *Makes the file private, which is a bad idea if you don't own it* fi

[-p namedpipe]	file 'namedpipe' exists and is a named pipe.	A named pipe is a file in /dev/fd/ that can be read just once.
		
		if [-p $file]; then
		  cp $file tmp.tmp # *Make sure we'll be able to read*
		  file="tmp.tmp" # *the file as many times as we like*
		fi

[-r readablefile]	file 'readablefile' exists and is readable to the script.	if [-r file]; then content=$(cat file) # *Set $content to the content of the file* fi

[-s nonemptyfile]	file 'nonemptyfile' exists and has a size of more than 0 bytes.	if [-s logfile]; then
		  gzip logfile # *Backup the old logfile*
		  touch logfile # *before creating a fresh one.*
		fi

[-S socket]	file 'socket' exists and is a socket.	A socket file is used for inter-process communication, and features an interface similar to a network connection. if [-S /var/lib/mysql/mysql.sock]; then mysql --socket=/var/lib/mysql/mysql.sock fi
[-t openterminal]	file descriptor 'openterminal' exists and refers to an open terminal.	Virtually everything is done using files on Linux/UNIX, and the terminal is no exception. if [-t /dev/pts/3]; then echo -e "\nHello there. Message from terminal $(tty) to you." > /dev/pts/3 # *Anyone using that terminal will actually see this message!* fi
[-u suidfile]	file 'suidfile' exists and is set-user-ID.	Setting the suid-bit on a file causes execution of that file to be done with the credentials of the owner of the file, not of the executing user. if [-u executable]; then echo "Running program executable as user $(ls -l executable | awk '{ print $3 }')." fi

[-w writeablefile]	file 'writeablefile' exists and is writeable to the script.	if [-w /dev/hda]; then grub-install /dev/hda fi
[-x executablefile]	file 'executablefile' exists and is executable for the script.	Note that the execute permission on a directory means that it's searchable (you can see which files it contains). if [-x /root]; then echo "You can view the contents of the /root directory." fi

[newerfile -nt olderfile]	file 'newerfile' was changed more recently than 'olderfile', or if 'newerfile' exists and 'olderfile' doesn't.	if [story.txt1 -nt story.txt]; then echo "story.txt1 is newer than story.txt; I suggest continuing with the former." fi
[olderfile -ot newerfile]	file 'olderfile' was changed longer ago than 'newerfile', or if 'newerfile' exists and 'olderfile' doesn't.	if [/mnt/remote/remotefile -ot localfile]; then cp -f localfile /mnt/remote/remotefile # *Make sure the remote location has the newest version of the file, too* fi
[same -ef file]	file 'same' and file 'file' refer to the same device/inode number.	if [/dev/cdrom -ef /dev/dvd]; then echo "Your primary cd drive appears to read dvd's, too." fi

String-based conditions

Condition	True if	Example/explanation
[STRING1 == STRING2]	STRING1 is equal to STRING2.	if ["$1" == "moo"]; then echo $cow # *Ever tried executing 'apt-get moo'?* fi Note: you can also use a single "=" instead of a double one.

[STRING1 != STRING2]	STRING1 is not equal to STRING2.	if ["$userinput" != "$password"]; then
		  echo "Access denied! Wrong password!"
		  exit 1 # *Stops script execution right here*
		fi

[STRING1 \> STRING2]	STRING1 sorts after STRING2 in the current locale (lexographically).	The backslash before the angle bracket is there because the bracket needs to be escaped to be interpreted correctly. As an example we have a basic bubble sort:
		
		(Don't feel ashamed if you don't understand this, it is a more complex example)
		
		array=(linux tutorial blog)
		swaps=1
		while ((swaps > 0)); do
		
		swaps=0
		  for ((i=0; i < ((${#array[@]} - 1)) ; i++)); do
		  if ["${array[$i]}" \> "${array[$((i + 1))]}"];
		then # *Here is the sorting condition*
		  tempstring=${array[$i]}
		  array[$i]=${array[$((i + 1))]}
		  array[$((i + 1))]=$tempstring
		  ((swaps=swaps + 1))
		  fi
		  done
		done
		echo ${array[@]} # *Returns "blog linux tutorial"*

[STRING1 \< STRING2]	STRING1 sorts before STRING2 in the current locale (lexographically).	

[-n NONEMPTYSTRING]	NONEMPTYSTRING has a length of more than zero.	This condition only accepts valid strings, so be sure to quote anything you give to it. if [-n "$userinput"]; then userinput=parse($userinput) # *Only parse if the user actually gave some input.* fi

		Note that you can also omit the "-n", as brackets with just a string in it behave the same.
[-z EMPTYSTRING]	EMPTYSTRING is an empty string.	This condition also accepts non-string input, like an uninitialized variable: if [-z $uninitializedvar]; then uninitializedvar="initialized" # *-z returns true on an uninitialized variable, so we initialize it here.* fi
Double-bracket syntax only: [[STRING1 =~ REGEXPATTERN]]	STRING1 matches REGEXPATTERN.	If you are familiar with Regular Expressions, you can use this conditions to perform a regex match. if [["$email" =~ "\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,4}\b"]]; then echo "\$email contains a valid e-mail address." fi

Arithmetic (number-based) conditions:

Condition	True if	Example/explanation
[NUM1 -eq NUM2]	NUM1 is **EQ**ual to NUM2.	These conditions only accept integer numbers. Strings will be converted to integer numbers, if possible. Some random examples: if [$? -eq 0]; then # *$? returns the exit status of the previous command*
[NUM1 -ne NUM2]	NUM1 is **N**ot **E**qual to NUM2.	
[NUM1 -gt NUM2]	NUM1 is **G**reater **T**han NUM2.	

[NUM1 -ge NUM2]	NUM1 is **G**reater than or **E**qual to NUM2.	echo "Previous command ran succesfully."
		fi
		
		if [$(ps -p $pid -o ni=) -ne $(nice)]; then
		echo "Process $pid is running with a non-default nice value"
		fi
		
		if [$num -lt 0]; then
		echo "Negative numbers not allowed; exiting..."
		exit 1
		fi

[NUM1 -lt NUM2]	NUM1 is **L**ess **T**han NUM2.	
[NUM1 -le NUM2]	NUM1 is **L**ess than or **E**qual to NUM2.	