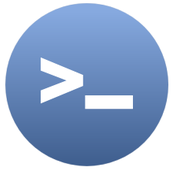
IF statements

INTRODUCTION TO BASH SCRIPTING



Alex Scriven
Data Scientist

A basic IF statement

A basic IF statement in Bash has the following structure:

```
if [ CONDITION ]; then
    # SOME CODE
else
    # SOME OTHER CODE
fi
```

Two Tips:

- Spaces between square brackets and conditional elements inside (first line)
- Semi-colon after close-bracket `];`

IF statement and strings

We could do a basic string comparison in an IF statement:

```
x="Queen"
if [ $x == "King" ]; then
    echo "$x is a King!"
else
    echo "$x is not a King!"
fi
```

```
Queen is not a King!
```

You could also use `!=` for 'not equal to'

Arithmetic IF statements (option 1)

Arithmetic IF statements can use the double-parenthesis structure:

```
x=10
if (($x > 5)); then
    echo "$x is more than 5!"
fi
```

```
10 is more than 5!
```

Arithmetic IF statements (option 2)

Arithmetic IF statements can also use square brackets and an arithmetic flag rather than (`>` , `<` , `=` , `!=` etc.):

- `-eq` for 'equal to'
- `-ne` for 'not equal to'
- `-lt` for 'less than'
- `-le` for 'less than or equal to'
- `-gt` for 'greater than'
- `-ge` for 'greater than or equal to'

Arithmetic IF statement example

Here we re-create the last example using square bracket notation:

```
x=10
if [ $x -gt 5 ]; then
    echo "$x is more than 5!"
fi
```

```
10 is more than 5!
```

Other Bash conditional flags

Bash also comes with a variety of file-related flags such as:

- `-e` if the file exists
- `-s` if the file exists and has size greater than zero
- `-r` if the file exists and is readable
- `-w` if the file exists and is writable

And a variety of others:

- https://www.gnu.org/software/bash/manual/html_node/Bash-Conditional-Expressions.html

Using AND and OR in Bash

To combine conditions (AND) or use an OR statement in Bash you use the following symbols:

- `&&` for AND
- `||` for OR

Multiple conditions

In Bash you can either chain conditionals as follows:

```
x=10
if [ $x -gt 5 ] && [ $x -lt 11 ]; then
    echo "$x is more than 5 and less than 11!"
fi
```

Or use double-square-bracket notation:

```
x=10
if [[ $x -gt 5 && $x -lt 11 ]]; then
    echo "$x is more than 5 and less than 11!"
fi
```

IF and command-line programs

You can also use many command-line programs directly in the conditional, removing the square brackets.

For example, if the file `words.txt` has 'Hello World!' inside:

```
if grep -q 'Hello' words.txt; then
    echo "Hello is inside!"
fi
```

```
Hello is inside!
```

IF with shell-within-a-shell

Or you can call a shell-within-a-shell as well for your conditional.

Let's rewrite the last example, which will produce the same result.

```
if $(grep -q 'Hello' words.txt); then
    echo "Hello is inside!"
fi
```

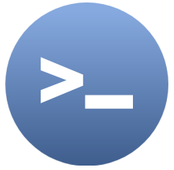
```
Hello is inside!
```

Let's practice!

INTRODUCTION TO BASH SCRIPTING

FOR loops & WHILE statements

INTRODUCTION TO BASH SCRIPTING



Alex Scriven
Data Scientist

Basic FOR Loop structure

Python:

```
for x in range(3):
    print(x)
```

```
0
1
2
```

R:

```
for (x in seq(3)){
    print(x)
}
```

```
[1] 1
[1] 2
[1] 3
```

FOR Loop in Bash

The basic structure in Bash is a similar:

```
for x in 1 2 3
do
    echo $x
done
```

```
1
2
3
```

FOR loop number ranges

Bash has a neat way to create a numeric range called 'brace expansion':

- `{START..STOP..INCREMENT}`

```
for x in {1..5..2}
do
    echo $x
done
```

```
1
3
5
```

FOR loop three expression syntax

Another common way to write FOR loops is the 'three expression' syntax.

- Surround three expressions with double parenthesis
- The first part is the start expression (`x=2`)
- The middle part is the terminating condition (`x<=4`)
- The end part is the increment (or decrement) expression (`x+=2`)

```
for ((x=2;x<=4;x+=2))
do
    echo $x
done
```

```
2
4
```

Glob expansions

Bash also allows pattern-matching expansions into a for loop using the `*` symbol such as files in a directory.

For example, assume there are two text documents in the folder `/books` :

```
for book in books/*
do
    echo $book
done
```

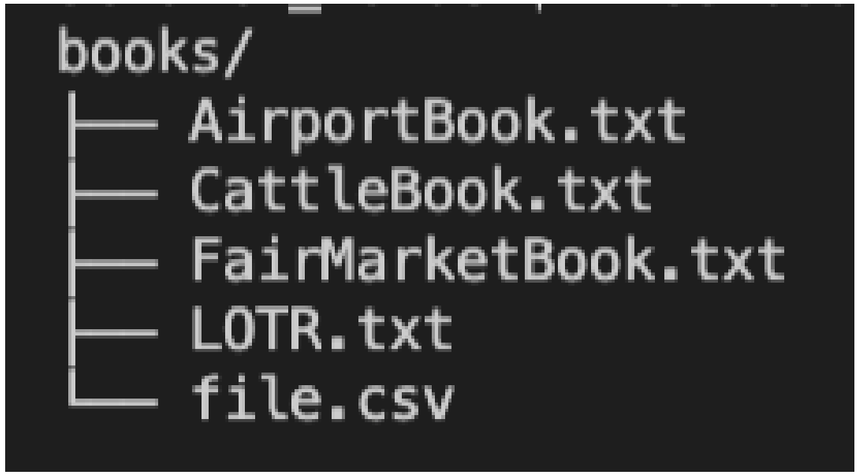
```
books/book1.txt
books/book2.txt
```

Shell-within-a-shell revisited

Remember creating a shell-within-a-shell using `$()` notation?

You can call in-place for a for loop!

Let's assume a folder structure like so:



Shell-within-a-shell to FOR loop

We could loop through the result of a call to shell-within-a-shell:

```
for book in $(ls books/ | grep -i 'air')
do
    echo $book
done
```

```
AirportBook.txt
FairMarketBook.txt
```

WHILE statement syntax

Similar to a FOR loop. Except you set a condition which is tested at each iteration.

Iterations continue until this is no longer met!

- Use the word `while` instead of `for`
- Surround the condition in square brackets
  - Use of same flags for numerical comparison from IF statements (such as `-le`)
- Multiple conditions can be chained or use double-brackets just like 'IF' statements along with `&&` (AND) or `||` (OR)
- Ensure there is a change inside the code that will trigger a stop (else you may have an infinite loop!)

WHILE statement example

Here is a simple example:

```
x=1
while [ $x -le 3 ];
do
    echo $x
    ((x+=1))
done
```

```
1
2
3
```

Beware the infinite loop

Beware the infinite WHILE loop, if the break condition is never met.

```
x=1
while [ $x -le 3 ];
do
    echo $x
    # don't increment x. It never reaches 3!
    # ((x+=1))
done
```

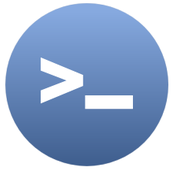
This will print out 1 forever!

Let's practice!

INTRODUCTION TO BASH SCRIPTING

CASE statements

INTRODUCTION TO BASH SCRIPTING



Alex Scriven
Data Scientist

The need for CASE statements

Case statements can be more optimal than IF statements when you have multiple or complex conditionals.

Let's say you wanted to test the following conditions and actions:

- If a file contains `sydney` then move it into the `/sydney` directory
- If a file contains `melbourne` or `brisbane` then delete it
- If a file contains `canberra` then rename it to `IMPORTANT_filename` where `filename` was the original filename

A complex IF statement

You could construct multiple IF statements like so:

- This code calls `grep` on the first ARGV argument for the conditional.

```
if grep -q 'sydney' $1; then
    mv $1 sydney/
fi
if grep -q 'melbourne|brisbane' $1; then
    rm $1
fi
if grep -q 'canberra' $1; then
    mv $1 "IMPORTANT_$1"
fi
```

- Seems complex and repetitious huh?

Build a CASE statement

- Begin by selecting which variable or string to match against
  - You could call shell-within-a-shell here!
- Add as many possible matches & actions as you like.
  - You can use regex for the `PATTERN`. Such as `Air*` for 'starts with Air' or `*hat*` for 'contains hat'.
- Ensure to separate the pattern and code to run by a close-parenthesis and finish commands with double semi-colon

Basic CASE statement format:

```
case 'STRINGVAR' in
  PATTERN1)
  COMMAND1;;
  PATTERN2)
  COMMAND2;;
```

Build a CASE statement

- `*) DEFAULT COMMAND;;`
  - It is common (but not required) to finish with a default command that runs if none of the other patterns match.
- `esac` Finally, the finishing word is 'esac'
  - This is 'case' spelled backwards!

Basic CASE statement format:

```
case 'STRING' in
  PATTERN1)
  COMMAND1;;
  PATTERN2)
  COMMAND2;;
  *)
  DEFAULT COMMAND;;
esac
```

From IF to CASE

Our old IF statement:

```
if grep -q 'sydney' $1; then
    mv $1 sydney/
fi

if grep -q 'melbourne|brisbane' $1; then
    rm $1
fi

if grep -q 'canberra' $1; then
    mv $1 "IMPORTANT_$1"
fi
```

Our new CASE statement:

```
case $(cat $1) in
  *sydney*)
  mv $1 sydney/ ;;
  *melbourne*|*brisbane*)
  rm $1 ;;
  *canberra*)
  mv $1 "IMPORTANT_$1" ;;
  *)
  echo "No cities found" ;;
esac
```

Let's practice!

INTRODUCTION TO BASH SCRIPTING