Lab 6 - Loops in Bash

Loops in Shell Scripting

Similar to C/C++ and Python, Bash also has commands to perform repetitive tasks, these are called loops.

In bash, there are primarily for types of loops:

- for Loop
- while Loop
- until Loop [We won't be discussing this]
- select Loop [We won't be discussing this]

For Loop:

In Bash scripting, the `for` loop is a control structure used for iterating over a range of values, a list of items, or a series of commands.

There are primary two ways `for` loops can be used in bash:

1. Iterating over a list of items
2. Iterating over a given range

1. Iterating over a list of items:

The following syntax can be used to iterate through a list of items:

```
for i in list
do
  <perform-tasks>
done
```

The `do` keywords marks the beginning of loop and `done` marks the end of the loop.

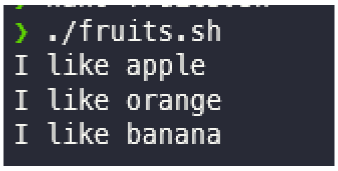
The list in bash have to `space-seperated` in order to work properly with loops.

In order to understand this, consider the following example:

→ Given a list of fruits, print all the values by prefixing with `I like`:

```
#!/bin/bash

fruits="apple orange banana"
for fruit in $fruits; do
  echo "I like $fruit"
done
```

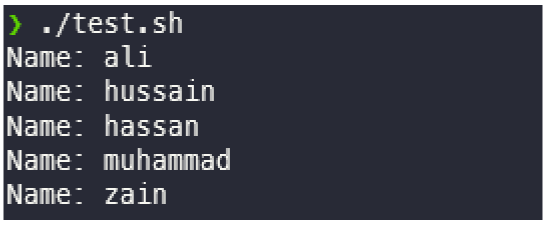


```
❯ ./fruits.sh
I like apple
I like orange
I like banana
```

→ A list can also be created by adding the `space-seperated` data inside a file, reading the file into a variable and iterating over that variable.

```
#!/bin/bash

## -n to prevent a newline from being added at the end
echo -n "ali hussain hassan muhammad zain" > names.txt
for name in $(cat names.txt) # This can also work as we're reading directly from file.
do
  echo "Name: $name"
done
```



```
❯ ./test.sh
Name: ali
Name: hussain
Name: hassan
Name: muhammad
Name: zain
```

2. Iterating over a given range:

One expansion that we've yet to talk about is arthimetic expansion.

Arithmetic Expansion

In Bash scripting, arithmetic expansion is a feature that allows you to perform arithmetic operations within double parentheses `(( ... ))` or using the `$(( ... ))` syntax. You can use this feature to perform various arithmetic operations on variables and values. Here are some examples of arithmetic expansion with explanations:

1. **Increment `i` by 1 (`i++`):**

```
bashCopy code
i=5
((i++))
echo "Incremented i: $i"
```

In this example, we start with `i=5`. The `((i++))` expression increments `i` by 1, and the result is `i=6`.

2. **Add `j` to `i` (`i+=j`):**

```
i=5
j=3
((i+=j))
echo "i += j: $i"
```

Here, `i` is initially set to 5, and `j` is set to 3. The `((i+=j))` expression adds the value of `j` to `i`, resulting in `i=8`.

3. **Add a constant value to `i` (`i+10`):**

```
i=5
((i+10))
echo "i + 10: $i"
```

In this case, the `((i+10))` expression calculates the result of adding 10 to the current value of `i`, but it doesn't change the value of `i`. The value of `i` remains 5.

4. **Calculate the remainder of `i` divided by 10 (`i%10`):**

```
i=27
((i%=10))
echo "i % 10: $i"
```

Here, `i` is initially set to 27. The `((i%=10))` expression calculates the remainder of `i` divided by 10 and assigns the result back to `i`. The value of `i` is updated to 7 because 27 divided by 10 leaves a remainder of 7.

Now, in this loop, we utilize the power of arthimetic expansion and repition to use a `C-style` for loop which is easier to understand. The syntax for this loop is as follows:

```
for (( initialization; condition; step )); do
    # Commands to be executed in each iteration
done
```

- `initialization`: This is where you initialize a counter variable. It can also be used to set the starting value of the counter.
- `condition`: This is the condition that determines whether the loop should continue. The loop will run as long as this condition is true.
- `step`: This is used to update the counter variable in each iteration. You can increment or decrement the counter here.

Example

Consider you want to print all number in the range of `0` to `10`

```
for (( i=0; i<10; i++ )); do
    echo "Current number: $i"
done
```

> In the previous syntax of For Loop that we talked about, we can perform this same range-based iteration by using the following: `for i in $(seq 0 10)`

Class Task - 1:

Write a simple bash script that asks the user for a number and then, starting from 0, prints all the numbers upto that number.

---

while Loop:

Similar to C/C++/Python, a `while` loop runs as long as a specified condition remains true. The syntax for the while loop is as follows:

```
while [condition]
do
  # Tasks to be performed
done
```

Considering an example of this syntax: If we want to print all numbers from 1 to 10, we'll use the following loop:

```
#!/bin/bash

count=1

while [[ $count -le 10 ]]; do
  echo "Count is: $count"
  count=$((count+1))
done

echo "After loop, count is: $count"
```

Similar to the `if` statements that we read eariler, the condition in `while` loop are the same.

BONUS:

Compiling your C/C++ code in Linux:

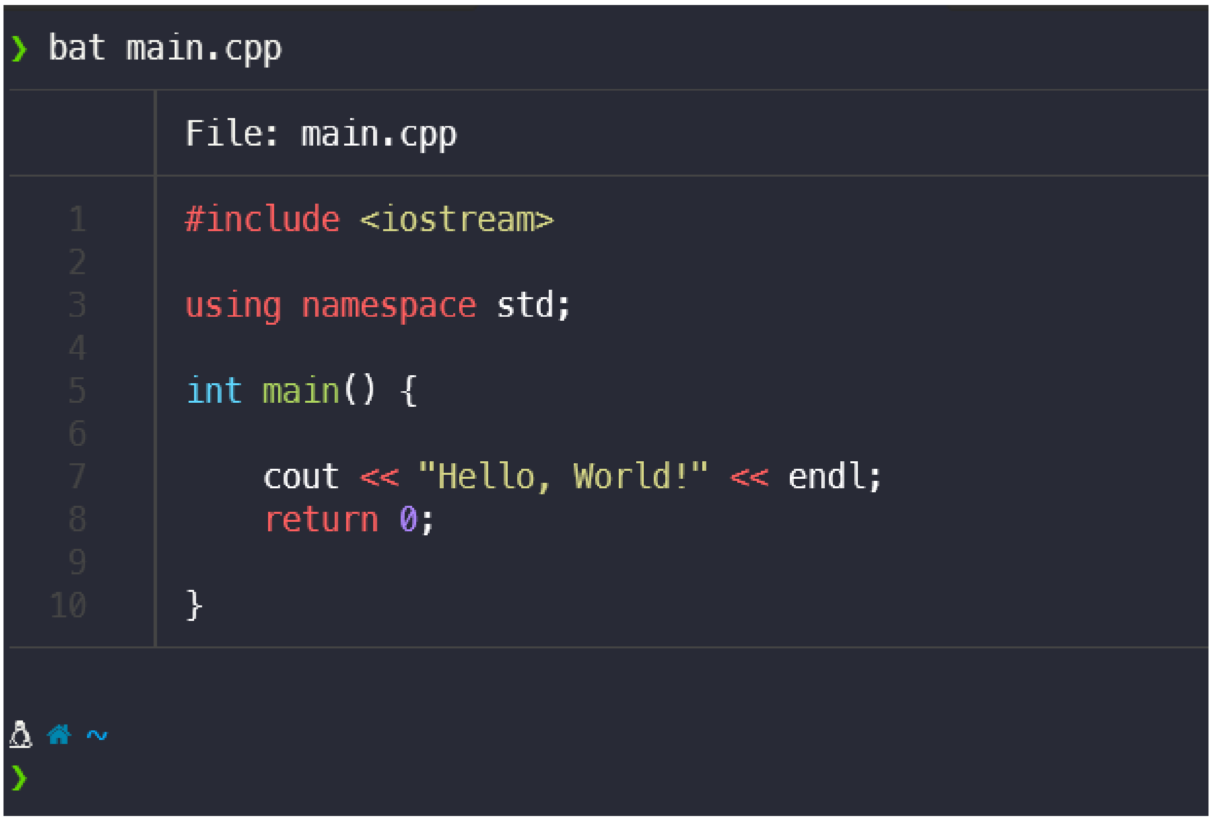
We'll take a look at how you can compile your C/C++ code in Linux by making use of `gcc` and `g++` compilers.

In case you don't have these installed, use the following commands in order to install them on your Ubuntu/Kali:

```
sudo apt update && sudo apt install gcc g++
```

Once that's done, you need to create a new file called `main.cpp` that will contain the following source code:

```
#include <iostream>
using namespace std;
int main() {
  cout << "Hello, World!" << endl;
  return 0;
}
```



```
❯ bat main.cpp
       File: main.cpp

       #include <iostream>

       using namespace std;

       int main() {

           cout << "Hello, World!" << endl;
           return 0;

       }

❯
```

👉 You might not have the `bat` command installed. If you want to install it, you can follow this github page: https://github.com/sharkdp/bat

Once this is done, you need to compile it using `g++`.

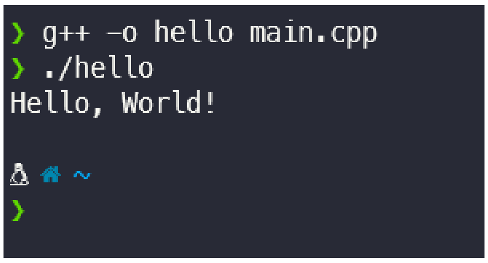
👉 **g++** is **the traditional nickname of GNU C++, a freely redistributable C++ compiler produced by the Free Software Foundation plus dozens of skilled volunteers**.

The command for compilation is as follows:

```
g++ -o <output-file-name> <c/cpp-file-name>
## Example:
g++ -o hello main.cpp
```

Once the program gets compiled without any errors/warnings, we can simply run it using:

```
./<output-file-name>
## Example:
./hello
```



```
❯ g++ -o hello main.cpp
❯ ./hello
Hello, World!

❯
```
