# HOP05 – BASH – Looping and branching

## CS 340 – Operating Systems

**School of Technology & Computing (STC) @ City University of Seattle (CityU)**

```
7/13/2019 Developed by Kevin Wang
06/17/2020 Reviewed by Kim Nguyen
03/29/2021 Updated by Matt Raio
02/14/2022 Reviewed by Ken Ling
```


### Before You Start

- This exercise assumes that the user is working with the Linux distribution in Virtualbox
- All commands and code discussed in this exercise will run in the Ubuntu console.
- The directory path shown in the screenshots may be different from yours.
- Some steps are not explained in the tutorial. If you are not sure what to do:
    - Consult the resources listed below, experiment in the Ubuntu console, and try to solve the problem yourself.
    - If you cannot solve the problem after a few tries, ask a TA for help.

### Learning Outcomes

- Students will be able to:
    - Use test command
    - Understand the conditional execution
    - Use input/Output, pipeline, and command substitution

### Resources

- Linux command line: bash + utilities
- Nano/Basics Guide

### Preparation

1. Connect to your Ubuntu instance

---

## Basic usage of the test command

1. Type the following command to test integers:

```
test 1 -eq 1
```

| code | Meaning |
|---|---|
| `-eq` | equal |
| `-ne` | not equal |
| `-gt` | greater than |
| `-lt` | less than |
| `-ge` | greater or equal |
| `-le` | less or equal |

2. Check the result by typing:

   - `$?` is a special parameter that contains the exit code for the last command.
   - `0` means successful when `1` means error.
   - When we use a test command, `0` means `true` and `1` means `false`.

```
echo $?
```

3. `[]` is the equivalent of test. We can test an integer like this:

```
[ 1 -lt -2 ]
And then, retrieve the result:
echo $?
```

   - Note: Linux Bash script is case sensitive as well as space sensitive.
   - For using the `[]`, we have to keep a space after `[` and before `]`.

4. Use `[[]]` to test a regular expression

```
[[abc =~ ^a.*c$]]
And retrieve the result
echo $?
```

- Note: We will practice more regular expression in other modules.

5. Use (()) to evaluate an arithmetic expression. (This is a nonstandard feature)

```
(( 2 - 1 ))
echo $?
(( 1 - 1 ))
echo $?
```

In Linux, when arithmetic expression value is `0`, it will return `False`. Otherwise, a `True` will be returned.

## If conditional execution

1. The basic syntax of if command shows below:

```
if <condition list>
then
<list>
elif <condition list>
then
<list>
else
<list>
fi
```

2. Create a `ConditionTest.sh` file by typing following command:

```
nano ConditionTest.sh
```

3. Type the following script in the file:

```
#! /bin/bash
printf "How many pizzas do you want to buy?\n"
read amount

if (( amount <= 0 ))
then
   printf "Please enter a number that is greater than 0.\n"
elif (( amount > 10 ))
then
   printf "Sorry, we do not have that many pizzas.\n"
   exit 1
else
   printf "There you go. Here are %d pizzas for you.\n" "$amount"
fi
```

- Hit the `CTRL + x` key to quit and save the file.

4. Type the following command to execute the script:

```
bash ConditionTest.sh
```

> Then type in a number to test it. Please run the program several times to try different numbers.

## Case condition execution

1. The basic syntax for `case`:

```
case WORD in
PATTERN) COMMANDS ;;
PATTERN) COMMANDS ;; ## optional
esac
```

2. Create a `CaseTest.sh` file by typing the following command:

```
nano CaseTest.sh
```

3. Type the following script in the file:

```
#! /bin/bash
case $1 in
    *[!0-9]*) message="non-numeric character(s).";;
    *) message="a number.";;
esac

printf "You entered %s\n" "$message"
```

   - Hit the `control + x` key to quit and save the file.

4. Type the following command to see the result

```
bash CaseTest.sh 10
bash CaseTest.sh string
```

## Looping

1. Type the following command to create a WhileTest.sh file:

```
nano WhileTest.sh
```

2. Type the following script in the file:

```
#! /bin/bash
n=1;
while [ $n -le $1 ]
do
  printf "The loop has already run %d times.\n" "$n"
  n=$(( $n + 1 ))
done
```

   - Click the `control + x` key to quit and save

3. Type the following command to test the file. ***You can give a different number to the parameter***.

```
bash WhileTest.sh 10
```

4. An `Until` loop is the opposite of `while` loop, which will run the code as long as the condition fails.

   - Type the following command to create a `UntilTest.sh` file:

```
nano UntilTest.sh
```

   - Click the `control + x` key to quit and save.

5. Type the following command to `test` the file:

```
bash UntilTest.sh 5
```

6. Another common loop is `for` loop. Type the following command to create a `ForTest.sh` file:

```
nano ForTest.sh
```

7. Type the following script in the file:

```
#! /bin/bash
n=1;
until [ $n -gt $1 ]
do
  printf "The loop has already run %d times.\n" "$n"
  n=$(( $n + 1 ))
done
```

   - Click the `control + x` key to quit and save.

8. Type the following command to `test` the file:

```
 bash ForTest.sh
```

# Challenge

1. Take substitute variables for a random number and run the ConditionTest.sh again. Take a screenshot.
2. Change the condition in ForTest loop to output a different print statement that shows how many times the loop ran. Take a screenshot.
3. Submit your Work to Brightspace
    - Please upload all your files for this hands-on practice to the HOP assignment on Brightspace.