**BASH**

**Conventions**

* **# -** Prompts execution of commands with **root privileges**, either with **default root user** or with ***sudo***.
* **$ -** Executes commands as a **non-privileged user**.

**Getting Location of Bash**

***$ which bash***

**First Script**

***#!usr/bin/bash***

***str="Hello, World!"***

***echo $str***

* **#!** is called ***shebang***.
* When it’s a **shebang**, it is **not** read as a comment.
* There is **no space** around assignation operator.

**Script Execution**

***$ bash filename.sh***

**Global v/s Local Variables**

* Just like most programming & scripting languages, **global variables** in Bash are declared **outside** the function.
* And **local variable** is declared **inside** a function.

Function in Bash:-

***#!usr/bin/bash***

***function my\_func {***

***var="local"***

***echo $var***

***}***

***my\_func # Calling the function***

**Echo Mischiefs**

***#!usr/bin/bash***

***echo $1 $2 $3 # Statement 1***

***echo '$1 $2 $3' # Statement 2***

***echo "$1 $2 $3" # Statement 3***

***1 2 3***

* Only **statement 2** is executed successfully, notice it between **single inverted commas**.
* While **statement 1** & **3** doesn’t.

**Passing Arguments**

***#!usr/bin/bash***

***args=("$@")***

***echo ${args[0]} ${args[1]} ${args[2]} ${args[3]} # Line 1***

***echo $@ # Line 2***

***echo $# # Line 3***

* **Line 1 –** Prints first 4 arguments passed with command **“bash test.sh”**.
* **Line 2 –** Prints **all** arguments passed.
* **Line 3 –** Prints the **number of arguments** passed.

**Echoing Terminal Commands**

***#!usr/bin/bash***

***echo $(uname -o) # Line 1***

***echo uname -o # Line 2***

* **Line 1 –** Prints **GNU/Linux** i.e. the name of the operating system.
* **Line 2 –** Prints literally **uname -o**.

**Taking User Inputs**

***#!usr/bin/bash***

***echo "Enter your age: "***

***read age***

***echo "$age"***

***echo "Enter your full name: "***

***read name sur #Todd Baker***

***echo "Name: $name, Surname: $sur" #Name: Todd, Surname: Baker***

***echo "Enter your full name: "***

***read name sur #Gourav Kumar Mallick***

***echo "Name: $name, Surname: $sur" #Name: Gourav, Surname: Kumar Mallick***

**Default Read Variable**

***echo "Say something: "***

***read #Stores passed argument in REPLY variable***

***echo "You said $REPLY"***

***Error Skipping***

***echo -e "Write here: "***

***read -e word***

* ***-e*** is used to **jump** to next statement during execution, if any **error** is caused in the line it is written at.

**Reading Arguments Into Array**

***echo "List some fruits: "***

***read -a fruits***

***echo "${fruits[0]}, ${fruits[1]}, ${fruits[2]}"***

* We named the **array** as **food** & **-a** was used to tell Bash that it is an **array**.

**Bash Trap Command**

***#!usr/bin/bash***

***trap bashtrap INT***

***clear;***

***bashtrap()***

***{***

***echo "\"Ctrl + C\" is detected!"***

***}***

* ***bashtrap*** is the variable name we gave to our **trap**, of type **INT** (**interrupt**).
* (**;**) is used to write **multiple commands** on same line.

**Difference In Bash Shell Commands**

***#!usr/bin/bash***

***ls #Line 1***

***echo "$(ls)" #Line 2***

* **Line 1 –** **Direct** & makes some **proper indentation** to shell texts.
* **Line 2 –** Comparatively **less efficient**.

**Array**

***ARR=('Ubuntu' 'Debian' Arch Mint) # Line 1***

***TOTAL=${#ARR[@]} # Gives total elements in ARR***

***for ((i=0; i<$TOTAL; i++)); do # Notice use of semicolon***

***echo ${ARR[$i]}***

***done***

***Ubuntu***

***Debian***

***Arch***

***Mint***

**Declaration**

***declare -a ARR #Array declaration***

***let COUNT=0 #Integer declaration***

***let COUNT=COUNT-1***

***NUM=0 #Declaration but requires extra efforts when being used***

**Conditional Statements**

***if [ $X -lt -1 ]***

***then***

***echo "Negative number."***

***else***

***echo "Semi-positive"***

***fi***

* We use ***elif*** statement in **Bash**.

**Comparisons**

|  |  |
| --- | --- |
| **Arithmetic Comparison** | **String Comparison** |
| **-lt** | **<** |
| **-gt** | **>** |
| **-le** |  |
| **-ge** |  |
| **-eq** | **=** |
| **-ne** | **!=** |
|  | **-n STR** |
|  | **-z STR** |

* ***-n STR*** means string ***STR*** **isn’t** empty.
* ***-z STR*** means string ***STR*** is empty.

**File Testing**

|  |  |
| --- | --- |
| **-b filename** | **Block special file.** |
| **-c filename** | **Special character file.** |
| **-d directory\_name** | **Check for directory existence.** |
| **-e filename** | **Check for file existence.** |
| **-f filename** | **Check for regular file existence & not directory.** |
| **-G filename** | **Check if file exists & owned by effective group ID.** |
| **-g filename** | **True if file exists & group ID is set.** |
| **-k filename** | **Sticky bit.** |
| **-L filename** | **Symbolic bit.** |
| **-O filename** | **True if file exists & owned by effective user ID.** |
| **-r filename** | **Check if file is readable.** |
| **-S filename** | **Check if file is socket.** |
| **-s filename** | **Check if file is of non-zero size.** |
| **-u filename** | **Check if file has set user ID bit.** |
| **-w filename** | **Check if file is writable.** |
| **-x filename** | **Check if file is executable.** |

Example 1:-

***#!usr/bin/bash***

***FILE="MyCodes/test.cpp"***

***if [ ! -e $FILE ] # If file doesn’t exist***

***then***

***echo "File not found!"***

***fi***

**Sleep**

***sleep 1 # Program sleeps for 1 second***

**Loops**

For loop:-

***#!usr/bin/bash***

***for f in ( ls MyCode/)***

***do***

***echo $f***

***done***

Until loop:-

* **Until** loop is **opposite** of **while** loop, it is executed until the condition **doesn’t** become **true**.

***#!usr/bin/bash***

***COUNT=0***

***until [ $COUNT -eq 5 ]***

***do***

***echo $COUNT***

***((COUNT++)) # Increments value of COUNT***

***done***

**An Intermediate Bash Script**

***#!usr/bin/bash***

***DIR="."***

***find $DIR -type f | while read FILE # Line 1***

***do***

***if [[ "$FILE" = \*[[:space:]]\* ]] # Line 2***

***then***

***mv "$FILE" `echo $FILE | tr " " "\_" # Line 3***

***fi***

***done***

* In **line 1**, **DIR** with value **"."** represents **all files** in the directory.
* In **line 2**, **\*[[:space:]]\*** represents if any file contains **at least one white line** character.
* **Line 3** replaces **space ( )** in file with **underscore (\_)**.

**Functions**

***#!usr/bin/bash***

***function my\_func***

***{***

***echo $2 # Prints second argument***

***}***

***my\_func "Gourav" "Aarav" "Papa" "Mamma" # These are arguments passed***

***Aarav***

**Selection**

***#!usr/bin/bash***

***select WORD in "Mauritius" "Quebec" "Switzerland" "Chicago"***

***do***

***echo "Selected word: $WORD"***

***break***

***done***

***exit 0***

***1) Mauritius***

***2) Quebec***

***3) Switzerland***

***4) Chicago***

***#? 2***

***Selected word: Quebec***

**Cases**

***#!usr/bin/bash***

***read WORD***

***case $WORD in***

***1) echo "Alto’s Odyssey"***

***2) echo "Riptide"***

***3) echo "Pacman"***

***4) exit***

***esac***

* **1, 2, 3 etc** are patterns which as input ***WORD***, affect which case to be executed.
* It can be replaced with other **characters**, or even **words**.

**Quotes & Quotations**

* ***Meta characters*** like **$** can be escaped with **\** just like **C/C++**.
* Even single quotes **' '** can be used to enclose whole string, keeping meta characters **literally**.
* Double quotes **" "** takes all meta characters **literally**, except **$**, **\** and **`**.

ANSI-C style quoting:-

|  |  |
| --- | --- |
| **Escape Character** | **Meaning** |
| **\a** | **Alert sound** |
| **\e** | **Escape character** |
| **\nnn** | **Octal value of characters** |
| **\f** | **Form feed** |
| **\r** | **Carriage return** |
| **\v** | **Vertical tab** |
| **\xnn** | **Hexadecimal value of characters** |

**Arithmetic Operations**

***#!usr/bin/bash***

***NUM=43***

***NUM=22***

***let SUM1=$NUM1+$NUM2***

***echo "$SUM1 = $NUM1 + $NUM2" # 65 = 43 + 22***

***declare -i SUM2***

***SUM2=$NUM1+$NUM2***

***echo "$SUM2 = $NUM1 + $NUM2" # 65 = 43 + 22***

***echo "$NUM1 + $NUM2" # 43 + 22***

***echo "$(($NUM1 + $NUM2))" # 65***

* Its ***-i*** after ***declare*** because it means the declared variable is an **integer**.
* Otherwise, it may work as **string** if **not** defined explicitly.
* Notice the difference between ***let*** & ***declare*** clearly above.
* We can use other operators as well, including **modulus** & **\*\*** for **power**.

**Base Conversions**

***#!usr/bin/bash***

***echo "Binary 1001 in decimal is $((2#1001))"***

***echo "Octal 36 in decimal is $((8#36))"***

***echo "Hex EC3 in decimal is $((16#EC3))***

* The **capital** alphabetical digits in **hexadecimal** can also be replaced with **smaller** ones, and even can be written in **mixed** form.

**Rounding Off Floats**

***#!usr/bin/bash***

***FLOAT=3.14***

***for ROUND\_OFF in $(printf %.0f $FLOAT)***

***do***

***echo "$FLOAT rounded off is $ROUND\_OFF"***

***done***

**Bash Calculator**

***#!usr/bin/bash***

***echo "Enter something to calculate: "***

***read INPUT***

***echo "Result with 3 digits after decimal point: "***

***echo "scale=3; ${INPUT}" | bc***

* The **semicolon** after scale is important, which when **not** being there can cause error.