Bash Scripting 1

PREPARED BY ENES ERTEN

Contents

- Core Concepts
- Conditional Statements
- Input and Output

Core Concepts

What is Bash script

A Bash script is a plain text file which contains a series of commands. These commands are a mixture of commands we would normally type on the command line (such as ls or cp for example) and commands we could type on the command line but generally wouldn't . An important point to remember though is:

Anything you can run normally on the command line can be put into a script and it will do exactly the same thing. Similarly, anything you can put into a script can also be run normally on the command line and it will do exactly the same thing.

Core Concept

- An example script

```
#/bin/bash

echo "This Script will give us environment information"
echo "============================================="
echo "Hello Username: $USER"
echo ""
echo "Your home directory is: $HOME"
echo ""
echo "Your History File will Ignore: $HISTCONTROL"
echo ""
echo ""
```

Core Concepts

- echo command:

  Prints out the variable expression or anything

  # echo "$USER"

  # echo "hello world"
- # used for comments line
- All bash scripts starts with #!/bin/bash comment line to dictating the script language for example python scripts #!/bin/python.

Core Concepts

- Declaring Variable: Variable names mostly declared in uppercase letters.

  # VAR="enes"

- A variable reached with $ sign

  # echo "$VAR"

Core Concepts

- To assign a command to the variable done by `command`

  # VAR=`date`

- export keyword with a variable is used to set an environment variable

  # export VAR=`date`

Core Concepts

- Setting a file path to PATH variable is mostly used for projects to directly access to bash scripts like commands

  export PATH="$PATH":FILEPATH

Core Concepts

- Exit status: According to the script run or commands run successfully or not exit status settled generally 0 exit status indicates command or script execute successfully other situations means error. #$? Commands will print out the last command or script exit status.

  exit 0 -> can be used for script is successfully worked.

  exit # -> can be used for error for script error occurred.

Core Concepts

- Arithmetic Operations: done by expr command

  # expr 2 + 2

  # expr 2 + 2 – 1

  # expr 10 / 2

  # expr 10 * 5

  # expr 15 % 4

  # expr \(2 + 2 \) * 4

Core Concepts

- /dev/null : this file location is used for empty all things in Linux systems. In bash scripting generally used for redirection location of standard error.

  # error 2> /dev/null

- read: is used for reading values from command line

  # read VAR

Core Concepts

- Array: Is a data structure consisting of a collection of elements, each identified by at least one array index or key. An array is stored such that the position of each element can be computed from its index tuple by a mathematical formula.

  # MYARR=("FIRST" "SECOND" "THIRD")

  # "{$MYARR[#]}" -> reaching #th index element

  # "{$MYARR[*]}" -> reaching whole elements

Core Concepts

- Passing variable to Scripts on the command line: variables can be passed to bash scripts when interpret the script it can be done # /path/script.sh VAR1 VAR2 in bash scripts that values can be used $1 $2 in orderly

Core Concepts

- In Bash, you can run multiple commands based on the following format: <Command> <option> <Command

  && Run the following command only if the previous succeeds or has no errors

  | | Run the following command only if the previous fails or results in error

Conditional Statements

- if [condition];
  then
    #commands to be run if true
  else
    #commands to be run if false
  fi

Conditional Statements

- if [condition];
  then
  #commands to be run if true
  elif [condition];
  then
  #commands to be run if true
  else
  #commands to be run if false
  fi
- if [condition] OPERATOR [condition];
  if [condition] || [condition];
  if [$g == 1 && $c == 123] || [$g == 2 && $c == 456];
  if [[(Condition) OPERATOR (Condition)]];
  if [[(Condition) || (Condition)]];
  if [[($g == 1 && $c == 123) || ($g == 2 && $c == 456)]];

Conditional Statements

- case "$VAR" in
  pattern_1)
  # Commands to be executed
  ;;
  pattern_2)
  # Commands to be executed
  ;;
  *)
  # Default
  ;;
  esac

Conditional Statements

- File Tests
  -a <FILE> : True if <FILE> exists, but may cause conflicts
  -e <FILE> : True if <FILE> exists
  -f <FILE> : True if <FILE> exists and is a regular file
  -d <FILE> : True if <FILE> exists and is a directory
  -c <FILE> : True if <FILE> exists and is a character special file
  -b <FILE> : True if <FILE> exists and is a block special file
  -p <FILE> : True if <FILE> exists and is a named pipe (FIFO)
  -S <FILE> : True if <FILE> is a socket file
  -L <FILE> : True if <FILE> exists and is a symbolic link
  -h <FILE> : True if <FILE> exists and is a symbolic link
  -g <FILE> : True if <FILE> exists and has sgid bit set
  -u <FILE> : True if <FILE> exists and has suid bit set
  -r <FILE> : True if <FILE> exists and is readable
  -w <FILE> : True if <FILE> exists and is writable
  -x <FILE> : True if <FILE> exists and is executable
  -s <FILE> : True if <FILE> exists and has size bigger than 0
  -t <fd> : True if file descriptor <fd> is open and refers to a terminal
- <FILE1> -nt <FILE2> : True if <FILE1> is newer than <FILE2>
  <FILE1> -ot <FILE2> : True if <FILE1> is older than <FILE2>
  <FILE1> -ef <FILE2> : True if <FILE1> and <FILE2> refer to the same device and inode numbers

Conditional Statements

- String Tests
  -z <STRING> : True if <STRING> is empty
  -n <STRING> : True if <STRING> is not empty, and is the default operation
  <STRING1> = <STRING2> : True if the strings are equal
  <STRING1> != <STRING2> : True if the strings are not equal
  <STRING1> < <STRING2> : True if <STRING1> sorts before <STRING2> lexicographically
  Remember to escape (\<)
  <STRING1> > <STRING2> : True if <STRING1> sorts after <STRING2> lexicographically
  Remember to escape (\>)

Conditional Statements

- while [condition] do
  #command(s)
  #increment
  done
- for arg in [list]
  do
  #command(s)
  done
- for ((expression1; expression2; expression3)) do
  # Command 1
  # Command 2