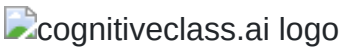

Hands-on Lab: Bash Scripting Advanced

Estimated time needed: **30** minutes

Objectives

After completing this lab you will be able to make use of the following features of the bash shell

- Metacharacters
- Quoting
- Variables
- Command substitution
- I/O Redirection
- Pipes and Filters
- Command line arguments

About Skills Network Cloud IDE

Skills Network Cloud IDE (based on Theia and Docker) provides an environment for hands on labs for course and project related labs. Theia is an open source IDE (Integrated Development Environment), that can be run on desktop or on the cloud. to complete this lab, we will be using the Cloud IDE based on Theia running in a Docker container.

Important Notice about this lab environment

Please be aware that sessions for this lab environment are not persisted. Every time you connect to this lab, a new environment is created for you. Any data you may have saved in the earlier session would get lost. Plan to complete these labs in a single session, to avoid losing your data.

Exercise 1 - Metacharacters

Run the commands given below on a newly opened terminal.

▶ Click here for Hint

There are several characters which have special meanings to the shell.

Following are some of the special characters and their usage.

1.1 '#' - For adding comments

Lines beginning with a # (with the exception of #!) are comments and will not be executed.

```
# This is a comment line
```

1.2 ';' - Command seperator

Multiple commands can be seperated from each other using a ; when used in a single command line.

```
pwd;date
```

1.3 * - wildcard used in filename expansion

The '*' character matches any number of any character in filename patterns. By itself, it matches every filename in a given directory.

The following example lists all files whose name ends with a '.conf' in the /etc directory.

```
ls /etc/*.conf
```

1.4 '?' - wildcard used in filename expansion

The '?' character represents a single character in a filename pattern.

The following command lists all files whose name starts with any single character followed by 'grep'.

```
ls /bin/?grep
```

Exercise 2 - Quoting

If any special character has to be treated without their special meaning, we need to quote them.

The following examples show how quoting is done in shell.

2.1 Quoting using backslash (\)

Backslash removes the meaning of the special character that follows it.

```
echo The symbol for multiplicaton is \*
```

2.2 Quoting using single quote (')

A pair of single quotes removes special meanings of all special characters within them (except another single quote).

```
echo 'Following are some special characters in shell - < > ; " ( ) \ [ ]  '
```

2.3 Quoting using double quote (")

A pair of double quotes removes special meanings of all special characters within them *except another double quote, variable substitution and command substitution.*.

Try out the examples below with double quotes as well as single quotes to see the difference between their usage.

```
echo "Current user name: $USERNAME"
```

```
echo 'Current user name: $USERNAME'
```

Exercise 3 - Working with variables

About Variables

Variables help store data for the script. The data may be in the form of a number or a character string.

You may create, remove or display the variables.

Let us now see how they are used in the shell.

3.1 List the variables already defined in the shell.

```
set
```

You should see a lot of variables in the output.

3.2 Create new variables

Use the syntax **variable_name=value**.

Create a new variable called 'balance' with a value of 10000. List all the variables again.

```
balance=10000
```

Run the set command to check if the variable `balance` has been created.

```
set
```

3.3 Create an environment variable

Environment variables are just like any other variable. They differ in the fact that they are copied to any child process created from the shell.

export command can be used to convert a regular variable to environment variable.

Make the variable ' balance' an environment variable.

```
export balance
```

3.4 List environment variables

Use the following command to list all the environment variables.

```
env
```

You should see a lot of variables in the output.

3.5 Display the value of a variable

To display or interpolate the value of a variable in a command, we use the feature of shell called **variable substitution**.

It is done by preceding the name of the variable with a **$** (dollar) symbol.

The command below prints the value of the variable $balance.

```
echo "Current account balance is $balance"
```

3.6 Remove a variable

To remove variables, use **unset** command.

Remove variable 'balance'.

```
unset balance
```

Run the set command to check if the variable `balance` has been removed.

```
set
```

Exercise 4 - Command substitution

Command substitution is a feature of the shell, which helps save the output generated by a command in a variable.

It can also be used to nest multiple commands , so that the innermost command's output can be used by outer commands.
The inner command is enclosed in **$()** and will execute first.

Let us try the following examples.

**4.1 Store the output of the command `hostname -i` in a variable named $myip

```
myip=$(hostname -i)
echo $myip
```

4.2 Print the following message on screen:

"Running on host : *host_name*" ,

Where 'host_name' should be your current hostname.

```
echo "Running on host: $(hostname)"
```

Command substitution can be done using the backquote syntax also.

```
ls -l `which cat`
```

The output of command `which cat` is the path to the command `cat`. This path is sent to ls -l as an input. You should see the permissions for the file `cat` in the output.

Exercise 5 - I/O Redirection

Linux sends the output of a command to **standard output (display)** and any error generated is sent to **standard error (display)**.

Similarly, the input required by a command is received from **standard input (keyboard)**.

If we need to change these defaults, shell provides a feature called **I/O Redirection**.

This is achieved using the following special characters.

Symbol	Meaning
<	Input Redirection
>	Output Redirecton
>>	Append Output
2>	Error Redirection

Let us try a few examples.

5.1 Save the network configuration details into a file called `output.txt`

In this example, we will send the output of `ifconfig` command to the file instead of standard output(display).

```
ifconfig > output.txt
```

Check out the contents of output.txt

```
cat output.txt
```

5.2 Save the output of `date` command into the file 'output.txt'.

```
date > output.txt
```

Check out the contents of output.txt

```
cat output.txt
```

Did you notice, that previous contents of output.txt were overwritten?

When you redirect using `>` the contents of the target file are overwritten.

5.3 Append output to a file

Now, we will try the following sequence, where we use '>>' instead of '>'.

Run the commands below.

```
uname -a >> newoutput.txt
date >> newoutput.txt
```

Check out the contents of newoutput.txt

```
cat newoutput.txt
```

You should see the output of `uname` and `date` commands appended to the file newoutput.txt

5.4 Dipslay the contents of file 'newoutput.txt' in all uppercase.

You can use the command **tr** for this translation.

tr command does not accept file names as arguments. But it accepts standard input.

So, we will redirect the content of file 'newoutput.txt' to the input of tr command.

```
tr "[a-z]" "[A-Z]" < newoutput.txt
```

You should see all capital letters in the output.

Exercise 6 - Pipes and Filters

Command pipeline is a feature of the shell, that helps us to combine different unrelated commands in such a way that one command's output is sent directly as input to the next command. This way, what is not possible with a single command can be made possible by connecting multiple commands.

Only filter commands can be used in this manner.

A **filter command** is a command which can accept input from standard input and send output to standard output.

Let us see some examples using few filter commands which we have already discussed.

6.1 Count the total number of files in your current directory.

Since the **ls** command doesn't provide an option to get a count, let us get help from **wc** command.

By combining them using command pipeline syntax, we get the following command.

```
ls | wc -l
```

6.2 Find the total disk space usage.

df -h command gives disk usage for all individual filesystems including the total usage across the server under the head `overlay`.

You can get the overall disk usage if you `grep` for overlay from the output of `df -h`

```
df -h|grep overlay
```

6.3 List five largest files.

The -S option of **ls** command sorts the files from largest to smallest.

We will send this sorted list through a pipe to the **head** command.

```
ls -lS /bin | head -6
```

You should see the list of five largest files from the /bin directory.

Exercise 7 - Command line arguments

Command line arguments are a very convenient way to pass inputs to a script.

Command line arguments can be accessed inside the script as $1, $2 and so on. $1 is the first arugment, $2 is the second argument.

7.1 Create a simple bash script that handles two arguments.

Save the below code as wish.sh

```
#! /bin/bash

echo "Hi $1 $2"

#$1 is the first argument passed to the script

echo "$1 is your firstname"

#$2 is the second argument passed to the script
echo "$2 is your lastname"
```

Make the script executable to everyone.

```
chmod +x wish.sh
```

Run the script with the two arguments as shown below.

```
./wish.sh Ramesh Sannareddy
```

You should see the below output.

Hi Ramesh Sannareddy

Ramesh is your firstname

Sannareddy is your lastname

7.2 Find the total disk space usage.

Let us create a bash script named `dirinfo.sh` that takes the directory name as an argument and prints the total number of the the directories and the number of files in it.

We will make use of the find command with -type option which will list only files or directories depending upon the usage of `d` switch or `f` switch respectively.

The command `wc -l` will count the lines.

Save the below code as dirinfo.sh

```
#! /bin/bash

dircount=$(find $1 -type d|wc -l)

filecount=$(find $1 -type f|wc -l)

echo "There are $dircount directories in the directory $1"

echo "There are $filecount files in the directory $1"
```

Make the script executable to everyone.

```
chmod +x dirinfo.sh
```

Run the script with the one argument as shown below.

```
./dirinfo.sh /tmp
```

In the output you should see number of files and directories in the directory /tmp.

Practice exercises

1. Problem.

> *Create a variable called 'color' and store the string 'light green' in it.*

▶ Click here for Hint

▼ Click here for Solution

```
color='light green'
```

The value should be in quotes since it has a space character in it.

2. Problem.

> *Display the list of all the files whose name starts with 'b' and ends with '.log' in the directory /var/log.*

▶ Click here for Hint

▼ Click here for Solution

```
ls /var/log/b*.log
```

3. Problem.

> *Display the count of all files whose name starts with 'c' in the /bin directory.*

▶ Click here for Hint

▼ Click here for Solution

```
ls /bin/c* | wc -l
```

4. Problem.

> *Display the value of variable 'color'.*

▶ Click here for Hint

▼ Click here for Solution

```
echo $color
```

5. Problem.

> *Store the value of the variable 'color' in a file 'color.txt'*

▶ Click here for Hint

▼ Click here for Solution

```
echo $color > color.txt
```

6. Problem.

> *Write a shell script named `latest_warnings.sh` that prints the latest 5 warnings from the /var/log/bootstrap.log file.*

▶ Click here for Hint

▼ Click here for Solution

```
#! /bin/bash
grep warning /var/log/bootstrap.log|tail -5
```

Authors

Ramesh Sannareddy

Other Contributors

Rav Ahuja

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2021-05-30	0.1	Ramesh Sannareddy	Created initial version of the lab