

SIRE511 : LINUX AND BIOINFORMATICS DATA SKILLS

Fundamental Linux Part IV :

4th Week: Network & Shell scripting part I,
24/09/2024

Kwanrutai Mairiang, Ph.D

Network in Linux

Exploring IP Address Settings

- Hostname

```
$ hostname
```

- The command for getting hostname of the Linux computer

- Private IP address

```
$ hostname -i
```

- The 'hostname' command with option '-I' will return the IP address that was assigned to this Linux computer. This IP is private IP. If this computer connect to internet, the private IP address will be translated to public IP address.

- Ping command

```
$ ping IP_address/URL
```

- Ping command is used for verification of connectivity with remote servers.

- Nslookup

```
$ nslookup URL/IP_address
```

- Command for querying the Domain Name System (DNS) to obtain domain name or IP address

Exploring IP Address Settings (Cont.)

The commands used to display information about all network interfaces on the system are `ip address` and `ifconfig`

```
$ ip address
```

```
/ # ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state LINKDOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00 IP for loopback interface
    inet 127.0.0.1/8 scope host lo address is reserved on all systems
        valid_lft forever preferred_lft forever
2: tunl0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN qlen 1000
    link/ipip 0.0.0.0 brd 0.0.0.0
3: ip6tnl0@NONE: <NOARP> mtu 1452 qdisc noop state DOWN qlen 1000
    link/tunnel6 00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00 Mac address of this interface is hardware address
5: eth0@if6: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 state UP virtual and it was added by Docker.
    link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.2/16 brd 172.17.255.255 scope global eth0 Private IP address
        valid_lft forever preferred_lft forever
```

IP for loopback interface or **localhost**. This IP address is reserved only for loopback purposes.

Mac address of this computer. MAC address is hardware address, but in this case it is virtual and it was assigned automatically by Docker.

Private IP address of this Linux computer.

Exploring IP Address Settings (Cont.)

```
$ ifconfig
```

```
/ # ifconfig
```

```
eth0      Link encap:Ethernet  HWaddr 02:42:AC:11:00:02  
          inet addr:172.17.0.2  Bcast:172.17.255.255  Mask:255.255.0.0  
          UP BROADCAST RUNNING MULTICAST  MTU:65535  Metric:1  
          RX packets:9 errors:0 dropped:0 overruns:0 frame:0  
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0  
          collisions:0 txqueuelen:0  
          RX bytes:1046 (1.0 KiB)  TX bytes:0 (0.0 B)
```

Mac address of this computer

Private IP address of this
Linux computer

```
lo        Link encap:Local Loopback  
          inet addr:127.0.0.1  Mask:255.0.0.0  
          inet6 addr: ::1/128 Scope:Host  
          UP LOOPBACK RUNNING  MTU:65536  Metric:1  
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0  
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0  
          collisions:0 txqueuelen:1000  
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

IP for loopback interface
or localhost

Enable **Secure Shell (SSH)** on the Linux sever

- What is SSH protocol?
 - SSH known as secure shell is a network protocol that gives users a secure way to access a computer over an unsecured network.
 - The SSH protocol encrypts all communication between two hosts that communicate via public open network like Internet.
 - SSH operates over TCP using port 22, so it's essential to have 22 open when using SSH.

Enable Secure Shell (SSH) on the Linux sever

1). Let's create a Docker container with the '-p' option to publish the internal port 22.

```
$ docker run -it -p external_port:internal_port IMAGE
```

```
$ docker run -it -p 2222:22 ubuntu
```

2). Install SSH

```
$ sudo apt update
```

```
$ sudo apt install openssh-server
```

3). Let's check that ssh is running.

```
$ service ssh status
```

4). If not, let's start SSH service

```
$ service ssh start
```

5). Add new user to server

```
$ adduser username
```

Connecting to the Linux server remotely using secure shell

- For macOS, open a new tab or a new terminal. For Windows, open PowerShell.
- The command to connect to a Linux server using SSH is :
 - `$ ssh username@hostname`
- We will connect to the Docker container on this computer. In this case, the hostname would be 'localhost', and you also need to specify the custom port 2222.
 - `$ssh -p 2222 username@localhost`

```
(base) kwan@MacBook-Pro-khxng-kwanrutai ~ % ssh -p 2222 kwan@localhost
The authenticity of host '[localhost]:2222 ([::1]:2222)' can't be established.
ED25519 key fingerprint is SHA256:JHy+0EDRIzigHb7oj5RJmGdfou/y/SdHqJBHkjWjn5w.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])?
```


Add user to Sudo group

By default, ssh does not allow remote login as the root user. The configuration file “/etc/ssh/sshd_config” contains the line:

```
#PermitRootLogin prohibit-password
```

The better approach is to grant the user sudo permissions.

1. Adding the user to the sudo group

- Only the root or a Sudoer user can perform this command!!

```
# usermod -aG sudo [name-of-user]           #By root  
$ sudo usermod -aG sudo [name-of-user]      #By sudoer user
```

2. Verify user belongs to sudo group

```
$ groups [name-of-user]
```

3. Verify sudo access

- Switch users use `su` command
- Test installing the nano program

```
# su - [name-of-user]  
$ sudo apt install nano
```

Bash Scripting Part I

What is Bash scripting?

- A Bash script is a plain text file that **contains many lines of Linux commands** (e.g. **echo, ls, cp**) to be performed in a batch, as opposed to entering each command line individually in the Linux terminal.
- Bash scripting could be used to automate multiple or repetitive tasks on Linux.
- Bash scripts are written in the **Bash programming language**, which has its **own syntaxes and structures**, including loops, conditional constructions (if...else), and data containers, comparable to those of other programming languages.

Creating executable script

A Bash script file must be created and checked for the execution permission status before running

[Create bash script](#)

For convenience, the name of script can follow this format.

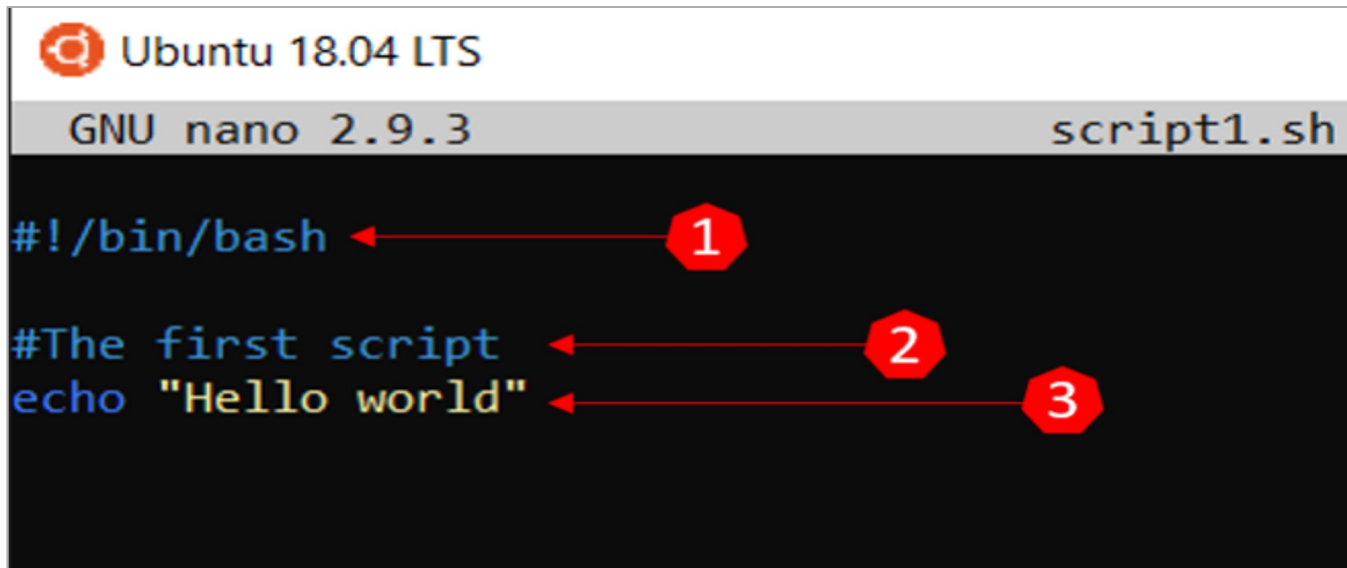
- Avoid adding spaces in the name, use underscore instead.
- Use alphanumerical [a-zA-Z0-9]
- File name has the extension “.sh”

```
## Create and open file “script1.sh” for editing
```

```
nano script1.sh ↵
```

Creating executable script

Write the following script in 'script1.sh' using nano.



```
#!/bin/bash
#The first script
echo "Hello world"
```

1. Shebang (`#!/`) at the first line of script is used to instruct the OS to use bash as a command interpreter and specified the path of the interpreter.
2. The line starts with `#` will not be executed by interpreter. This line is referred to as a "comment" and is useful for describing the script.
3. Line of code. This code will print **Hello world** on the screen.

Set the execution permission

Make a script executable

```
chmod +x script1.sh
```

```
kwan — root@3a24cc42a0e7: /data — com.docker.cli • docker start -i -a 3 — 60x14
root@3a24cc42a0e7:/data# ls -l
total 4
-rw-r--r-- 1 root root 50 Sep 17 05:11 script1.sh
drwxr-xr-x 4 root root 128 Sep 17 05:25 week1
drwxr-xr-x 5 root root 160 Sep 17 05:17 week2
root@3a24cc42a0e7:/data# chmod +x script1.sh
root@3a24cc42a0e7:/data# ls -l
total 4
-rwxr-xr-x 1 root root 50 Sep 17 05:11 script1.sh
drwxr-xr-x 4 root root 128 Sep 17 05:25 week1
drwxr-xr-x 5 root root 160 Sep 17 05:17 week2
root@3a24cc42a0e7:/data#
```

- The current status of the execute permission of “script1.sh” is “denied”.
- To change the execute permission, a command “chmod”, which is short for “change mode,” will be used.

To run the script, just type “/path/to/file/script1.sh”: `./script1.sh`

Variable in Linux

Variables

- Variables are important parts of programming. Variables store data to be use later in the script. There are two types of bash variables in a shell or Linux system.
 - **System**-Defined Variables (Environment variable)
 - **User**-Defined Variables

System-Defined Variables (Environment variable)

These are the variables that are automatically assigned by LINUX operating system (i.e. built-in variables). They generally named in **CAPITAL LETTER**. An example list of System-Defined Variables is shown below.

Variables	Meaning	Example value
BASH	Return the bash path	/bin/bash
BASH_VERSION	Return the shell version	5.1.16(1)-release
HOME	Specifies the home directory	/root
PWD	Specifies the current working directory	/data
PATH	List of directories in which the shell looks for commands.	/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

use: **“env”** to check in the system

User-Defined Variables (Local variable)

The variables created by user. This type of variables can be defined in either upper or lower case, but generally in lower cases. The rules for naming user-defined bash variables are as follows.

- A variable name can include alphabets, digit, and underscore (_)
 - Valid names:
 - level, level1, _level, level_1
 - Names **cannot start with digit**:
 - 1level, 1_level
- The variable name might be in all CAPS, all lowercase, or a mixture of both
- The variable name is case-sensitive.

Setting variables in bash script.

- The equal sign (=) is used for assigning a value to a variable.
 - The variable is located on the left of equal sign while value is on the right.
 - The whitespace **should not** be added on either side of equal sign.

```
name=kwanrutai
```

- When referring to a previously defined variable, the dollar sign (\$) is prefixed to the variable's name.

```
echo $name
```

Practical: Setting variables in bash script.

Edit '[script2.sh](#)' using nano, and then execute the script.

```
#!/bin/bash

name= #Add your name (var1)

gender= #Add your gender (var2)

age= #Add your age (var3)

echo "My name is #var1. My gender is #var2. I'm
#var3 years old."
```

String Manipulation

String Length

There are many ways to calculate the string length.

1) A simple way to calculate the length of the string is to use **# symbol**.

Syntax:

```
${#string_variable_name}
```

2) Calculate the length of the string using an **“expr”** command with an option **“length”**.

Syntax:

```
expr length "$string_variable_name"
```

3) Use an **“awk”** command to calculate the length of the string

Syntax:

```
echo $string_variable_name | awk '{print length}'
```

Practical: String Length

```
#!/bin/bash
```

```
str= # Add any sentence
```

##Syntax 1

```
length1=${#str}
```

```
echo "Syntax 1: Length of '$str' is $length1"
```

##Syntax 2

```
length2=$(expr length "$str")
```

```
echo "Syntax 2: Length of '$str' is $length2"
```

##Syntax 3

```
length3=$(echo $str | awk '{print length}')
```

```
echo "Syntax 3: Length of '$str' is $length3"
```

Edit '[stringLen.sh](#)' using nano, and then execute the script.

Substring

- Bash scripting provide an option to **extract a substring from a string.**

Syntax:

```
$ { string:position:length }
```

Extract **Length** characters of substring from **String** at **Position**.
The **Position** starts from 0.

Practical: Substring

Execute the script “substring.sh”

Extract first 10 characters of string (position = 0, length = 10)

```
#!/bin/bash

str="My name is Kwanrutai"

substr="${str:0:10}"

echo "Full string: $str"

echo "Substring: $substr"
```

Practical: Substring

Use nano to edit the script 'substring.sh'.

- Extract substring from specific character onward
 - Ex1. Extract substring 11th character onwards (position = 11, length = end of string)
`substr="${str:11}"`
- Extract substring at the middle of string
 - Ex2. Delete the first 3 characters and then print 12 subsequent characters (position = 3, length = 12)
`substr="${str:3:12}"`
- Extract a specific number of characters counting from the end of the string
 - Ex3. Extract last 9 character (position = -9, length = end of string)
`substr="${str: (-9) }"`

Shortest (non-greedy) **substring match**

The syntax for **deleting the shortest match** of the substring from the string

Syntax: Delete matched **substring** **from the beginning** of **string**

```
$ { string#substring }
```

Syntax: Delete matched **substring** **from the end** of **string**

```
$ { string%substring }
```

Practical: Shortest substring match

Delete matched substring from full string

[substringMatch1.sh](#)

```
#!/bin/bash

filename="p1.1.fastq.gz"

begin=${filename#*.} #Delete from the beginning

end=${filename%.*} #Delete from the end

echo "Shortest match from the beginning: $begin"

echo "Shortest match from the end: $end"
```

Longest (greedy) substring match

The syntax for deleting the longest match of substring from string

Syntax: Delete match **substring** from the beginning of **string**

\$ { **string##substring** }

Syntax: Delete match **substring** from the end of **string**

\$ { **string%%substring** }

Practical: Longest (greedy) substring match

Delete matched substring from full string

[substringMatch2.sh](#)

```
#!/bin/bash

filename="p1.1.fastq.gz"

begin=${filename##*.} #Delete from the beginning

end=${filename%%.*} #Delete from the end

echo "Longest match from the beginning: $begin"

echo "Longest match from the end: $end"
```

Find and replace

1). Replace only the first match

Find the **pattern** in **string** and replace only the first match by **replacement**.

Syntax:

```
$ { string/pattern/replacement }
```

Practical 1: Replace only the first match

stringReplacement1.sh

```
#!/bin/bash

filename="p1_1.fastq.gz"

replacement=${filename/_*.gz/.paired.fastq}

echo "After replacement: $replacement"
```


Find and replace

2). Replace all the matches

Find the **pattern** in **string** and replace all matches by **replacement**.

Syntax:

```
$ { string / pattern / replacement }
```

Practical 2: Replace all matches

stringReplacement2.sh

```
#!/bin/bash

filename="Path of the bash is /bin/bash"

replacement=${filename//bash/sh}

echo "After replacement: $replacement"
```

Find and replace

3). Replace at the beginning or the end

Find the **pattern** in **string** and replace only first match by **replacement**.

Syntax: Replace matched **pattern** with the **replacement** from the beginning of the **string**

```
$ { string / # pattern / replacement }
```

Syntax: Replace matched **pattern** with the **replacement** from the end of the **string**

```
$ { string / % pattern / replacement }
```

Practical 3: Replace at the beginning or the end of string

stringReplacement3.sh

```
#!/bin/bash

filename="p1_1.fastq.gz"

begin=${filename/#*_/p2_} #Replace from the beginning

end=${filename/%.*/.paired.bam} #Replace from the end

echo "Replace at the beginning: $begin"

echo "Replace at the end: $end"
```

Tips: Remove special characters from a text file

Remove everything except the printable characters (character class `[:print:]`), using `sed` command

```
$ sed -i 's/^[^[:print:]]\t//g' file.txt
```

`[:print:]` is printable characters includes:

`[:alnum:]` (alpha-numerics)

`[:punct:]` (punctuations)