H3ABioNet
Pan African Bioinformatics Network for H3Africa

Introduction to Bioinformatics online course: IBT

Linux

Linux: Permissions, groups, and process control

Learning Objectives

① Understand file and directory permissions and how to change them

② Understand loops, variables and script generation to automate tasks

③ Understand environment variables and why they are important

④ Learn how to ssh onto a remote machine

Learning Outcomes

① Be able to change file permissions

② Be able to write a simple script

③ Know some of the environment variables

④ Be able to connect via ssh onto a remote machine

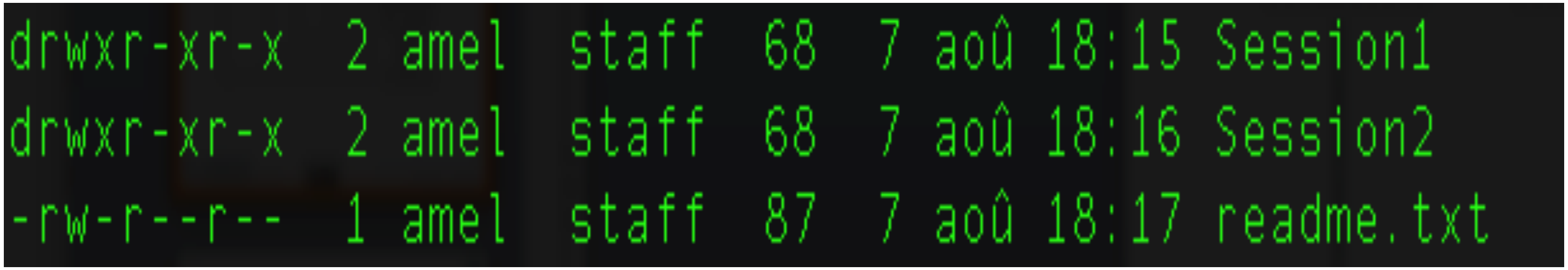
Part 1

Files and directories permissions

Linux is a multi-users OS

- On a Linux system, each file and directory is assigned access rights for the owner of the file, the members of a group of related users, and everybody else

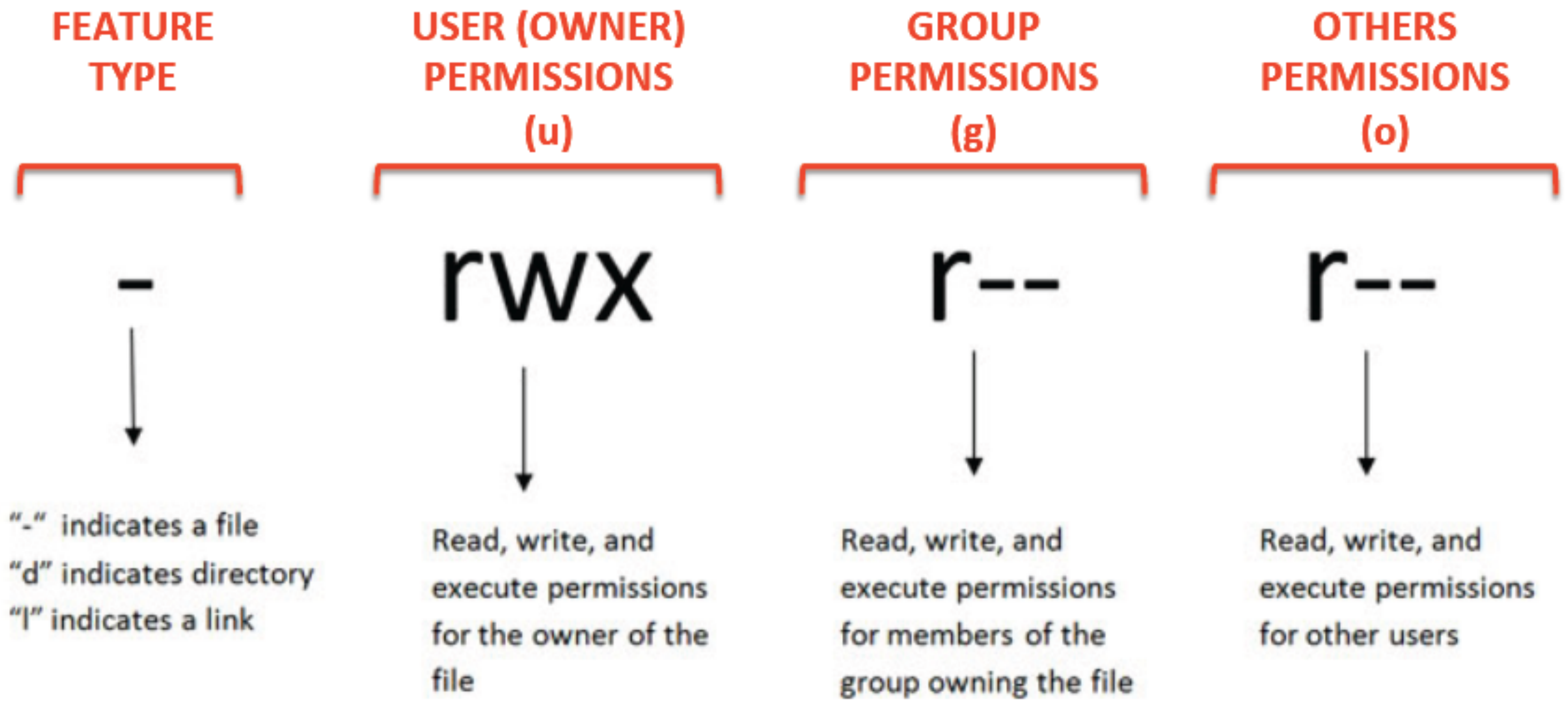
Remember the ls -l example



```
drwxr-xr-x  2 amel  staff  68  7 aoû 18:15 Session1
drwxr-xr-x  2 amel  staff  68  7 aoû 18:16 Session2
-rw-r--r--  1 amel  staff  87  7 aoû 18:17 readme.txt
```

```
----------     -------  -------  --------  -----------  ------------
    |              |       |        |          |              |
    |              |       |        |          |          File Name
    |              |       |        |          |
    |              |       |        |          +---   Modification Time
    |              |       |        |
    |              |       |        +-------------     Size (in bytes)
    |              |       |
    |              |       +-----------------------         Group
    |              |
    |              +-------------------------------         Owner
    |
    +----------------------------------------------    File Permissions
```

Permissions are broken into 4 sections



FEATURE TYPE

-

"-" indicates a file
"d" indicates directory
"l" indicates a link

USER (OWNER) PERMISSIONS (u)

rwx

Read, write, and execute permissions for the owner of the file

GROUP PERMISSIONS (g)

r--

Read, write, and execute permissions for members of the group owning the file

OTHERS PERMISSIONS (o)

r--

Read, write, and execute permissions for other users

Source: www.pluralsight.com

Access permissions on files

- r indicates read permission: the permission to read and and copy the file
- w indicates write permission: the permission to change a file
- x indicates execution permission: the permission to execute a file, where appropriate

Access permissions on directories

- r indicates the permissions to list files in the directory
- w indicates that users may delete files from the directory or move files into it
- x indicates means the right to access files in the directory. This implies that you may read files in the directory provided you have read permission on the individual files

chmod command

- Used to change the permissions of a file or a directory.
- Syntax: chmod options permissions filename
- Only the owner of the file can use chmod to change the permissions
- Permissions define permissions for the owner, the group of users and anyone else (others)
- There are two ways to specify the permissions:
  - ✓Symbols: alphanumeric characters
  - ✓Octals: digits (0 to 7)

chmod options

Symbol	Meaning
u	user
g	group
o	other
a	all
r	read
w	write (and delete)
x	execute (and access directory)
+	add permission
-	take away permission

Octal permissions

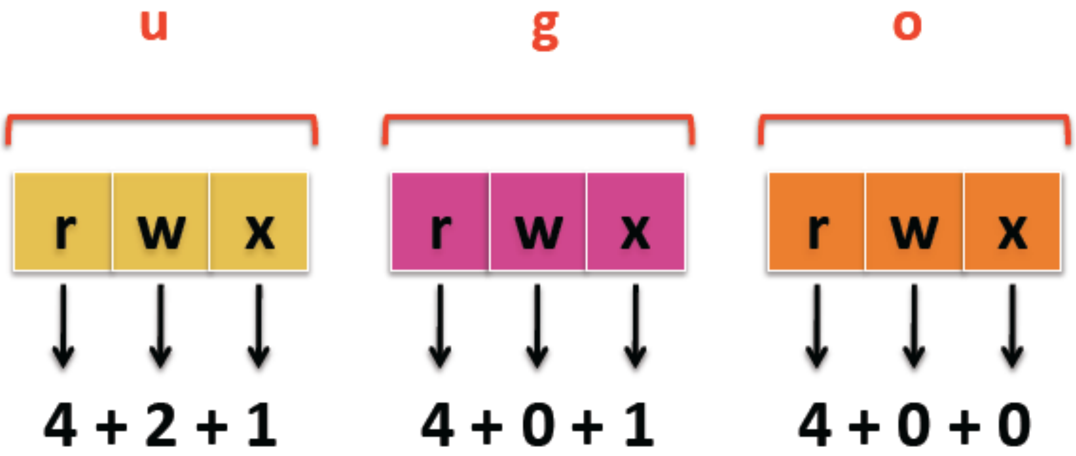
- **4** stands for "read”
- **2** stands for "write”
- **1** stands for "execute”
- **0** stands for "no permission"

chmod examples

- chmod u=rwx,g=rx,o=r filename

same as

- chmod 754 filename



More examples

- 777: (rwxrwxrwx) No restrictions on permissions. Anybody may do anything
- 755: (rwxrxrx) The file's owner may read, write, and execute the file. All others may read and execute the file (common for programs that are used by all users)
- 700: (rwx) The file's owner has all the rights. Nobody else has any rights (private for the owner)
- 666: (rwrwrw) All users may read and write the file.
- 644: (rwrr) The owner may read and write a file, while all others may only read the file (everybody may read, but only the owner may change)
- 600: (rw) The owner may read and write a file. All others have no rights

Part 2

Environment variable

Variables

- Variables are areas of memory that can be used to store information and are referred to by a name
- How to create a variable: a line that contains the name of the variable followed immediately by an equal sign ("=").
- 2 types of variables: shell variables and environment variables
- Some variables are already set in your shell session
- printenv: prints the values of all your environment variables

What is an environment variable

- An environment variable is a dynamic "object" on a computer that stores a value, which in turn can be referenced by one or more programs.
- Environment variables help programs know what directory to install files in, where to store temporary files, where to find user profile settings, and other things.
- Environment variables help to create and shape the environment of where a program runs.

Examples of environment variables

- HOME: the environmental value that shows the current user's home directory,
- PATH: the environmental variable, which contains a colon-separated list of the directories that the system searches to find the executable program corresponding to a command issued by the user
- PWD: always stores the value of your current working directory

Part 3

Shell scripting

echo command

- Syntax: echo options arguments
- Writes arguments to the standard output
- echo: just prints its command-line parameters to standard output
- If you redirect the result your arguments will be written into the file you are redirecting to
- Commonly used by the shell scripts to display results or ask the user to enter parameters

Let’s echo some stuff

- ✓ echo Bioinformatics is great starting writing scripts
- ✓ If you want to jump to another line add \n and use the option –e
- ✓ echo –e “Bioinformatics is great \n starting writing scripts”
- ✓ Setting a variable: X=firstvariable
- ✓ echo X: prints X
- ✓ echo $X prints firstvariable (the value of the variable)
- ✓ echo ‘$X’ ➔ $X
- ✓ echo “$X” ➔ firstvariable

Print the result of a command

- Asking the shell to substitute the results of a given command
- `command` or $(command)

- echo `pwd` or echo $(pwd)

What is a shell script

- Short programs written in shell programming language useful to automate tasks under Linux OS
- A shell script is a file containing a series of commands
- Could be helpful to perform the same actions on many different files
- Shell script= scripting interpreter+ command line interface to the system
- echo is also commonly used to have a shell script display a message or instructions, such as *Please enter Y or N* in an interactive session with users.

Let’s start using the power of scripting

1. nano myfirstscript
2. Write the content of your script for example:

```
#!/bin/bash
clear
echo “Hey, I am starting writing shell scripts”
echo “Let the fun begin!!! ”
```

The shebang

3. Run your script (using ./)

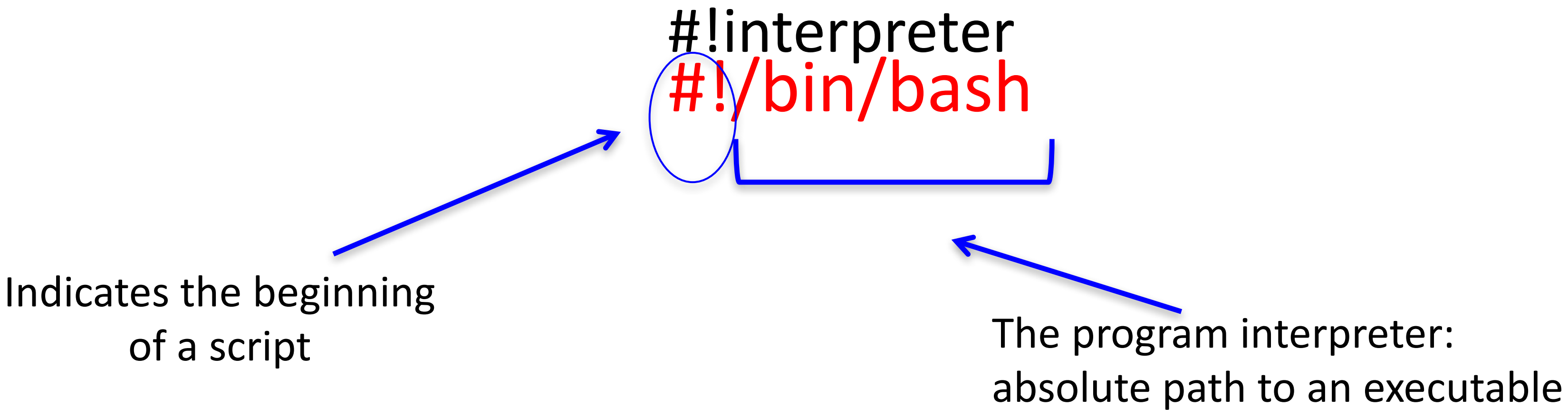
4. Change the rights to make sure you have the right to execute

```
chmod u+x myfirstscript
```

or

```
chmod 744 myfirstscript
```

The shebang



#!interpreter

#!/bin/bash

Indicates the beginning of a script

The program interpreter: absolute path to an executable program

- A perl script could begin by #!/usr/bin/perl (absolute path)
- You can use the which command to locate the executable file associated with a given command
- which perl → /usr/bin/perl
- which bash → /bin/bash

Use variables in your scripts

- Makes your script easier to maintain
- Reduces the amount of typing !

If statements in shell scripting

Syntax:

```
if [ conditional
expression ]
then
        commands
else
        commands
fi
```

Example:

```
#!/bin/bash
echo  Let's try some conditional tests
x=`find *.fasta| wc -l`
echo "The current working directory
contains $x fasta files"
if [ $x -gt $y ] # if (($x > $y))
then
echo there are many existing fasta files  in
this directory
else
echo There are very few fasta files in this
directory here is the listing:  `ls *.fasta`
fi
```

Loops in shell scripting (for)

Syntax:

```
for variable in values
do
        commands
done
```

Example:

```
#!/bin/bash
for x in file1 file2
do
        head -n 3 $x
        echo operation completed on file:$x
done
```

Loops in shell scripting (while)

Syntax:

```
while [ condition ]
do
        command1
        command2
done
```

Example:

```
#!/bin/bash
n=1
while [ $n <= 5 ]  #n should
have an initial value
do
        echo Welcome $n times
        n=$(( n+1 )) #increment
$n
done
```

Operators supported by shell (1)

Examples: consider 2 variables a= 10 and b=20

Operator	Description	Example
+	Addition - Adds values on either side of the operator	`expr $a + $b` will give 30
-	Subtraction - Subtracts right hand operand from left hand operand	`expr $a - $b` will give -10
*	Multiplication - Multiplies values on either side of the operator	`expr $a \* $b` will give 200
/	Division - Divides left hand operand by right hand operand	`expr $b / $a` will give 2
=	Assignment - Assign right operand in left operand	a=$b would assign value of b into a
==	Equality - Compares two numbers, if both are same then returns true.	[$a == $b] would return false.
!=	Not Equality - Compares two numbers, if both are different then returns true.	[$a != $b] would return true.

Source: http://www.tutorialspoint.com/unix/unix-basic-operators.htm

Operators supported by shell (2)

Examples: consider 2 variables a= 10 and b=20

Operator	Description	Example
-eq	Checks if the value of two operands are equal or not, if yes then condition becomes true.	[$a -eq $b] is not true.
-ne	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.	[$a -ne $b] is true.
-gt	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	[$a -gt $b] is not true.
-lt	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	[$a -lt $b] is true.
-ge	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	[$a -ge $b] is not true.
-le	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	[$a -le $b] is true.
-eq	Checks if the value of two operands are equal or not, if yes then condition becomes true.	[$a -eq $b] is not true.

Source: http://www.tutorialspoint.com/unix/unix-basic-operators.htm

Part 4

Controlling tasks

Commands to control processes

- ps: list the processes running on the system
- kill: send a signal to one or more processes (usually to "kill" a process)
- jobs: an alternate way of listing your own processes
- bg: put a process in the background

Launching a background job

- Programs that takes time or open a new Graphical User Interface

➔ The prompt doesn’t reappear after the program launched. The shell is waiting for the program to

finish before control returns to you

- ctlr+Z: interrupts a program

Or

- You can put it in the background so that the prompt will return immediately

➔ Use the command name followed by & to do so

Part 5

SSH into remote machine

What is SSH

- SSH (secure Shell) is a protocol used to securely log onto remote systems (remote Linux machine and Unix-like servers)
- ssh command is the tool used in Linux to connect via SSH protocol
- Syntax: ssh remoteusername@remotehost
- Remote host could be an IP address or domain name
- You will be asked to provide your password
- To exit and go back to your into your local session, use exit

Multi-users in a Linux machine

- While your computer only has one keyboard and monitor, it can still be used by more than on user.
- For example, if your computer is attached to a network, or the Internet, remote users can log in via ssh (secure shell) and operate the computer.

- Remote users can execute applications and have the output displayed on a remote computer

Copy files from or to a remote machine

- scp: secure copy
- Syntax: scp pathfrom pathto
- The difference: in scp, at least the source or the destination is in a remote machine
- Example: uploading all the .txt files from your current working directory to a remotehost

```
scp ./*.txt username@myhost.com:/home/username/folder
```

Thanks

Shaun Aron & Sumir Panji