Linux Tutorial

How to read the examples
When talking about how to launch commands and other things that is to be typed into the terminal, the following syntax is used:

$ application <parameters> file.txt

- The initial $ means that this is something you are going to write in your terminal.
- Words in <> brackets should be entered without the <> signs
- The filename file.txt denotes a generic filename

Example:

$ rm <-r> directory_3 **this should be typed as** ***rm –r directory_3***
this gives the instruction to rm – remove directory
-r is an option that states remove the contents of this directory recursively and directory_3 states the directory to be acted on. On the

1. Log on to a remote system on windows

Launch mobaxterm and fill in the credentials <**hpc.ilri.cgiar.org**> for a new ssh-session. If all works, you will see a new terminal opening with a greeting message.

Type interactive on the shell and hit enter to move onto Taurus

2. Managing files directories and permissions

This part of the course is about making you feel familiar with the most basic functionality of a unix shell. The current working directory can be considered as your place in the directory tree, this is where you are. To change the working directory can also be referred to as go to or change the directory.

Creating your file structure

The first thing we want to do is to make sure that the current working directory is you home directory. Your home folder will be ***/home/<user name>.***

Practical exercise

a) Go to your home directory by running: ***$ cd***
b) Verify that your working directory is your home directory by running: ***$ pwd***
c) Now create a directory named **results:** *$ **mkdir results***
d) Change to your results directory:*$ **cd results*** (This is where you will be storing the results for all the exercises).
e) Create a text file called **homedir.txt** in the results directory using the text editor vim: ***$ vim homedir.txt***
f) Write in this file by typing **i** to change to insert mode then type, **"This is my first Linux file"**. Save and exit vim by pressing **esc then :wq**
g) Make sure that the file is created by listing all files in the directory with: ***$ ls***
h) Try and rename your file by using the command **mv:**
$ mv homedir.txt exercise.txt
i) Now use the ls command to see the change in name: ***$ ls*** (you should notice the file homedir.txt has been replaced with exercise.txt)
j) Make a copy of this file by using the cp command:
$ cp exercise.txt exercise-two.txt (use ***$ls*** to observe changes)
k) Create a new directory called trial: ***$ mkdir trial*** now try to copy the file exercise-two.txt to this directory: ***$ cp exercise-two.txt trial/***
l) Use the command rm to delete the second file: $ ***rm exercise-two.txt***
m) Try and use the same command rm to delete the folder *trial* (hint if you get an error use the option ***$ rm –r***)

You have now created the results directory where all answers to questions in this set of exercises will be saved. Now you will have to remember how to go to the results directory and create new files with appropriate filenames. There are many ways to create the files, the simplest but not always the best is to use the text editor vim to write the results.

Now it's time to get the files for this tutorial. You will now have to download some files to your home directory from the workshop website. These are the files you will be working with in the rest of this tutorial

Practical exercise

a) Go back to your home directory:***$ cd***
b) Make a directory called tutorial: ***$ mkdir tutorial***
c) Change to the tutorial directory: ***$ cd tutorial***

d) Download files to your home directory from the workshop website using the wget command that retrieves content from web servers:
$wget http://hpc.ilri.cgiar.org/beca/training/sequence.fasta

e) Use the long list option of ls command to view the file you have just downloaded: ***$ ls -l***

f) Use the command cat, more, less to view the contents of the file you have downloaded (you can type q to quit viewing): ***$ cat sequence.fasta or $ more sequence.fasta or $ less sequence.fasta***

g) By using the following command, you will read the help detail for the command wget. Use this to describe the meaning of the '-V' parameter. You can scroll up and down by using the arrow keys: ***$ wget -h***

h) Write a file in the results folder called wget.txt where you describe in short (Hint use the vim editor to create this file):

   i. **The meaning of the '-V' parameter**
   ii. **The names of the files you've just downloaded using the wget command (tip: use the command ls to find out names of files in a directory)**

i) You can also access files in different folders then your current folder, but then you will have to specify the relative or absolute path to the file. The absolute path is the whole path to a file and the relative path is the path from where you're standing in the directory tree.

j) Go to your home folder: ***$ cd***

k) To read the file **wget.txt** from this location by using relative paths you will have to enter the path between your position and the file (to read the wget.txt file you have to point to the results folder then the wget.txt file). **$ less results/wget.txt**

3. Redirection of streams

In this excise, you will use the powerful concept of managing streams. You will learn how to send data between applications and store program outputs in files.

Everything you write and read on the screen or type on your keyboard can be considered as character streams, or just streams. Described below is the three types of streams you need to know about.

stdout—-Standard output—-This is the normal output for a program. This is what you can see being displayed on the screen.

stderr—-Standard error—-When an application displays error messages it normally uses this stream for presenting these. The messages are usually presented on the same way as stdout streams.

stdin—-Standard input—-This is the standard input stream for a program which it normally gets from the things you write on the keyboard.

To redirect a stream, you simply write the command to start your application follow by an operator and a file name as**:$ application <operator> <file name>**
For instance if you want to save a directory listing to file:*$ ls > myFileList.txt*

> - Create a new file (or replace the current one) with the stdout stream of the application.

» — Rather then to replace the current file, this operator adds the stdout to the end of the existing file.

2> —Writes the stderr to a new file (or to replace the current one)

< — Takes an existing file and writes it to the stdin stream of an application.

2>&1 —Redirects stderr to stdout

The simplest redirection of streams is when you want to store the output of a program to a file. Instead of creating a new file by hand you could use the stream redirection feature to let the program create this file for you.

Practical exercise

a) Start by going to your home folder (you should now know how to do this if not refer to previous practical exercises)

b) Test to run this command, what does it do? ***$ echo "The current time and date is:"***

c) Now redirect this output to a new file in your results folder called date.txt: ***$ echo "The current time and date is:" > results/date.txt*** (less results/date.txt to view your results)

d) Test to run this command date what does it give : ***$ date***
e) Use a suitable redirection operator for adding the current *date* at the end of the previously created file using the program date. (hint you can use the >> to append your results to a current file)

There is a good program for searching in streams called "grep". This is an application that you probably will use a lot! Now we will be using this program to search for a given string in a big file of sequences **tutorial/sequence.fasta**. This file contains dna sequences of various 16S genes. Let's search this file for the occurrences of the string ***"ribosomal"***.

Practical exercise

a) You have to be in your home folder in order to get the relative paths to the file to work: ***$ cd***
b) Start by taking a quick look in the file by using the application less (Quit by using q). ***$ less*** **tutorial/sequence.fasta**
c) To search for a certain string with grep, use the redirection of stdin from a file to grep (This should give all the matches it finds in the file to **ribosomal**): *$* **grep 'ribosomal' < tutorial/sequence.fasta**
d) You can use multiple stream redirection operations on the same line. With this in mind, take the command listed above and make it write it's output to the file **results/sequence_search.txt : *$*** **grep 'ribosomal' < tutorial/sequence.fasta** > **results/ sequence_search.txt**
e) Use the less command to try and view the **sequence_search.txt**

4. Piping

Piping is a special form a stream redirection where you take the output from a program and sends it to the input of the next program. This operation can be carried out in chains, which makes it very useful and powerful. The pipe operator is a | character that is placed

right between the two application. To put a pipe between applicationA and applicationB would result in the following syntax:

$ applicationA | applicationB
You can also make longer chains like:

$ applicationA | applicationB | applicationC | applicationD
Now we're going to use pipes to carry out the grep example above in an other way.

Practical exercise

a) To be able to do this, we must get the file **results/ sequence_search.txt** to be printed as an output. This is exactly what the program cat does. $ ***cat results/sequence_search.txt***
b) Now use the pipe operator between the cat command from above and the grep command to search for "***ribosomal***" (*grep "**ribosomal**"*). The output should look the same as in the example with stdin redirection.
$ cat tutorial/sequence.fasta |grep 'ribosomal'

c) To complicate things even further, add a third command to your piping-chain that's counting the number of lines outputted from your previous piping-chain. Using the program wc –l (this counts the number of lines in the out put). Now you have a chain that's counting the number of occurrences of "***ribosomal***" search hits. Hint :
$ cat **tutorial/sequence.fasta**| grep "***ribosomal***" | wc –l

d) Using the redirection > Try and redirect the results to the file **results/sequence_lines.txt.**

5. Combining pipes and stream redirects

Piping and stream redirection can be combined when the redirecting is on the fist program input or the last program output. Like:

$ applicationA < inputFile.txt | applicationB | applicationC > outputFile.txt
We are now going to use this function to get a small selection of lines from a big file. This is really useful when testing computational heavy algorithms.

We only want to get the 20 first lines containing the string **"16S"** in the file **tutorial/sequence.fasta** and store this in the file **results/sequence_selection.txt.**

Practical exercise

a) Use the approach in the previous example to only get all the lines containing the desired search string **"16S".** (hint use grep

b) The program head is used to only output the first lines from stdin. There is a parameter that lets you modify how many lines it should output try this: $ head -20 tutorial/sequence.fasta

c) Now use this application in you piping chain to only print the 20 first lines containing the search string. (Hint $ grep "pattern" tutorial/sequences.fasta | head -20)

d) Use a stream redirection to store this output the file **results/sequence_head_selection.txt.**

6. Permissions

Now that you have created a directory for your results, you probably want to keep these results to your self. You don't want the next group to be able to copy your results and claim your success! An easy way of preventing other users on the system to be able to look at your files is to change the permissions of the files.

So what is permissions? It's basically a system for telling which users that are allowed to carry out certain tasks. The users of a system is divided into three groups:

1. Owner (or user) is the single user that owns a certain file
2. Group is a set of users that belong to the same group
3. Others are all users that aren't members of the group nor the owner

Each of the groups above has three different types of permissions:

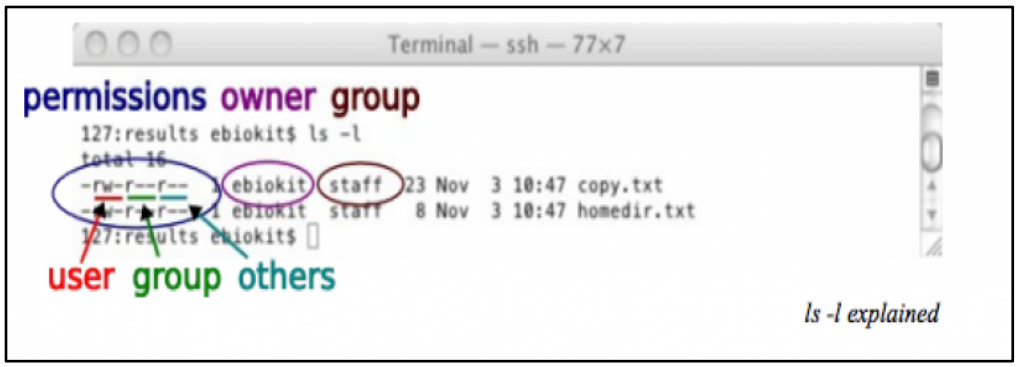
1. You can read ® the file. For folders, this means listing the contents of the folder

2. You can write (w) to (change) the file. For folders, this means creating and deleting files in the folder
3. You can execute (x) (run) the file, if it's a program or script. For folders, this means accessing files in the folder.

All three groups of users have each three different types of permissions, in total nine different permissions to modify. Take a look at the image below for an explanation. Now back to the problem. First we want to take a look at the current permissions on the files. The easiest way of doing this is to list all files in your results folder.

Practical exercise

a) Go to your results folder:*$ cd results*
b) Now list the files in the folder. Use the parameter '-l' to get the files in a detailed list *$ ls –l*
c) Now analyze the information from your files which should be similar to the image above.



d) Create a file in results called initial_permissions.txt where you answer the following questions:

- **Who are allowed to edit the files?**
- **Who are allowed to read the files?**
- **To which user and group does this file belong?**

e) The next step is change the permissions so that the files are only readable and writeable to the owner of the files (you). There are basically two ways of doing this. We will be applying these two methods for both files in the picture above. To change permissions on the files we will be using the program chmod.

Since we know that we want to remove the reading permission for group and others and still keep the permissions for the user, it's convenient to apply the changes we want to

make one at the time. The fist command means that we remove the reading permission for group, the second means that we remove the reading permission from others.

$ chmod g-r copy.txt $ chmod o-r copy.txt

Running this way you change a single permission one at a time. If you want to add a permission instead of removing one, you will use a + sign instead of a -. For instance if you want to add executable permissions to the owner you write:

$ chmod u+x copy.txt

2 This method is a bit more complicated and requires some knowledge about binary numbers. Observe that we still want the owner to have read and write permissions while every one else should have no permissions at all. User, group and others are represented with a decimal number that is derived from a desired permissions string. Follow the example below to create the permission mentioned above. (The desired string is r w _ _ _ _ _ _ _ _)

This example would lead to the command: $ chmod 650 myFile.txt To change the permissions on all individual files would potentially create lots of work since we have to do this same operation on every new file. A better solution for the problem would be to change the permissions for the entire folder. To accomplish this, you will have to give your self (user) read, write and execution permissions and no permissions at all for *group* and *others.*

Acknowledgment

EBioKit Tutorials for the LINUX tutorial material