13-UnixCommands

August 11, 2020

1 Basic Commands in the Unix Shell

For windows 10 users, activate bash.

Or install Jupyter Lab

```
conda install jupyterlab -c conda-forge
jupyter lab
```

1.1 Unix Shell

The shell is a command programming language that provides an interface to the UNIX operating system. Documentation of unix command is displayed by command `man`. Exemple:

```
man whoami
```

[1]:
```
#%%bash
#man whoami
```

1.2 Directories

The shell should start you in your home directory. This is your individual space on the UNIX system for your files. You can find out the name of your current working directory with the unix command `pwd`.

In the terminal, type the letters 'p', 'w', 'd', and then "enter" - always conclude each command by pressing the "enter" key. The response that follows on the next line will be the name of your home directory, where the name following the last slash should be your username.) The directory structure can be conceptualized as an inverted tree.

In the jupyter notebook, unix shell command can be executed using the escape character "!" or add `%%bash` to the cell first line. You can type command directly in a terminal without the "!".

[2]:
```
#%%bash
#pwd
```

Some unix command (not all) are also jupyter magic command like %pwd

[3]:
```
#%pwd
```

1.3 Home directory

No matter where in the directory structure you are, you can always get back to your home directory with `cd`.

1.3.1 Create a new subdirectory named "primer" :

```
mkdir primer
```

[4]:
```
#%%bash
#rm -rf primer  # remove primer directory if it exists
#mkdir  primer  # make the new directory
```

Now change to the "primer" subdirectory, making it your current working directory:

```
cd primer
pwd
```

[5]:
```
#%cd primer
```

[6]:
```
#pwd
```

1.4 Files

Create a file using `date` command and `whoami`:

```
date >> first.txt
whoami >> first.txt
```

date and whoami are not jupyter magic commands

[7]:
```
#%%bash
#
#date >> first.txt
#whoami >> first.txt
```

1.4.1 List files and directories

Files live within directories. You can see a list of the files in your "primer" directory (which should be your current working directory) by typing:

```
ls
```

[8]:
```
#%%bash
#ls
```

1.4.2 Display file content

You can view a text file with the following command:

```
cat first.txt
```

("cat" is short for concatenate - you can use this to display multiple files together on the screen.) If you have a file that is longer than your 24-line console window, use instead "more" to list one page at a time or "less" to scroll the file down and up with the arrow keys. Don't use these programs to try to display binary (non-text) files on your console - the attempt to print the non-printable control characters might alter your console settings and render the console unusable.

[9]:
```
#%%bash
#cat first.txt
```

- Copy file "first" using the following command:

```
cp first.txt 2nd.txt
```

By doing this you have created a new file named "2nd.txt" which is a duplicate of file "first.txt". Geet he file listing with:

```
ls
```

[10]:
```
#%%bash
#cp first.txt 2nd.txt
#ls
```

- Now rename the file "2nd" to "second":

  ```
  mv 2nd.txt second.txt
  ```

  Listing the files still shows two files because you haven't created a new file, just changed an existing file's name:

  ```
  ls
  ```

[11]:
```
#%%bash
#mv 2nd.txt second.txt
#ls
```

If you "cat" the second file, you'll see the same sentence as in your first file:

```
cat second.txt
```

[12]:
```
#%%bash
#cat second.txt
```

"mv" will allow you to move files, not just rename them. Perform the following commands:

```
mkdir sub
mv second.txt sub
ls sub
ls
```

(where ”username” will be your username and ”group” will be your group name). Among other things, this lists the creation date and time, file access permissions, and file size in bytes. The letter ’d’ (the first character on the line) indicates the directory names.

[13]:
```
#%%bash
#mkdir sub
#mv second.txt sub
#ls sub
```

This creates a new subdirectory named ”sub”, moves ”second” into ”sub”, then lists the contents of both directories. You can list even more information about files by using the ”-l” option with ”ls”:

[14]:
```
#%%bash
#ls -l
```

Next perform the following commands:

```
cd sub
pwd
ls -l
cd ..
pwd
```

[15]:
```
#%%bash
## go to sub directory
#cd sub
## current working directory
#pwd
## list files with permissions
#ls -l
## go to parent directory
#cd ..
## current working directory
#pwd
```

Finally, clean up the duplicate files by removing the ”second.txt” file and the ”sub” subdirectory:

```
rm sub/second.txt
rmdir sub
ls -l
cd
```

This shows that you can refer to a file in a different directory using the relative path name to the file (you can also use the absolute path name to the file - something like ”/Users/username/primer/sub/second.txt”, depending on your home directory). You can also include the ”..” within the path name (for instance, you could have referred to the file as ”../primer/sub/second.txt”).

[16]:
```
#%%bash
#rm -f sub/second.txt
#rmdir sub
#ls -l
#cd ..
#rm -rf primer
```

1.5 Connect to a server

Remote login to another machine can be accomplished using the ”ssh” command:

```
ssh -l mylogin host
```

or

```
ssh mylogin@host
```

where ”myname” will be your username on the remote system (possibly identical to your username on this system) and ”host” is the name (or IP address) of the machine you are logging into.

Transfer files between machines using ”scp”. - To copy file ”myfile” from the remote machine named ”host”:

```
scp myname@host:myfile .
```

- To copy file ”myfile” from the local machine to the remote named ”host”:

  ```
  scp myfile myname@host:
  ```

- Use `ssh -r` option to copy a directory (The ”.” refers to your current working directory, meaning that the destination for ”myfile” is your current directory.)

1.5.1 Exercise

- Copy a file to the server `svmass2.mass.uhb.fr`
- Log on to this server and display this file with `cat`

1.6 Secure copy (scp)

Synchronize big-data directory on the cluster:

```
scp -r big-data svmass2:
```

This a secure copy of big-data directory to the server.

or

```
rsync -e ssh -avrz big-data svmass2:
```

It synchronizes the local directory big-data with the remote repository big-data on svmass2 server

1.7 Summary Of Basic Shell Commands

```
% pico myfile               # text edit file "myfile"
% ls                        # list files in current directory
% ls -l                     # long format listing
% touch myfile              # create new empty file "myfile"
% cat myfile                # view contents of text file "myfile"
% more myfile               # paged viewing of text file "myfile"
% less myfile               # scroll through text file "myfile"
% head myfile               # view 10 first lines of text file "myfile"
% tail myfile               # view 10 last lines of text file "myfile"
% cp srcfile destfile       # copy file "srcfile" to new file "destfile"
% mv oldname newname        # rename (or move) file "oldname" to "newname"
% rm myfile                 # remove file "myfile"
% mkdir subdir              # make new directory "subdir"
% cd subdir                 # change current working directory to "subdir"
% rmdir subdir              # remove (empty) directory "subdir"
% pwd                       # display current working directory
% date                      # display current date and time of day
% ssh -l myname host        # remote shell login of username "myname" to "host"
% scp myname@host:myfile .  # remote copy of file "myfile" to current directory
% scp myfile myname@host:   # copy of file "myfile" to remote server
% firefox &                 # start Firefox web browser (in background)
% jobs                      # display programs running in background
% kill %n                   # kill job number n (use jobs to get this number)
% man -k "topic"            # search manual pages for "topic"
% man command               # display man page for "command"
% exit                      # exit a terminal window
% logout                    # logout of a console session
```

1.8 Redirecting

Redirection is usually implemented by placing characters <,>,|,>> between commands.

- Use > to redirect output.

  `ls *.ipynb > file_list.txt`

  executes `ls`, placing the output in file_list.txt, as opposed to displaying it at the terminal, which is the usual destination for standard output. This will clobber any existing data in file1.

[17]:
```
#%%bash
#ls *.ipynb > file_list.txt
```

- Use < to redirect input.

  `wc < file_list.txt`

executes `wc`, with file_list.txt as the source of input, as opposed to the keyboard, which is the usual source for standard input.

1.8.1 Python example

[18]:
```
%%file test_stdin.py
#!/usr/bin env python
import sys

# input comes from standard input
k = 0
for file in sys.stdin:
    k +=1
    print('file {} : {}'.format(k,file))
```

```
Overwriting test_stdin.py
```

[19]:
```
# %%bash
# python test_stdin.py < file_list.txt
```

You can combine the two capabilities: read from an input file and write to an output file.

[20]:
```
# %%bash
# python test_stdin.py < file_list.txt > output.txt
```

[21]:
```
# %%bash
# cat output.txt
```

To append output to the end of the file, rather than clobbering it, the >> operator is used:

date >> output.txt

It will append the today date to the end of the file output.txt

[22]:
```
# %%bash
# date >> output.txt
# cat output.txt
```

1.9 Permissions

Every file on the system has associated with it a set of permissions. Permissions tell UNIX what can be done with that file and by whom. There are three things you can (or can't) do with a given file: - read, - write (modify), - execute.

Unix permissions specify what can 'owner', 'group' and 'all' can do.

If you try ls -l on the command prompt you get something like the following:

```
-rw-r--r--  1 navaro  staff   15799  5 oct 15:57 01.MapReduce.ipynb
-rw-r--r--  1 navaro  staff   18209 12 oct 16:04 02.Containers.ipynb
-rw-r--r--  1 navaro  staff   37963 12 oct 21:28 03.ParallelComputation.ipynb
```

Three bits specify access permissions: - **r** read, - **w** access, - **w** execute.

1.9.1 Example

`rwxr-xr--`

- the owner can do anything with the file,
- group owners and the can only read or execute it.
- rest of the world can only read

1.10 chmod

To set/modify a file's permissions you need to use the chmod program. Of course, only the owner of a file may use chmod to alter a file's permissions. chmod has the following syntax:

```
chmod [options] mode file(s)
```

- The 'mode' part specifies the new permissions for the file(s) that follow as arguments. A mode specifies which user's permissions should be changed, and afterwards which access types should be changed.
- We use `+` or `-` to change the mode for owner, group and the rest of the world.
- The permissions start with a letter specifying what users should be affected by the change.

Original permissions of script.py are `rw-------`

- `chmod u+x script.py` set permissions to `rwx------`
- `chmod a+x script.py` set permissions to `rwx--x--x`
- `chmod g+r script.py` set permissions to `rwxr-x--x`
- `chmod o-x script.py` set permissions to `rwxr-x---`
- `chmod og+w script.py` set permissions to `rwxrwx-w-`

1.11 Pipelining

```
ls | grep ipynb
```

executes `ls`, using its output as the input for `grep`.

1.11.1 Exercice 11.1

- Pipe `cat *.ipynb` output to `sort` command.
- Pipe `ls` output to `wc` command.
- Pipe `cat 11.UnixCommands.ipynb` to `less` command.

1.12 Chained pipelines

The redirection and piping tokens can be chained together to create complex commands.

1.12.1 Exercice 11.2

Use unix commands chained to display word count of file `sample.txt`.

Hints:

- `fmt -n` takes text as input and reformats it into paragraphs with no line longer than n.
- `sort` sort the output alphabetically
- `tr -d str` delete the string str from the output
- `uniq -c` writes a copy of each unique input and precede each word with the count of the number of occurences.

[23]:
```
from lorem import text
with open('sample.txt', 'w') as f:
    f.write(text())
```

1.12.2 Exercice 11.3

- Create a python script mapper.py to count words from stdin. The script prints out every word found in stdin with the value 1 separate by a tab.

  ```
  Consectetur 1
  adipisci    1
  quiquia 1
  sit 1
  ```

File `mapper.py` must be executable.

[24]:
```
# %%bash
# chmod +x mapper.py
```

1.12.3 Exercice 11.4

- Create a python script reducer.py to read output from mapper.py. The script prints out every word and number of occurences.

  ```
  cat sample.txt | ./mapper.py | ./reducer.py
  ```

```
7   porro
7   eius
6   non
6   dolore
```

[25]:
```
# %%bash
# chmod +x ./reducer.py
```