PHZ3150: INTRODUCTION TO NUMERICAL COMPUTING

FALL 2021

Shell commands 101

Open a terminal and let's start by going to your home directory:

Tip: The **tab** key is a powerful tool when you use the command line. Start typing something on the command line and use the tab key to let your computer finish the path for you. For example, if you want to concatenate (see later) the content of a file `/mnt/c/test.txt`
you can type: `cat /mnt/c/te` —> press tab —> if you only have one file starting with "te" the line will autofill to `/mnt/c/test.txt` . If you have more files starting with "te" it will list those files, so that you see what options there are (if there are too many it will ask you if you want to see them all). Try it out!

Tip: If you want to retype a command you typed before use your up/down arrows to revisit it.

Tip: If you have an external mouse copy-pasting becomes very easy in Unix. Highlight the text you want to copy, go to the file you want to paste it and press the scroll wheel to paste the text.

`cd` : change directory. Tells your computer which directory you want to go to.
Try to go to your phz3150/ directory:

```
cd phz3150
```

`ls` : list the contents of a folder. Try to `ls` your `phz3150/` directory. Assuming that you are in the `phz3150/` from the previous step, do:

```
ls
```

variations of `ls` are:

- `ls -l` which will show you a long format list with all permissions
- `ls -la` which will also show you hidden files (e.g., at your home directory that should show you the .bash_* files)
- `ls -lh` it will show you the file sizes in human readable format
- `ls -lt` it will short the list by time and date
- `ls -ltr` it will short the list by time and date in reverse order (great if you want to check some last additions to your folder)

Try them all out!

`mkdir` : make a directory. Let's make a new directory in your `phz3150/` directory named `shell_test` :

```
mkdir shell_test
```

Test that it is there by listing your current directory with an `ls`

`cp` : copy a file. For example download file 'test.dat' from Webcourses /Files/demos/. Assuming it is at your Downloads/ copy it to your shell_test directory:
for Linux/Mac the "~" points to your home) :

```
cp ~/Downloads/test.dat shell_test/
```
for Windows:
```
cp Windows/Downloads/test.dat shell_test/
```

variations of `cp` are:

- **`cp -r`** for copying a complete folder
- **`cp -i`** copy interactively, it will prompt before copying to ask if you want to copy the file
- **`cp -a`** copy a folder

For example, assuming that you are still on the `phz3150/` directory try to copy the folder `shell_test/` to `shell_test_cp/` :
Try: `cp shell_test/ shell_test_cp/`
since `shell_test/` is a folder, this should give you an error along the lines:
```
cp: shell_test/ is a directory (not copied).
```

Now try it with the -r option:
`cp -r shell_test/ shell_test_cp/`
and then do an `ls`. This should show you now both `shell_test/` and `shell_test_cp/`.

`rm` : remove a file. You should be very careful how you use this command as if going wrong, it can go “really wrong” (like removing your entire system files wrong….).

- **`rm -i`** for removing files interactively
- **`rm -r`** for removing a complete folder
- **`rm *.pdf`** for removing all files ending with .pdf (you can change with any extension)
- **`rmdir`** for removing empty directories

Let’s rm the `shell_test_cp/`:
```
rm -r shell_test_cp/
```

If you now do an `ls` you should only see the `shell_test/` . Let’s go back a step and copy `shell_test/` to `shell_test_cp1/` and `shell_test_cp2/`
Now if you do an `ls` you should see `shell_test/ shell_test_cp1/ shell_test_cp2/`.
Let’s remove both copies:

```
rm -r shell_test_cp1/ shell_test_cp2/
```

If you now do an `ls` you should again only see the `shell_test/` .

Copy `shell_test_cp/` to `shell_test_cp1/` and try to remove it with `rmdir` . Since it has a file in it you should get an error message

```
"rmdir: shell_test_cp1: Directory not empty"
```

Remove file `test.dat` from `shell_test_cp1` and try again the `rmdir` on `shell_test_cp1/`.

`mv` : moves a file or folder from place 1 to place 2. For example, download test_less.dat from Webcourses /Files/demos/. Move it to your `shell_test/` folder:

for Linux/Mac:

```
mv ~/Downloads/test_less.dat shell_test/
```

for Windows:

```
mv Windows/Downloads/test_less.dat shell_test/
```

`cat` : concatenate a file. Now that we know there is a file in shell_test, lets go there and see what it says! Go to `shell_test` (with `cd shell_test`) and concatenate the file `test.dat`:

```
cat test.dat
```

`less` : view the contents of a file without opening it. To quit press `q`. For example, let's read the contents of test_less.dat:

```
less test_less.dat
```

Let's now quit by pressing `q`, and then reopen the file but starting from line 11 were we saw an interesting part:

```
less +11 test_less.dat
```

Quit by pressing `q`.

`file` :to see the type of files you have in a folder. Let's see what type is `shell_test/`. Type:

```
file shell_test/
```

This should give you something like:

```
shell_test/: directory
```

Now let's see what type are all the contents of shell_test/ by typing:

```
file shell_test/*
```

`*` : indicates 'everything'. You may have noticed in some past examples the use of `*` to do something with all files in a folder. For example, to list all your pdf files in a folder type:

```
ls *.pdf
```

or to only copy all pdf files from `folder_1` to `folder_2` type:

```
cp folder_1/*.pdf folder_2/
```

Try to `ls` all `dat` files in your `shell_test/`

wc : does a word count. It gives the number of lines, words and characters in a file. For example, let's check the contents of test.dat and test_less.dat:

```
wc shell_test/test.dat
wc shell_test/test_less.dat
```

Variations are:

`wc -l` if only interested in how many lines there are in the file.
Try it out on `test_less.dat`: `wc -l shell_test/test_less.dat`

`wc -w` if only interested in how many words there are in a file
Try it out on `test_less.dat`: `wc -w shell_test/test_less.dat`

`wc -m` if only interested in how many characters there are in a file
Try it out on `test_less.dat`: `wc -m shell_test/test_less.dat`

`pwd:` shows you your present working directory. Not to be confused with `passwd` that will change your password in Unix systems. Let's try it out:

```
pwd
```

`.` : indicates the present location. So if you want to copy something from `shell_test/test.dat` to your current directory (should still be phz3150/ directory) do:

first test that you don't already have the file here:
```
ls
```
then lets copy the file here:
```
cp shell_test/test.dat .
```
lets verify that you now have the file here:
```
ls
```

`..` : indicates one folder-level up. Let's go in the folder `shell_test/` and remove from there the file `test.dat`:

```
cd shell_test/
rm ../test.dat
```

Let's verify that it worked:

```
ls ../
```

you should not be able to see `test.dat` in the list.

`man` : gives you the manual of any command. For example, let's see the manual of `wc`:

```
man wc
```

Type `q` to quit.

`chmod` : changes the modes/ permissions of files and folders. This will be useful for making your Python files executable for running from the terminal, but also when in the future you want to share files with others and want to limit or expand the rights they have for editing the file. Permissions can be defined either using numbers or letters.

Letters: **`r`** is for reading a file, **`w`** for writing, **`x`** for executing. **`u`** is the user, **`g`** the group and **`o`** any other person trying to access a file.
To set a file so that:
the user can read, write, and execute it, members of your group can read and execute it and others may only read it you type:

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

To make an existing file executable for all you can also simply use:

```
chmod +x filename
```

Numbers: **4** stands for reading, **2** is for writing, **1** is for executing, and **0** is for no permission.
Following the previous example, we would use:

```
chmod 754 filename
```

so that the user can read, write, and execute it, members of your group can read and execute it and others may only read it. Or, you could do

```
chmod 764 filename
```

so that the user can read, write, and execute it, members of your group can read and write it and others may only read it.

Let's try it out: Type ls -l to check the permissions on files in your `shell_test/` folder.
Then do: `chmod 754 shell_test/test.dat` and then do another `ls -l` to check the permissions on files. Compare permissions with previous

`ls -l`. Now do: `chmod 760 shell_test/test.dat` and then do another `ls -l` to check the permissions on files. Compare permissions with previous `ls -l`. Now put permissions back to original state (-rw-r—r--).

`zip` : zip a folder with: `zip -r <name>.zip <folder_you_want_to_zip>`. So to zip folder `shell_test` type:

```
zip -r shell_test.zip shell_test
```

To zip a file type: `zip test.zip shell_test/test.dat` (i.e., without the `-r`).

You can also see the contents of the .zip file without opening with:

```
zip -sf shell_test.zip
```

Try it out.

`unzip` : unzip a file or folder. Type `unzip <name>.zip`

`tar` : another form of compressing a folder. You can tar a file/ folder with

```
tar zcf <tar_name>.tar.gz files (or directories)
```

For example let's tar `shell_test`:

```
tar zcf shell_test.tar.gz shell_test
```

To unpack the tar file use:

```
tar zxpf shell_test.tar.gz
```

You can check the contents of the tarball without unpacking it with:

```
tar tf shell_test.tar.gz
```

Note that you can also tar a file without gunzip-ing it (just remove the `z` from the `zcf` or `zxpf` commands), but that just zips the file without any compression.

> : redirect output to a file. For example, let's do an ls of the `shell_test/` and save it in a file named `listed_shell_test.dat` :

```
ls shell_test > listed_shell_test.dat
```

and compare the `listed_shell_test.dat` (with a cat for example) with the `ls` of `shell_test`

`diff` : will show you the differences between two files. Use as

```
diff file1 file2
```

date : gives the current date and time. Useful for your log keeping!

~~~~~~
A (really non-exhaustive) list of other things you may encounter at some point:

head -N: show the heading N lines of a file. For example, you can try:
```
head -10 shell_test/test_less.dat
```

tail -N: show the trailing N lines of a file. For example, try:
```
tail -10 shell_test/test_less.dat
```

echo :to display line of text/string that are passed as an argument. You can use it to, e.g., see your home directory: `echo $HOME`

| : pipelining/ redirecting the output of one command/program/process to another command/program/process for further processing.

>> : append output to an existing file. For example, you can try to append your home directory at the end of `listed_shell_test.dat` with:
```
echo $HOME >> listed_shell_test.dat
```

vim : a text editor that you can use while in the terminal, and allows you to also edit things when remotely logged in to a machine without using X11 applications. It has full keyboard functionality but does require learning some commands to use it (e.g., press `i` for inserting text, `esc` for going back to visual mode, `:q` to quit or `:wq` to write and then quit etc).

vi : a text editor like vim, but with less keyboard versatility.

ssh : for creating a secure encrypted connection between two hosts over an insecure network. You may use it if you want to work with Stokes, UCF's cluster.

awk : a text-processing programming language.

grep : searching plain-text data sets for lines that match a regular expression. You can do, for example, a case-insensitive search (`-i`) for a word (say 'performed') searching recursively in the current directory and in all of its subdirectories (`-R`) in Linux with:
```
grep -iR 'performed' shell_test/
```
This will return that the only file in `shell_test/` where the word 'performed' exists is `shell_test/test_less.dat`, and it will show you the line for context.
~~~~~~