

An Introduction to the Linux Command Line



Non-HiPerGator version

Last updated: September 11, 2024

This version of the handout is for users who do not have a HiPerGator account. There are several options you can use. This should mostly work with any computer with a BASH interpreter.

There are also several free online tools that work. For example:

- GitHub Codespaces
- replit.com

These instructions are focused on using GitHub Codespaces. You will need a GitHub account to use these instructions. Sign up at [GitHub.com](https://github.com)!

Important notes:

- **To run in Codespaces, you will need to be using this repository:** https://github.com/UFIT-RC-Linux-Training/Linux_training
- GitHub provides educational Organizations with 3,000 minutes of Codespaces CPU time per month. Anyone can use this repository, but:
 - Users are limited to 1 Codespace at a time
 - Users are limited to the 2-core VMs
 - Idle timeout is 30 minutes
 - When the 3,000 minutes a month are used up, no more Codespaces will be able to be launched.


In this documentation, we will use the convention of putting text that needs to be substituted for your specific case in *italic font*.

Getting Around in Linux

- File paths (directories or folders): `/`, `/home/Codespace`, `/workspaces/Linux_training`
- `pwd`, `cd`, `ls` (Where am I, change directory, list directory)
- `cp`, `mv`, `rm` (copy, move (also used for rename), delete)
- `more`, `less`, `head`, `tail`, `cat` (examine files)
- `nano`, `vim` (text editors in Linux)

Note that most things in Linux are case-sensitive. This applies to commands and file names. e.g. `CD` `test` will not work to change directories to the "test" folder, while `cd test` will work (assuming there is a folder named "test" in the current folder). Similarly, `Test` and `test` are **different** folders or files.

Making Things Easier

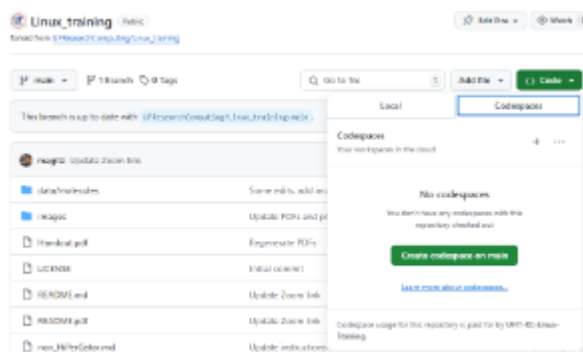
- **Tab completion**- type part of a path or file name and hit tab-key, the shell will auto-complete for you.
- **history**: redo something that you did before without retyping (use  arrow key)
- **man**: getting help. Many built-in Linux applications have **manual** pages that document their use and options that they have. e.g. `man ls` will bring up the page for the `ls` command, documenting the many options that change how the `ls` command functions.
 - Another way to get help information about applications is using the name of the application followed by either the `-h` or `--help` flags. Many applications use this convention to provide documentation to users.

Learning by Doing

(Note: some of the data and examples are taken from software-carpentry.org):

Option 1: GitHub Codespaces

1. Login to GitHub.com
2. Navigate to the repository https://github.com/UFIT-RC-Linux-Training/Linux_training
3. Click the "<> Code" button and select the Codespaces tab:



4. Click the "Create Codespace on main" button.

Option 2: repl.it.com

1. Connect to a repl.it Bash server:
 1. Navigate to <https://repl.it.com/>
 2. Click on the <> **Start coding** button
 3. In the Create new repl Language selection window, type "Bash", select that, and click Create repl.
 4. When the repl opens, for the most part we will focus on the command prompt on the right-hand side. You can resize that pane to get some more room to work.
 5. Copy and paste (*you will probably need to use right-click to paste*) this command to get a copy of these notes and the data for the exercise: `git clone https://github.com/UFResearchComputing/Linux_training.git`

All options should be similar from here

1. Where are you when you login? `pwd`
 1. In Codespaces, it should be `/workspaces/Linux_training`.
 2. The repl.it paths are a bit different. As an example, I see:

```
> pwd
/home/runner/WarlikeBeneficialAutoresponder
>
```

2. What files are there? `ls`

3. Let's make a directory to put some data in: `mkdir cli_demo`

4. Now what's there? `ls -l`

1. Linux commands usually have flags to change how they work
2. `man`, `-h` or `--help` often give you help

5. Change into `cli_demo` directory: `cd cli_demo` or `cd` `cl-Tab-key`

6. Copy some demo data here (`.`):

- In Codespaces: `cp -r ../data/molecules .`
- In replit: `cp -r ../Linux_training/data/molecules .`
 1. Note the `-r` to recursively copy, since `cp` won't copy directories by default
 2. Also note the `"."` at the end to copy the molecules directory to your current location.

You could use the copy in the repository you cloned, but this keeps the directions in parallel with the directions for users on HiPerGator and practices the `mkdir` and `cp` commands. This folder originated from the [software-carpentry Shell Training materials](#).

7. Check that the copy worked: `ls`

8. Change directories into the molecules directory: `cd molecules`

9. Look at these files with `more`, `cat`, `head`, `tail`:

1. `more propane.pdb` and `cat propane.pdb`
2. `head propane.pdb` or `head -n2 propane.pdb`
3. `tail propane.pdb` or `tail -n2 propane.pdb`

10. **Redirects:** You can redirect the output of a command to a file with the `">"` character (see below for more information about STDIN, STDOUT and STDERR).

Caution: Redirection with `">"` first erases the file you are redirecting to if it exists, replacing it with the new contents. **You can append to a file with `">>"`.**

1. As an example, we can get the lengths, in number of lines, of all the files ending in `".pdb"` using the wildcard `"*"` and redirect the output to a file: `wc -l *.pdb > lengths.txt`
2. Let's see what this file looks like: `cat lengths.txt`

Wildcard expansion (e.g. `*.pdb`) is a convenient way to operate on a list of files matching some characteristic: e.g. having the `.pdb` ending, starting with `"p"` (`p*`), etc. The wildcard is evaluated and the list of matching files is passed into the command.

11. **Sorting:** We might want the lengths sorted: `sort -n lengths.txt`

1. What happens without the `-n`?

12. **Pipes:** We can connect commands together by piping the output from one command into the input for the next command using the vertical bar character `|`:

The pipe character, `|`, is typically located above the Enter-key on US keyboards with the `"\`" and is accessed with the Shift-key

1. `wc -l *.pdb | sort -n > lengths.txt`

2. Or if we only want to know the shortest file: `wc -l *.pdb | sort -n | head -n1`

Why this works: Most Linux programs can take input from what is called "standard in", often abbreviated as "STDIN". In addition, they typically have two output streams, "standard out", "STDOUT", and "standard error", "STDERR"--both of which print to the screen by default, but can be redirected as we saw above (in fact we only redirect STDOUT with the `>`"; any STDERR would still print to the screen).

So, while you can run a command like `wc -l propane.pdb`, you can also get the same result by running the `cat` command and "piping" the STDOUT of that to the STDIN of `wc`: `cat propane.pdb | wc -l`. Similarly, you can keep piping commands connecting the output of one command to the input for the next command.

13. **grep:** We can search for text using `grep`:

1. `grep ATOM propane.pdb`

14. **awk:** `awk` can do a lot, but one thing it's good it is pulling out columns from files, in this case the 3rd column:

1. `grep ATOM propane.pdb | awk '{print $3}'`

15. **uniq:** `uniq` is a command to find unique entries in a **sorted** list: `grep ATOM propane.pdb | awk '{print $3}' | sort | uniq`

16. **Loops:** One of the great things about the command line is the ability to automate repetitive tasks. Let's say we want to verify that all our molecules are hydrocarbons (made up of only C and H atoms). Below is what the loop looks like neatly spaced out as if it were in a script:

```
for molecule in *.pdb
do
    echo $molecule
    grep ATOM $molecule | awk '{print $3}' | sort | uniq
done
```

As we type this in on the command line, once you hit the Enter-key at the end of the first line, Bash gives you the continuation prompt, `>`, essentially telling you that you have started a command (a for loop), but not finished it:

```
$ for molecule in *.pdb  
>
```

Note: In replit.com, the Bash prompt is a symbol very similar to the `>` sign.

You can keep typing the rest of the lines, so that it looks like this:

```
$ for molecule in *.pdb  
> do  
> echo $molecule  
> grep ATOM $molecule | awk '{print $3}' | sort | uniq  
> done
```

Once you hit the Enter-key after the "done", that completes the for loop and Bash executes it, showing the output.

If you find yourself stuck in the continuation prompt mode, you can use the key combination **Ctrl-C** to cancel and exit back to the main command prompt. **Ctrl-C** will usually cancel execution of a program in Bash and is a handy key-combination to know!

17. **Deleting files:** Let's get rid of the lengths.txt file: `rm lengths.txt`

1. That file is now gone!! There is no undo, no recycle bin or trash can. As soon as you type the command and hit return, the file is gone!
2. Be careful... but don't keep everything forever either!

Additional exercises

- Which molecule has the most H atoms?
- Make a directory in your `cli_demo` folder and copy the `methane.pdb` file there (preferably without moving from the `molecules` directory)
- From the `molecules` directory, get the `head` of the `methane.pdb` file in the directory you created above.
- Change directories to the directory you made above, rename the `methane.pdb` file to `my_methane.pdb`.
- Edit the `my_methane.pdb` file and put your name as the AUTHOR. I typically recommend the `nano` text editor on the command line for new users: `nano my_methane.pdb` and use the arrow keys to move around. The commands at the bottom of the screen use the Ctrl-key in combination with another key, so `^X` means hold down the Ctrl-key and the X-key at the same time to exit.
- How many ATOMS does methane have?