Command Line Primer
Software Engineering for Scientist

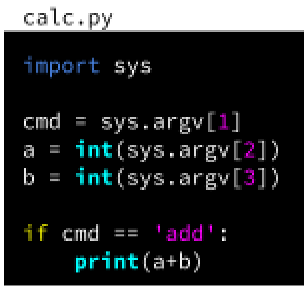
The command line is a text interface to your computer or a remote server. Commands (programs) are executed by typing out the program name and options. When a data set's size or compute resource demands surpass what is available on a PC, work must be moved to remote servers and clusters which often run a form of Unix or Linux and the easiest way to access them is through the command line. A lot of science is done on remote servers and a lot of scientific software is written for the command line. Here are the basics.

Anatomy of a command

A command is typed into a prompt (the part that ends with `$`). That command consists of a program and a list of parameters. Once you are done typing your command, hit return and the command line interface will, in turn, execute the program that you specified and pass it the parameters you gave. The program takes those parameres and makes decisions based on its internal logic.

For example, the command that executes the `calc.py` python script is:

```
$ python calc.py add 5 2
7
```



calc.py

```
import sys

cmd = sys.argv[1]
a = int(sys.argv[2])
b = int(sys.argv[3])

if cmd == 'add':
    print(a+b)
```

In this command, the program is `python` and the options are `calc.py`, `add`, `5`, and `2`. In executing this command, the python interpreter executes the code in `calc.py` and passes it three parameters (`add`, `5`, and `2`). Based on those parameters, `calc.py` prints 7.

File system navigation

When navigating the file system, you are always *in* a directory (i.e., a folder). That directory is called the **working directory**. When you first log in the working directory is typically your **home directory**, which is a directory in a larger **file system** that is structured like most other file systems. Files in directories, directories in directories. You will have permission to read and write in your home directory. In most cases, other users cannot read and write in your home directory.

To see where you are, use the pwd command.

```
$ pwd
/home/ryan
```

`/home/ryan` is the full **path** to the `ryan` directory, which is the home directory of the user named `ryan`. The `ryan` folder is in the `home` folder. The `home` folder is at the top-level (root level) of the file system. Suppose the file `data.txt` was in the `ryan` folder. The full path (**absolute path**) to that file that would be `/home/ryan/data.txt`. If the working directory was `/home`, then the **relative path** to that file is `ryan/data`

Other navigation commands

`ls`	list the contents of the working directory	`$ ls`
		`data.txt`

`ls -l`	get extra details in that list	`$ ls -l`

```
rw-r--r--  1 ryan  dev  11164 Jun 24 12:13 x.x
```

`ls /home` list the contents of a different directory

```
$ ls /home
ryan alice laura
```

`cd` change directories

```
$ cd /home
$ pwd
/home
$ cd
$ pwd
/home/ryan
```

`mkdir` make a directory

```
$ mkdir data
$ ls
data
```

`rmdir` remove a directory

```
$ rmdir data
```

File manipulation commands

`cat` print the full contents of a file

```
$ cat data.txt
date,county,state,fips,cases,deaths
2020-01-21,Snohomish,Washington,53061,1,0
2020-01-22,Snohomish,Washington,53061,1,0
2020-08-23,Uinta,Wyoming,56041,283,2
2020-08-23,Washakie,Wyoming,56043,107,5
2020-08-23,Weston,Wyoming,56045,12,0
```

`head` see the first few lines of a file

```
$ head data.txt
date,county,state,fips,cases,deaths
2020-01-21,Snohomish,Washington,53061,1,0
2020-01-22,Snohomish,Washington,53061,1,0
```

`tail` see the end of a file

```
$ tail data.txt
2020-08-23,Uinta,Wyoming,56041,283,2
2020-08-23,Washakie,Wyoming,56043,107,5
2020-08-23,Weston,Wyoming,56045,12,0
```

`cp` copy a file

```
$ cp data.txt /tmp/
$ ls
data.txt
$ ls /tmp
data.txt
```

`mv` move a file

```
$ mv data.txt /tmp/
$ ls
$ ls /tmp
data.txt
```

`rm` remove a file

```
$ ls /tmp
data.txt
$ rm /tmp/data.txt
$ ls /tmp
```

Other useful commands

`grep` search for lines contain a string

```
$ grep Boulder us-counties.csv
2020-03-14,Boulder,Colorado,08013,1,0
2020-03-15,Boulder,Colorado,08013,7,0
2020-03-16,Boulder,Colorado,08013,7,0
2020-03-17,Boulder,Colorado,08013,8,0
```

`cut` extract particaurly fields

```
$ cut -d "," -f 3,5 us-counties.csv
state,cases
Washington,1
Washington,1
Washington,1
Illinois,1
```

`sort` order the input by a set of fields

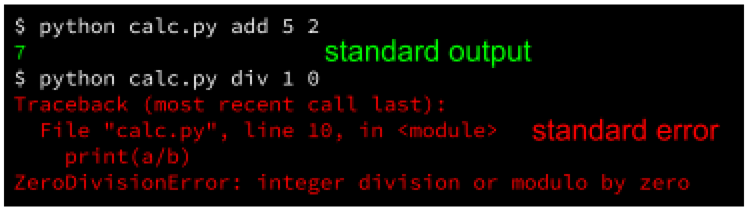
```
$ sort -r -n -k 5 -t "," us-states.csv
2020-08-23,New York,36,434462,32482
2020-08-22,New York,36,433881,32464
2020-08-21,New York,36,433230,32456
2020-08-20,New York,36,432523,32451
2020-08-19,New York,36,431924,32451
```

`wc` count nuber of lines, words, and/or characters

```
$ wc us-states.csv
    9584   12254  314630 us-states.csv
```

I/O redirection

Program input and output in the command-line environment occurs in three streams: **standard input** (**STDIN**), **standard output** (**STDOUT**), and **standard error** (**STDERR**).


```
$ python calc.py add 5 2
7
$ python calc.py div 1 0
Traceback (most recent call last):
  File "calc.py", line 10, in <module>
    print(a/b)
ZeroDivisionError: integer division or modulo by zero
```

Standard input streams data to a program, which could be from a keyboard or another file (does not include command-line parameters). When a program prints a value (i.e., `print(a+b)`), that message will follow the standard output stream and typically end up displayed on the screen. Similarly, standard error carries error messages (typically) to the screen.

While these streams have a default destination, they can be redirected to files.

`>` overwrites a file with standard output

```
$ python calc.py add 5 2 > math.out
$ cat math.out
7
```

`>>` appends a file with standard output

```
$ python calc.py add 3 2 >> math.out
$ cat math.out
7
5
```

`2>` overwrites a file with standard error

```
$ python calc.py div 1 0 2> math.err
$ cat math.err
```

```
Traceback (most recent call last):
  File "calc.py", line 10, in <module>
    print(a/b)
ZeroDivisionError: integer division or modulo by zero
```

`2>>` appends a file with standard error

```
$ python p.py div 1 a 2>> math.err
$ cat math.err
Traceback (most recent call last):
  File "calc.py", line 10, in <module>
    print(a/b)
ZeroDivisionError: integer division or modulo by zero
Traceback (most recent call last):
  File "p.py", line 5, in <module>
    b = int(sys.argv[3])
ValueError: invalid literal for int() with base 10: 'a'
```

In addition to redirecting steams to files, the standard output of one program can be sent to the standard input of another program with the **pipe** command. For this to work, the program being pipe to needs to contain logic that allows it to accept input from standard input. Many, but not all, command-line programs can accept input from standard input. For example:

```
$ cat us-counties.csv
date,county,state,fips,cases,deaths
2020-01-21,Snohomish,Washington,53061,1,0
2020-01-22,Snohomish,Washington,53061,1,0
2020-01-23,Snohomish,Washington,53061,1,0
2020-01-24,Cook,Illinois,17031,1,0
```

```
$ cat us-counties.csv | grep Colorado
2020-03-05,Douglas,Colorado,08035,1,0
2020-03-05,Jefferson,Colorado,08059,1,0
2020-03-06,Denver,Colorado,08031,2,0
2020-03-06,Douglas,Colorado,08035,3,0
2020-03-06,Eagle,Colorado,08037,1,0
2020-03-06,El Paso,Colorado,08041,1,0
```

```
$ cat us-counties.csv | grep Colorado | sort -n -k 5 -t "," | head
2020-03-05,Douglas,Colorado,08035,1,0
2020-03-05,Jefferson,Colorado,08059,1,0
2020-03-06,Eagle,Colorado,08037,1,0
2020-03-06,El Paso,Colorado,08041,1,0
2020-03-06,Jefferson,Colorado,08059,1,0
```

```
$ cat us-counties.csv | grep Colorado | sort -n -k 5 -t "," | tail -n 1
2020-08-23,Denver,Colorado,08031,10901,426
```