

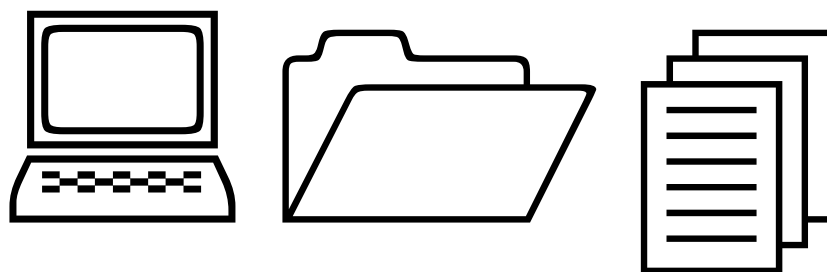
# Survival command line

**for Text Terminal workshops**

## ABSTRACT

This short workshop will provide hands-on overview of commands in a text Terminal. We'll review the handful of commands that are most useful.

JEAN-YVES SGRO



The Biochemistry Computational Research Facility (BCRF) provides access to a computational computer cluster and hands-on tutorials


**Image credits:**

Wingding font icons

Workbook – © 2019 – All Rights Reserved

Corresponding author: Jean-Yves Sgro - [jsgro@wisc.edu](mailto:jsgro@wisc.edu) -  
All rights reserved.

**Instructor:**

	Jean-Yves Sgro, Ph.D
	<p>Distinguished &amp; Data Scientist II. Biotechnology Center &amp; Biochemistry Dept.</p> <p><u>Biochemistry Office:</u> 433 Babcock Drive, room 201 Madison WI 53706</p> <p><u>Email:</u> <a href="mailto:jsgro@wisc.edu">jsgro@wisc.edu</a></p>

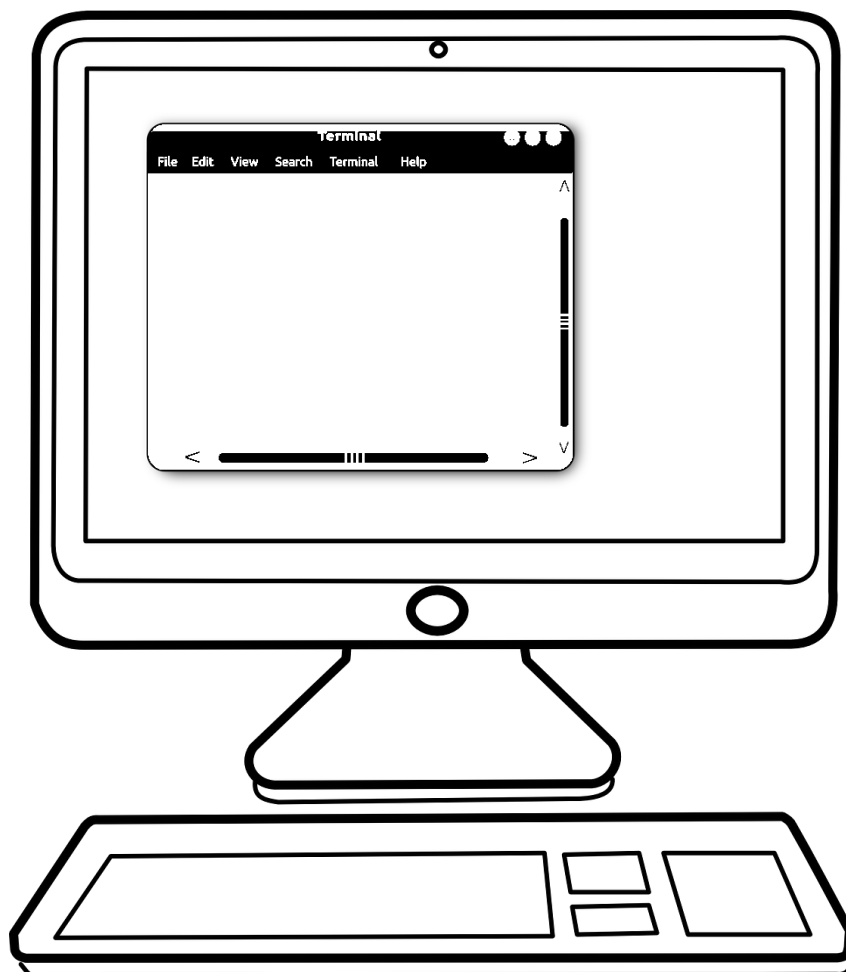
# Survival command-line for Biologists

Jean-Yves Sgro

December 03,2019

<b>INTRODUCTION.....</b>	<b>3</b>
<i>Learning objectives:.....</i>	<i>3</i>
<b>TERMINAL AND SHELL.....</b>	<b>5</b>
<i>Text terminal .....</i>	<i>5</i>
<i>The shell .....</i>	<i>5</i>
<i>Shells.....</i>	<i>6</i>
<b>SET-UP.....</b>	<b>7</b>
<b>WORKING IN TERMINAL .....</b>	<b>8</b>
<i>Username.....</i>	<i>8</i>
<i>Home.....</i>	<i>8</i>
<i>Prompt.....</i>	<i>9</i>
<i>Preferences, \$ and Variables.....</i>	<i>9</i>
<i>Home and username revisited.....</i>	<i>10</i>
<i>Who am I.....</i>	<i>10</i>
<i>Where am I looking.....</i>	<i>11</i>
<b>HARD DRIVE: WHERE THINGS ARE .....</b>	<b>12</b>
<i>Summary so far:.....</i>	<i>13</i>
<b>DIRECTORIES .....</b>	<b>15</b>
<i>New directories.....</i>	<i>15</i>
<i>Path.....</i>	<i>18</i>
<i>Home again.....</i>	<i>18</i>
<b>FILES .....</b>	<b>19</b>
<i>Exploring file contents.....</i>	<i>19</i>
<i>File editing .....</i>	<i>21</i>
<i>Compressed web files.....</i>	<i>22</i>
<i>Section summary:.....</i>	<i>24</i>
<b>STREAMS AND PIPES: KEY CONCEPTS.....</b>	<b>25</b>
<i>Standard input and output.....</i>	<i>25</i>
<i>Data stream and pipes.....</i>	<i>26</i>
<i>Redirection.....</i>	<i>27</i>
<i>Pipes.....</i>	<i>28</i>
<b>REMOTE CONNECTION.....</b>	<b>30</b>
<b>SUMMARY.....</b>	<b>31</b>
<i>Concepts.....</i>	<i>31</i>
<i>Symbols.....</i>	<i>31</i>

<i>Commands learned or mentioned:</i> .....	32
RESOURCES .....	33
<b>REFERENCES</b> .....	<b>34</b>



# Introduction

---

This short tutorial/workshop is meant to review and understand basic command-line as they are typed on a *text terminal*.

This specific workshop will focus on Macintosh Terminal, but most commands would also work on all Unix-style operating system (Linux, or Windows with added software.)

The goal is to review the most useful commands in order to operate the terminal for later workshops including those focused on **Docker**.

## Learning objectives:

The main objective is to become at ease with the command-line to perform routine tasks:

- Open a Terminal
- Understand the computer organization (file structure and “path”)
- Create, delete and navigate directories
- Create, delete, explore and edit text files
- Download Internet data files
- Apply key concepts (standards and streams) to tasks



### digital VT100 text terminal

DEC VT100 terminal at the Living Computer Museum (apparently connected to the museum's DEC PDP-11/70). Introduced in August 1978 by Digital Equipment Corporation (DEC)

Jason Scott - Flickr: IMG\_9976 CC BY 2.0

Source: [https://commons.wikimedia.org/wiki/File:DEC\\_VT100\\_terminal.jpg](https://commons.wikimedia.org/wiki/File:DEC_VT100_terminal.jpg)

# Terminal and Shell

---

## Text terminal

In the early 1990's "dumb terminals" were still used to access a remote, shared computer. The terminal was called "dumb" because it did not contain any operating system. The *digital VT100* ("VT100" 2019) was a very popular terminal:

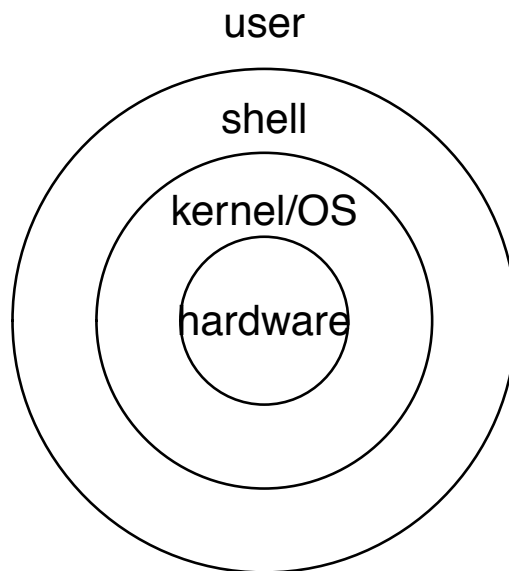
In contrast, a "personal computer" is an "all-in-one" that uses *terminal emulation software* within itself. On a Macintosh it is called `Terminal` and, on a Windows PC `cmd` or `PowerShell`.

These are *emulation* *i.e.* they mimic what the physical terminal used to do: let the user "talk" to the computer with typed commands. These commands are in the "shell language" that resembles English more than "machine language" made of binary or hexadecimal numbers.

## The shell

The *shell* is both a language and a software that takes commands from the keyboard (user input) and transmits them to the "kernel," the part of the operating system that can also "talk" to the hardware. For example, it is the kernel that will instruct the hard drive to make physical changes to record a file you are writing and saving.

This is perhaps why the shell is often represented surrounding both hardware and kernel.



*The shell: an intermediate between user and OS.*

Figure 1.

## Shells

The most current shell is called **bash** (Bourne Again SHell, an enhanced version of the original **sh** by Steve Bourne.) This is the default shell on most Linux systems and on Macintosh until now.

*Wikipedia Note*<sup>1</sup>: The newest MacOS (Catalina) now uses the *Z shell* which is an extended Bourne shell with many improvements, including some features of bash, ksh, and tcsh shells.



**Warning: shell commands are CaSE SenSIitive!**

---

<sup>1</sup> [https://en.wikipedia.org/wiki/Z\\_shell](https://en.wikipedia.org/wiki/Z_shell)



# Set-up

---

We'll use the iMacs from Biochemistry laboratories room 201<sup>2</sup>.

To get started we first need to open a text terminal as detailed below.



## TASK:

**Do one of the following:**

1. Find the `Terminal` icon in the `/Applications/Utilities` directory. Then double-click on the icon and `Terminal` will open.
2. **OR** use the top-right icon that looks like a magnifying glass (*Spotlight Search*,) start typing the word `Terminal` and press return. `Terminal` will open.



---

<sup>2</sup> These commands would also work on a Windows system with added software *e.g.* Ubuntu for Windows, see installation at <https://tutorials.ubuntu.com/tutorial/tutorial-ubuntu-on-windows>

# Working in Terminal

---

Remember that `Terminal` is a *software* version of what used to be a physical *hardware*. `Terminal` itself is “dumb” and only allows you to “talk” to the computer operating system through the shell with the shell language somewhat similar to English.

## Username



You will need to login the iMac with your **NetID** and a username will be created on this Mac is you have not logged in it before.

## Home



When you start `Terminal` you are immediately *connected internally* with the operating system (OS,) here it is MacOS, a derivative of Unix, an OS developed in the 1960's with very powerful features that we'll discover below.

When the connection is established, you “land” in the “home directory” *i.e.* the area on the disk reserved for your *username* (these computers can have multiple -even concurrent- users.)

In most modern Macintosh, Windows and Linux systems the disk areas reserved to users are defined in a similar way as a “high level” directory. We’ll see a bit later how to know exactly where this disk space is located.

## Prompt



When the terminal starts your username will appear together with the computer name followed by a \$. This is called the *prompt* and simply signals that the terminal is ready to accept shell commands. For example, my prompt looks like this:

```
Last login: Wed Nov 27 08:57:02 on ttys000
BIOCNB-01014M:~ jsgro$
```

BIOCNB-01014M is the name of the computer I am using and jsgro is my *username*.

On this line we’ll type shell commands.

## Preferences, \$ and Variables



In most modern software you’d find a menu options called **Preferences...** where you can change predefined choices. In the same way, there exists preference setting for the shell. They are called “*environment variables*” and can sometimes play an important role.

Variables have defined names, usually in uppercase, and the \$ symbol is also used to printout the value of a variable.

For example, we can check the name of the shell that is running in **Terminal** with the shell command **echo** and the specific variable associated with it: **SHELL**. By default, **echo** will simply repeat (print back on screen) what is typed on the keyboard, but with a preceding \$ to the typed word **echo** will print the *value* of a variable itself. This is best understood by practice as suggested below.

*Note:* without the \$ the program **echo** simply repeats what it is given.



**TASK: Run the following commands:**

```
echo SHELL
```

Answer: \_\_\_\_\_

```
echo $SHELL
```

Answer: \_\_\_\_\_

In the same way what is the output of the following commands?

```
echo $USER
```

Answer: \_\_\_\_\_

```
echo $HOSTNAME
```

Answer: \_\_\_\_\_

```
echo $HOME
```

Answer: \_\_\_\_\_

## Home and username revisited

### Who am I



Since we cannot see the directories in a graphical way, it is very useful to start with understanding where we “land” when we first launch `Terminal` and where we are “looking” inside the computer hard drive at any moment. The following commands will give us some clues about this:



**TASK: Run the following command:**

```
whoami
```

Answer: \_\_\_\_\_

This `whoami` command with an “existential feeling” used to be very useful when terminal stations were shared and someone forgot to log-off. In fact,

in modern shells, this information is also found as part of the prompt as we have seen before.

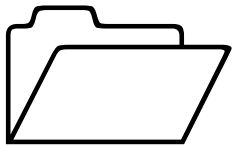
## Where am I looking



**TASK: Run the following command:**

```
pwd
```

Answer: \_\_\_\_\_



This is an important command that shows the *p*resent *w*orking *d*irectory.

In this case it is our “home” directory as we just arrived on the system.

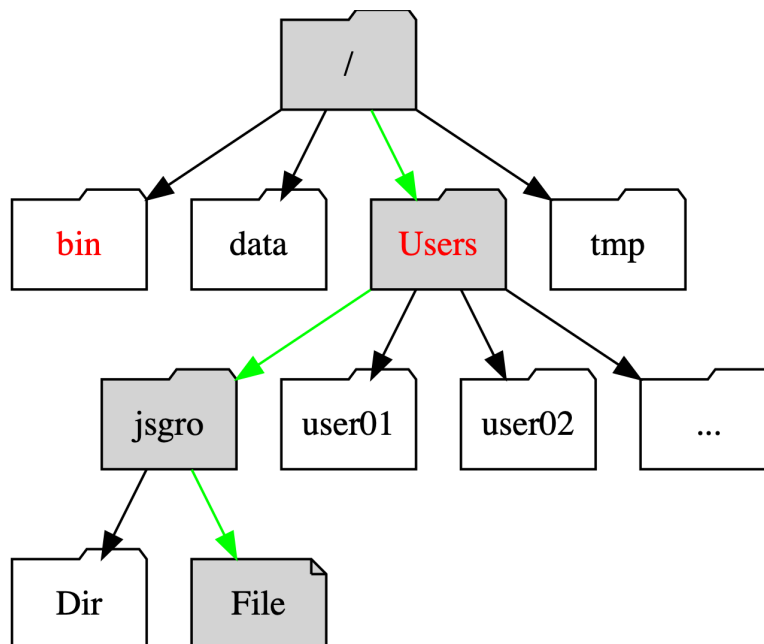
# Hard drive: where things are

---



Your home directory is just one area of the hard drive, that can be shared by the home directory of other users as illustrated below.

This type of organization is sometimes called an “inverted tree.” The top level is called root and it *branches out into* (contains) other directories, subdirectories, and files.



*Hard drive organization.*

Figure 2.

Important notes:

- The home directory shows a *path* (green arrows and grayed folder) from the very top level of the computer system denoted `/` and called the “*root*.” (See more on *path* below.)
- The separator between directory names is also the forward slash `/`
- The complete home directory path can also be replaced by the single symbol `~` (more on this later.)

Summary so far:

Command	Definition
<code>ssh</code>	secure shell connection software
<code>\$</code>	shell prompt: awaiting for command
<code>pwd</code>	print working directory
<code>~</code>	equivalent to home directory path
<code>/</code>	root and separator symbol

`mkdir`

`cd`

`mv`

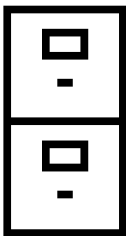
`rmdir`

`ls -a`



# Directories

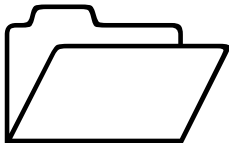
---



Directories as like file cabinets.

In this section we'll learn to create, delete and navigate between multiple directories. We'll also learn to list and delete files.

## New directories



Creating separate directories for various projects is the best way to organize your work (and text files, data files, sub-directories etc.) On your laptop you could easily create a directory with the mouse. But while working in `Terminal`, it is sometimes simpler to just use the `mkdir` shell command to “make a directory” in the location we are looking, currently our home directory.



### *Important Notes:*

- **Avoid blank spaces** in directory and file names. (It is possible but makes them harder to handle.)
- Names are case-sensitive: `A` is not the same as `a`. For example, `Myfile` is not the same as `myFile`.



**TASK: Run the following commands:**

First, we create a directory:

```
mkdir dir01
```

Did I hear **OOPS** ?

Perhaps the name should be something else, perhaps `myproject01`?

At this point there are 2 solutions:

1. **rename** it with the shell `mv` (move) command:
  - `mv dir01 myproject01`
2. **delete** it with shell command `rmdir` (remove directory) and create a new one again:
  - `rmdir dir01`
  - `mkdir myproject01`



**TASK: Choose** one method so that you now have `myproject01` available to you.

OK - now we have a directory to work with.

To work *within* this `myproject01` directory we have to move our focus *into* that directory. For this we use the shell command `cd` that means “change directory.”

```
cd myproject01
```

We can then verify that is where we have now landed with this useful shell command that we have already learned:

```
pwd
```

We should now be within the `myproject01` directory which is empty. However, as part of the operating system, two invisible files are created that are an integral part of the directory. We can see them by adding `-a` to the `ls` command to list **all** file:

```
ls -a
```

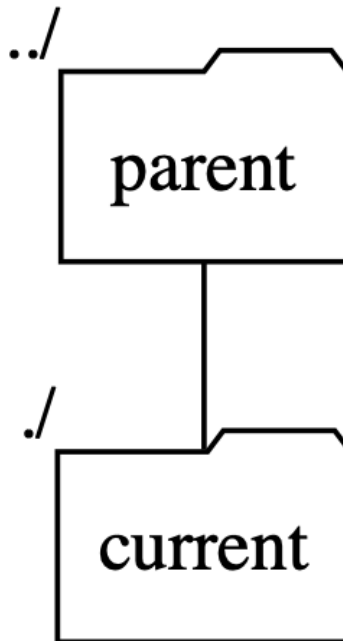
What do you see?

Answer: \_\_\_\_ \_\_\_\_

These two items are the symbolic representation of the “current directory” (dot) and the “parent directory” (dot dot).

<i>Notation</i>	<i>Spoken Name</i>	<i>Definition</i>
.	“dot”	<b>Current</b> directory
..	“dot dot”	<b>Parent</b> directory: directory “above” containing the “current” directory.

We will use this knowledge in the following section.



*Current (dot) and parent (dot dot) directories.*

Figure 3.

*Note:* If you use the command `ls -aF` instead you will see the following result:

```
./ ../
```

The trailing / signifies that these 2 items are in fact folders: the current one and the parent one.

## Path



The existence of `.` and `..` provides that we can specify the location of a file on the system with either an *absolute* or a *relative* path.

**An absolute path** is a description starting from `/` (**root**) which is therefore complete and unambiguous since there is only one **root** within the computer file organization.

For example, in the hard drive organization figure above the *absolute path* to `File` is: `/Users/jsagro/File`.

**A relative path** is a description starting from a folder *other than root*.

Examples:

- `pwd` provides an absolute path starting from the very beginning of the **root** with `/`.
- `ls ..` is a command that will list files from the parent directory *relative* to our current directory location.

## Home again



If you are “lost” on the system of course `pwd` can help, but just typing `cd` will bring you *back into your home directory*.

# Files

---



A file is a “container” of information. Some files contain plain text and are easy to handle and explore on a text-only terminal. More complex files contain binary information that would display as “gibberish” on a text-only terminal.

It is useful to name files with a *filename extension*, for example `.txt` for a plain text file as a reminder of the type of content.

(In addition, on most current operating systems, the filename extension will lead the OS to show the file graphically with a specific icon.)

## Exploring file contents

In this section we’ll learn to download a file directly from the Internet, view top and bottom portions of a text-only file, and finally to read its content one screen at a time.

On MacOS the shell command to copy a web address `curl` can be used to download files directly into the current directory. We’ll download a plain text file listing the known chemical elements named `chemical_elements.txt`. At the same time we’ll change the name of the file into a new, simpler name: `elem.txt` for easier typing.



**TASK: Run the following commands:** (type on a single line)

```
curl -o elem.txt https://static-bcrf.biochem.wisc.edu/tutorials/unix/survival_command/chemical_elements.txt
```

curl

head

tail

rm -r

nano

cat

more

*Note:* if you need to actually *type* you can use this short link instead:  
[tiny.cc/chemelem](https://tiny.cc/chemelem)

*Note:* you can type `ls` to verify that the file has been transferred and that its name is what it should be.

Now we are going to explore the content of the file.

- list the first few lines at the top of the file. The default for `head` is 10 lines, but we can modify that number simply. For example to see the first 5 lines:

```
head -5 elem.txt
```

- In the same way the command `tail` can print the lines from the end of the file. To see the last 3 lines of the file issue the command:

```
tail -3 elem.txt
```

Two other commands are useful:

- `cat` will print the whole content of the file at once, therefore making it impossible to see the top if the file is too long. (In Windows DOS the command would be `Type`.)
- `more` (or its newest incarnation `less`) will show one screen at a time. Pressing Enter will show one more line at a time, while pressing Space bar will show one screenfull at a time. Pressing q will quit and return to the prompt.

*Note:* to remove a file use the command `rm` followed by the name of the file.

*Note:* Earlier we learned `rmdir` to delete an empty directory. However, to remove a directory that is *not empty*, the (dangerous) command is using a recursive method (adding `-r`) for example: `rm -r somedirectory` (**Warning!**- there is **NO UNDO** and *this command will remove everything* thus make sure you are in the right place with e.g. `pwd` before issuing it!)

## File editing

The ability to create your own text files is essential and the full-screen text editor **nano** can be very helpful. This software can be used to open and modify existing files, or to create new text files.

`nano` can open an existing file to **modify** its content or create a new file. Let's create a simple file called `simple.txt` containing just a few lines.

```
cd ~/myproject01
```

```
nano simple.txt
```

This will open a full screen editor. Ctrl command options are shown at the bottom of the screen:

```
GNU nano 2.0.6                      File: simple.txt
- - - - THIS IS THE AREA WHERE YOU TYPE TEXT - - - -
- - - - Use up, down, left, and right arrows - - - -
- - - - to navigate, NOT the mouse! - - - -
- - - - When done, type Ctrl X to exit - - - -

^G Get Help ^O WriteOut ^R Read File^Y Prev Page^K Cut Text
^C Cur Pos
^X Exit      ^J Justify  ^W Where Is ^V Next Page^U UnCut Tex
^T To Spell
```

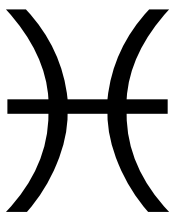
**Write some simple text**, then press Control and X keys at the same time to exit the program and write the new file to the current directory.

On exiting you may have to answer Yes or Y to the questions:

```
Save modified buffer (ANSWERING "No" WILL DESTROY CHANGES) ?
Y Yes
N No                  ^C Cancel
```

*Note:* The command shown as ^O for Control + O (capital letter Oh) would write the current changes but would not close the program but stay within the editing mode for further text editing.

## Compressed web files



We learned the command `curl` above.

It is often necessary to download files from the Internet. These files can also be compressed. The following section is a short exploration on how to handle such matters.



The following example will download a random DNA sequence file from the University of California at Santa Cruz (UCSC) from chromosome 4. The web page is at

<http://hgdownload.cse.ucsc.edu/goldenpath/hg19/chromosomes/>

Here is the command to download file `chr4_gl000194_random.fa.gz` with `curl`:



**TASK:** Run the following commands: (on a single line)

```
curl -o chr4_gl000194_random.fa.gz http://hgdownload.cse.ucsc.edu/goldenpath/hg19/chromosomes/chr4_gl000194_random.fa.gz
```

Note that on Linux system there exists also the command `wget` (“web get”) that can accomplish the same task. In this case the `-o` to specify output is not required. `wget` is not standard on Macs and `curl` is used instead.

```
wget http://hgdownload.cse.ucsc.edu/goldenpath/hg19/chromosomes/chr4_gl000194_random.fa.gz
```

*Note:* You can also go to the short URL equivalent of the web page <https://bit.ly/2kJpA0K> and then “right-click” on file `chr4_gl000194_random.fa.gz` and select “Copy link”

The file is a Fasta sequence format (`.fa`) and is also compressed with the *gnu-zip program* `gzip` and can be decompress by `gunzip` as shown with the following command:

```
gunzip chr4_gl000194_random.fa.gz
```

*Note* that the suffix `.gz` will automatically be removed. Now the file is a simple text file and can be explored as seen previously, *e.g.* with `head`, `tail` and `more`.

*Note:* The files ending with `.zip` should be uncompressed with the `unzip` command as these are *different formats and software*.

## Section summary:

Command	Definition
<code>mkdir</code>	create directory
<code>rmdir</code>	delete empty directory
<code>rm</code>	remove file
<code>rm -r</code>	recursively remove <b>everything</b> within a non empty directory
<code>ls</code>	list content of directory. <code>-a</code> for all, <code>-d</code> for directory etc.
<code>mv</code>	move and/or rename file or directory
<code>cd</code>	change directory
<code>.</code>	current directory
<code>..</code>	parent directory
path	Absolute start with <code>/</code> . Relative uses <code>.</code> and/or <code>..</code>
TAB	computer finishes typing
<code>head</code>	show first 10 lines by default
<code>tail</code>	show last 10 lines by default
<code>cat</code>	type all content of file on screen
<code>more</code>	type file content one screen at a time. Newer: <code>less</code>
<code>nano</code>	text editor
<code>curl</code>	obtain file from web address
<code>wget</code>	obtain file from web address

# Streams and pipes: key concepts

---

One of the reasons that *Unix-like* systems and its shells have withstood *Time* is the inherent power of a few key concepts detailed below.

## Standard input and output




*Unix-like* systems comprise Unix, Linux and MacOS operating systems.

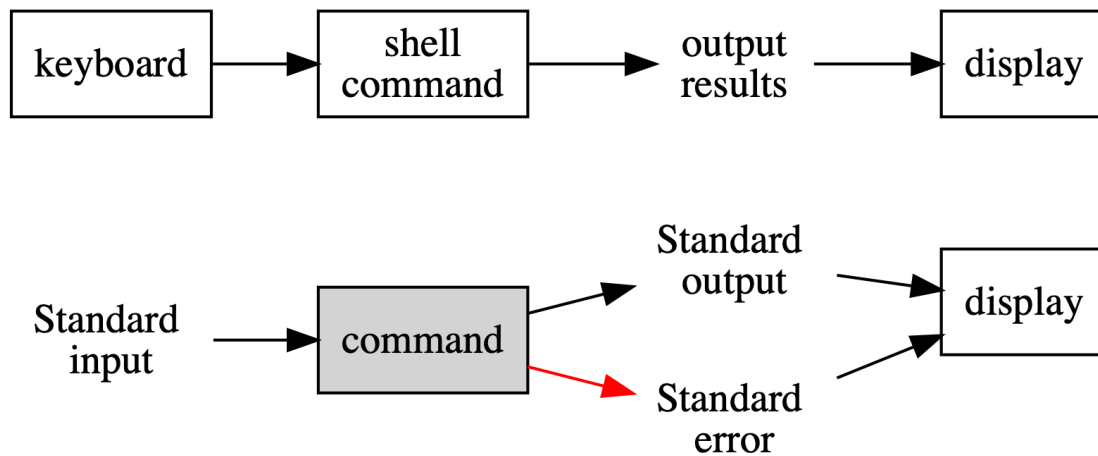
- The default **standard input** is your **keyboard**.
- The default **standard output** is the screen

**Standard output** (which is printed on the screen by default) is split between the normal output (**stdout**) and an error output (**stderr**) in case errors are to be reported.

There are therefore three “streaming” channels for Input/Output (I/O) labeled from zero to 2:

*Understanding I/O stream numbers*

Handle	Name	Description	
0	<b>stdin</b>	Standard input	
1	<b>stdout</b>	Standard output	
2	<b>stderr</b>	Standard error	



*Standard Input/Output channels.*

Figure 4

The above figure illustrates the standard input and output concepts.

On the top image, the *keyboard* (**stdin**) is used to enter a shell command (e.g. list the content of a directory with `ls`) which creates an output. The output is sent to the *standard output* (**stdout**) which is the computer screen by default.

The bottom image further shows that the output represents 2 channels, one for the normal output and another for error output.

## Data stream and pipes

We can redefine the 3 standards above with the added notion of “stream” as *preconnected input and output communication channels* Ritchie (1984), (“Standard Streams” 2019):

- Standard input is stream data (often text) going into a program.
- Standard output is the stream where a program writes its output data.
- Standard error is another output stream typically used by programs to output error messages or diagnostics.

One can imagine data *flowing* from *input* to *process* to *output*.

More importantly the data could be at any point *redirected* (hijacked) to a different destination.

## Redirection

**The first example is redirecting** the *standard output*, normally destined to appear onto the screen, into a file that will be saved onto the computer. This is accomplished with the redirect `>` symbol.

As example we can create a new file called `5lines.txt` based on a previous `head` example above. In this case nothing will appear on the screen and a new file will be created instead, *containing what would have been shown on the screen*:



**TASK: Run the following commands:**

```
head -5 elem.txt > 5lines.txt
```

We can verify that this is the case:



```
ls  
cat 5lines.txt
```

*Important Note:* using the same command with `>` would overwrite any existing file. To instead add to the end of the file (*append*) one should use the double redirect `>>` instead.

One useful use of this method is the ease of creating text files without a text editor. We can use the command `cat` that normally takes the content of a file and sends it to standard output. Instead of a file we can use the keyboard input and redirect the data into a file.

```
cat > example.txt
```

Note that there is no \$ prompt visible.

Here we write some text that will be saved to a file named example.txt

Everything we write here is redirected into the file (redirected standard output.)

We can add as many lines as we want.

BUT WHEN DONE WE NEED TO PRESS: CTRL-D as a signal that we are done!

CTRL-D means "end of file" and is like "closing the file", ending the redirection and returning us to the \$ prompt.

Again: CTRL + D will ends this process and return us to the \$ prompt.

## Pipes

The *data stream* issued from the process of a program can also be redirected in a different way with the pipe symbol |. In this case the *data stream* from one program coming out through *standard output* will be used a *standard input* by the next program. Multiple processes can therefore be connected in a single pipeline, as illustrated below:

```
process1 | process2 | process3 | process4
```

The process could be any running program that has proper *standard input* and *standard output* compliance. As an example we can use the utility `wc` (word count) to count the number of words of a data stream:

### TASK:

Run the following commands:

```
head -5 elem.txt | wc
```

Answers: \_\_\_\_

The three numbers represent the number of:

- lines,
- words and
- characters (including the return character.)

This other example uses the program `grep` to recognize a simple text *pattern*, here the pattern is `ron`, chosen to select only the lines that contain this pattern *within* any of the words on each line. The program `cat` sends the content of file `elem.txt` into the data stream as *standard input* which is then “piped” into program `grep`:

```
cat elem.txt | grep ron
```

Answers: B \_\_\_\_\_

Answers: I \_\_\_\_\_

Answers: S \_\_\_\_\_

We could even string these processes in a longer pipeline where we accomplish 3 tasks:

- send the data from `elem.txt` into the data stream. From `cat` the data arrives as part of the standard output, which serves as standard input to the next program `grep`.
- `grep` processes the data to recognize the pattern `ron`.
- finally, `wc` does the lines, word and character counting. The final standard output arrives onto the screen.

```
cat elem.txt | grep ron | wc
```

Here is the result you should obtain:

```
3      3      21
```

Of course, we could have also recuperated the final output into a file instead!

```
cat elem.txt | head -5 | grep ron | wc > count_5lines.txt
```

# Remote connection

---

In some cases, you may want (or need) to connect to a remote computer. In this case the remote computer is most likely to be a Linux cluster. For this we would use the command `ssh` which means “Secure SHell” as all connections and transmissions are encrypted (hence secure.)

In the Biochemistry department the connection to the Linux cluster required a **NetID** as well as prior authorization.

If you are authorized to connect, the command would have this form, where *myname* is your *NetID*.

```
ssh myname@submit.biochem.wisc.edu
```

You would be then prompted to accept the connection and prompted for your password. For further details refer to the Biochem web site.<sup>3</sup>

---

<sup>3</sup> <https://bcrf.biochem.wisc.edu/bcc/>



# Summary

---

## Concepts

Concept	Definition
Standard input	Default: the keyboard. Input piped data
Standard output	Default: the screen display. Redirect to file or pipe
Standard error	Default: the screen display.
Redirect	Take standard input and send to file with > or >>
Pipe	Take standard output and pass to next command as standard input with

## Symbols

### *Symbols and filters*

Symbol	Meaning
\$	Shell prompt
\$	Add to variables to extract value: <i>e.g.</i> <code>echo \$SHELL</code>
~	Shortcut for home directory
/	Root directory. Separator on path names
>	Single redirect: sends <i>standard output</i> into a named file.
>>	Double redirect: appends <i>standard output</i> to named file.
	Pipe: transfers standard output to next command / software.

## File descriptors

File	Meaning
.	Current directory. Can be written as ./
..	Parent directory. Can be written as ../
/dev/stdin	Standard input
/dev/stdout	Standard output
/dev/stderr	Standard error

## Commands learned or mentioned:

### Commands learned

Command	man page definition and/or example
echo	write arguments to the standard output. <code>echo \$SHELL</code>
whoami	display effective user id.
pwd	return working directory name.
cd	change directory
ls	list directory contents. <code>ls -F</code> , <code>ls -a</code>
wget	grabs a file from Internet with provided web address
curl	grabs a file from Internet with provided web address
unzip	list, test and extract compressed files in a ZIP archive.
mkdir	make directories.
mv	move files. (Can rename file/ directory in the process.)
cat	types file onto screen (or sends to <i>standard output</i> .)
head	display first lines of a file. Default 10.
tail	display the last part of a file. Default 10
nano	(Text editor) Nano's ANOther editor, an enhanced free Pico clone.
wc	word, line, character, and byte count.
rm	remove directory entries <i>i.e.</i> remove files. Remove non-empty dir with <code>rm -r</code>
rmdir	remove directories (empty dirs)
cp	copy files.

## Resources

I have selected the following resources, but you would find many more with a simple web-engine search.

### *Online resources*

<b>Name of Tutorial</b>	<b>URL</b>	<b>Archived</b>
UNIX Tutorial for Beginners	<a href="http://www.ee.surrey.ac.uk/Teaching/Unix/">http://www.ee.surrey.ac.uk/Teaching/Unix/</a>	<a href="http://bit.ly/1pixR8C">http://bit.ly/1pixR8C</a>
Unix Basics	<a href="https://www.ntu.edu.sg/home/ehchua/programming/howto/Unix_Basics.html">https://www.ntu.edu.sg/home/ehchua/programming/howto/Unix_Basics.html</a>	<a href="https://bit.ly/2lzyYo9">https://bit.ly/2lzyYo9</a>
Linux Tutorial	<a href="https://ryanstutorials.net/linuxtutorial/">https://ryanstutorials.net/linuxtutorial/</a>	<a href="https://bit.ly/2lzCtuN">https://bit.ly/2lzCtuN</a>
An A-Z of Linux – 40 Essential Commands You Should Know	<a href="https://www.makeuseof.com/tag/an-a-z-of-linux-40-essential-commands-you-should-know/">https://www.makeuseof.com/tag/an-a-z-of-linux-40-essential-commands-you-should-know/</a>	<a href="https://bit.ly/2mJntun">https://bit.ly/2mJntun</a>
UNIX Tutorial	<a href="http://people.ischool.berkeley.edu/~kevin/unix-tutorial/toc.html">http://people.ischool.berkeley.edu/~kevin/unix-tutorial/toc.html</a>	<a href="http://bit.ly/22374hN">http://bit.ly/22374hN</a>
A Practical Guide to Ubuntu Linux: The Shell	<a href="http://www.informit.com/articles/article.aspx?p=2273593&amp;seqNum=5">http://www.informit.com/articles/article.aspx?p=2273593&amp;seqNum=5</a>	<a href="http://bit.ly/1ZwILUA">http://bit.ly/1ZwILUA</a>
Unix Tutorial	<a href="http://www2.ocean.washington.edu/unix.tutorial.html">http://www2.ocean.washington.edu/unix.tutorial.html</a>	<a href="http://bit.ly/1LUgiFM">http://bit.ly/1LUgiFM</a>
Learn Unix	<a href="http://www.tutorialspoint.com/unix/">http://www.tutorialspoint.com/unix/</a>	<a href="http://bit.ly/1YCh8ZN">http://bit.ly/1YCh8ZN</a>
<i>Part1</i> : Survival guide for Unix newbies	<a href="http://matt.might.net/articles/basic-unix/">http://matt.might.net/articles/basic-unix/</a>	<a href="http://bit.ly/2237l4k">http://bit.ly/2237l4k</a>
<i>Part2</i> : Settling into Unix	<a href="http://matt.might.net/articles/settling-into-unix/">http://matt.might.net/articles/settling-into-unix/</a>	<a href="http://bit.ly/1LeFHd6">http://bit.ly/1LeFHd6</a>
The Linux Command Line	<a href="http://linuxcommand.org/">http://linuxcommand.org/</a>	<a href="http://bit.ly/223JcdO">http://bit.ly/223JcdO</a>

# REFERENCES

---

Ritchie, D. M. 1984. "A Stream Input-Output System." *AT&T Bell Laboratories Technical Journal* 68 (8).  
<https://cseweb.ucsd.edu/classes/fa01/cse221/papers/ritchie-stream-io-belllabs84.pdf>.

"Standard Streams." 2019. *Wikipedia*. Wikimedia Foundation.  
[https://en.wikipedia.org/wiki/Standard\\_streams](https://en.wikipedia.org/wiki/Standard_streams).

"VT100." 2019. *Wikipedia*. Wikimedia Foundation.  
<https://en.wikipedia.org/wiki/VT100>.