Basic Unix - Part I

Jean-Yves Sgro

Updated: December 7, 2016

Contents

1 Introduction

This workshop is meant to learn and understand basic line commands as they are typed on a *text terminal* for a Unix-style operating system (Linux or Macintosh, or Windows with added software.)

This workshop will be loosely inspired by the *Software Carpentry* class **The Unix Shell**.[1]

We will learn "daily commands" *i.e.* commands that are useful for "every day life" on the computer such as creating files and directories, editing simple text files etc.

2 Set-up: Login the iMac

Using your `NetID` credentials login the iMac.

If this is the first time you use this computer some quick set-up will occur.

When the `Apple ID` screen appears choose to skip this section.

Shortly after you will be logged in "**as you**" on the computer.

2.1 Terminal

The workshop will be conducted on Apple *iMac* computers and since MacOS X happens to be a flavor of Unix we will first start with a terminal on a Macintosh.

If we need to move to a specific Linux system we'll make specific arrangements, or we can try using a web browser embedded Linux terminal as a free service, for example:

Choice	URL
Best	http://www.tutorialspoint.com/unix_terminal_online.php
Good	http://bellard.org/jslinux/

However, for now we'll stay on the Macintosh system.

2.2 Open a Terminal:

The `Terminal` software is located in `/Applications/Utilities` and you can navigate there in different ways.

You can also simply use "Spotlight Search" that looks like a magnifying glass : Click the magnifying glass at the top right corner of your screen and type `Terminal` then press the `return` key.

The default `Terminal` has a white background and black text.

[1] http://swcarpentry.github.io/shell-novice/

Note: You can make the text bigger by typing together `command +` and to reduce the font size type together `command -`.

3 Hard drive

The hard drive of a Unix system is organized in the same way as that of your laptop computer. The highest level of organization might be called `C:\` on a Windows system, on a Unix system it is called `root` and is usually written with the symbol `/` (forward slash.)

The file organizational structure is that of an "inverted tree" starting from `root` and branching out into directories and files.

Two important directories are `bin` which contains all the basic operating system software (or *binaries*, hence the name `bin`) and the directory called `Users` which contains the directory of all the users of the system.

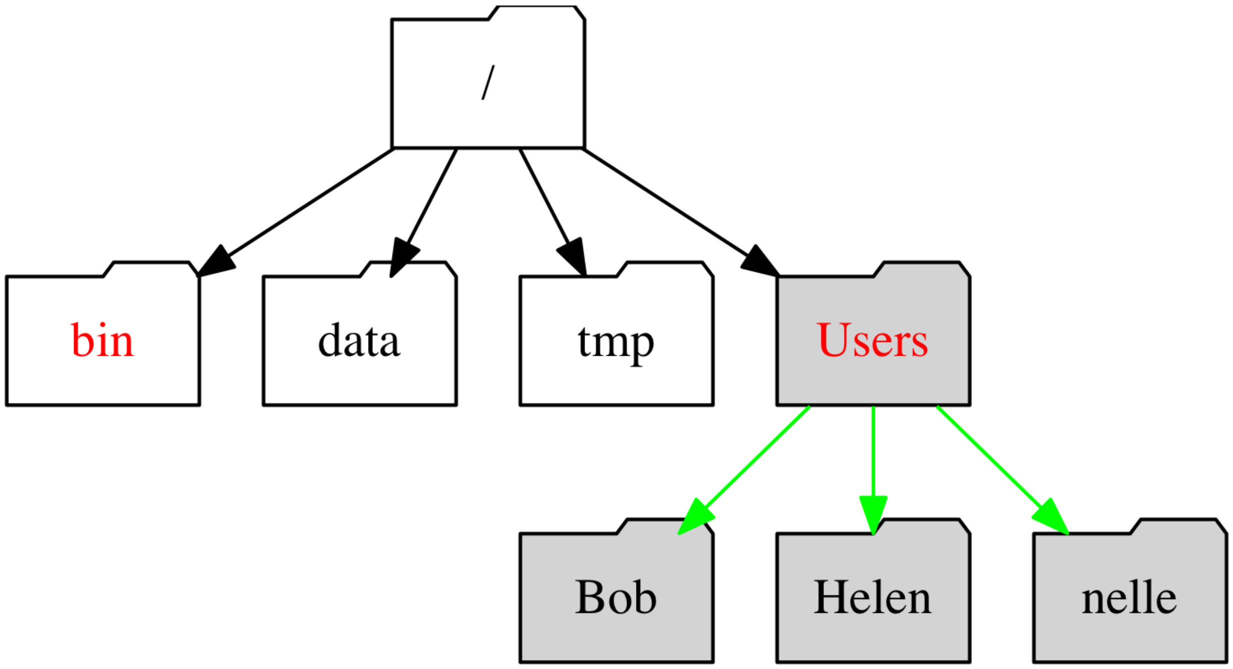

Figure 1: Hard drive organization.

Figure 1.

It should be noted that while `bin` is a standard name for most if not all systems, `Users` is a newer label and older system might have a different name or a different position within the tree. The common name of directories is available online[2].

On most modern systems the `Users` directory will contain the directory with *"your name on it"* containing your data and your files. We will explore this further below with the ***"home directory"*** . We will learn that we also use the *"forward slash"* or `/` to separate the name of successive directories within the hard drive "tree" of directories.

For user `nelle` that would be `/Users/nelle` where the leading slash represents `root`.

Note: Therefore there are two meanings for the `/` character. When it appears at the front of a file or directory name, it refers to the `root` directory. When it appears between names, it's just a separator.

[2]https://en.wikipedia.org/wiki/Filesystem_Hierarchy_Standard#Directory_structure

4 Command-line operation: The Shell

Before the development of graphical user interfaces (GUI) the command line interface (CLI) was the only way a user could interface with the computer.

A program called **the shell** takes commands typed on the keyboard and transfers them to the operating system (OS) for execution:

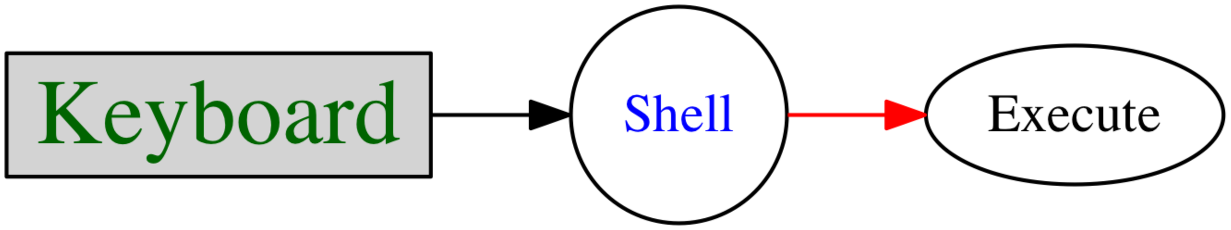

Figure 2: The shell transfers commands.

Figure 2

The **B**ourne **A**gain **SH**ell or `bash` shell is currently the default and most widely used shell, but other shell programs do exist and are available on a typical Unix-based system. These include: `ksh`, `tcsh` and `zsh`.

In this workshop we will only use `bash`.

5 The prompt: $

If you have not opened a Terminal yet please do so now using the information above.

Within the newly opened Terminal you can see some text, and the last character is likely a `$` which usually signifies that you are logged into a `bash` shell command line.

The `$` is called "the prompt" a word with many definitions but one of the definitions in the Merriam-Webster dictionnary summarizes it well:

"verb: to move to action : incite"

In other words it is an "invitation" from the computer for you to give it a command.

Since the Terminal invites us for a command, let's see if we can check the name of the shell we are running.

At the prompt type the following command with the word shell in uppercase preceded by an attached `$` sign which has a different function than the static `$` prompt. This will be reviwed in a later section.

```
echo $SHELL
```

```
/bin/bash
```

You can see that the shell is actually a software that resides within the `bin` directory.

6 Username: `whoami`

In the *Software Carpentry* class **The Unix Shell** we follow *Nelle Nemo* – (*Nemo* is Latin for "nobody" so she is probably not related to the famous *Captain Nemo*...).

Her username on the system is `nelle` and some commands refer to that.

Since you logged-in *as you* earlier on the iMac you probably know or easily guess your user name on this system. However, we'll learn our first command to verify that you are indeed logged-in *as you*!

The command is `whoami` and will echo on the screen your user name. Type the following command, press `return` and see who the `bash shell` thinks you are:

```
whoami
```

```
jsgro
```

The result is my username…

Of course for ***you*** the result will be different!

> ***Commands***:
> More specifically, when we type whoami the shell:
>
> 1. finds a program called `whoami`,
> 2. runs that program,
> 3. displays that program's output, then
> 4. displays a new prompt to tell us that it's ready for more commands.

7 Files and Directories

7.1 Home and working directories

7.1.1 Home directory

We already looked at the hard drive organization above and we could see that `nelle` (or you) will have your files stored in a directory named after your user name and contained within the `Users` directory. For Nelle Nemo that would be `/Users/nelle` which would represent her ***"home directory"***.

When you first login or open a new terminal you "land" inside the "home directory" wherever it may be located within the hard drive.

Note: There is a very convenient short-cut to signify the home directory which can be symbolically represented by the tilde symbol `~` usually the key below the `esc` ecape key on most English-based keyboards.

In the next paragraph we'll learn how we can **know** in which directory we have "landed" or, if we have changed directory, know in which directory we are currently.

7.1.2 Working directory:

For now we only know that we should be in the "home directory" but later we'll navigate in and out of directories that we'll create or copy. It's easy to get lost! Therefore knowing which directory you are currenly "looking into" is very useful.

The command for that is `pwd` or *print working directory*:

```
pwd
```

```
/Users/jsgro
```

This is my "home directory" and you will see your "home directory" when you run this command.

7.1.3 Changing directory

We don't have any other directory that we know of yet but we can learn this command already since ***"it will always bring you home"*** if you are lost.

To change directory we use the command `cd` which is made from the first two letters of the English phrase "**c**hange **d**irectory" as is the case for many other shell commands.

```
cd
```

We will use this command later to move in and out of directories.

7.2 Listing content directories: `ls`

Specific, empty standard directories are created wihtin the "home directory" when a new user is added on a Macintosh computer. These directories are empty. Since we landed in the "home directory" we can ask to see them with the command `ls` short for the word **l**i**s**t:

```
ls
```

You should see something like this on your terminal:

```
Last login: Tue Mar 22 08:50:34 on console
BIOCWK-00875M:~ jsgro$ ls
Desktop     Downloads   Movies    Pictures
Documents   Library     Music     Public
BIOCWK-00875M:~ jsgro$
```

`ls` prints the names of the files and directories contained within the current directory in alphabetical order, arranged neatly into columns.

If you have used this computer before there might be other files or directories.

We can update the file structure tree figure to reflect the content of your home directory, the top levels are simplified for clarity:

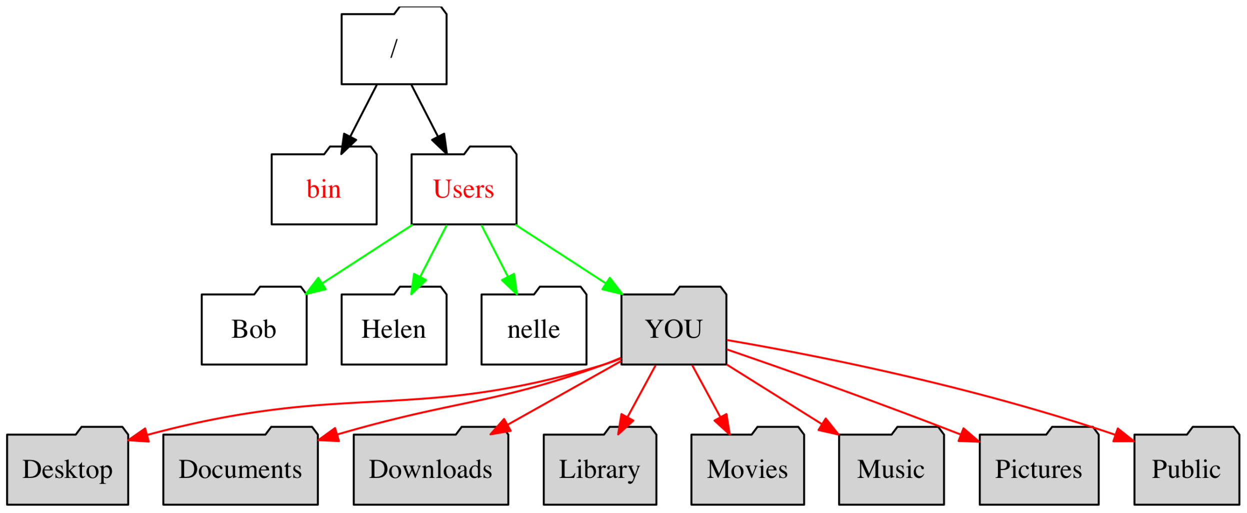

Figure 3: Hard drive organization for YOU.

Figure 3.

Of course, on the Macintosh itself you can use the graphial user interface (GUI) to look at the content of your directories. However, the purpose of this tutorial is to learn how to use the line command interface (CLI) which is useful when connecting to a remote computer that cannot be controlled with a GUI but only with the CLI.

Since these directories are new, for now they don't contain anything.

> Exercise: verify that the directories are empty.
> (Hint: `ls`.)

7.2.1 Command flags

We can modify the behavior of most shell commands by adding flags, for example the flag `-F` tells `ls` to add a trailing `/` to the name of directories (but not files.) Since we only have directories within the current folder all output will be flagged.

Note that there is a white space (of any length) between `ls` and `-F` to separate the two words. Without the space, the shell would think that we're trying to run a command called `ls-F`, which doesn't exist.

```
ls -F
```

```
Desktop/  Downloads/  Movies/ Pictures/
Documents/  Library/  Music/  Public/
```

7.2.2 All (hidden) files

You saw above that the directories are "empty" if you were a new user to this computer because they were freshly made. However, all directories contain within themselves at least two hidden files.

The flag to see all files is `-a`

```
ls -a Documents
```

```
.       ..      .localized
```

We can also use it together with the flag we already know.

Note that there is no space between them. (An alternate notation would be `ls -a -F` with a space between each of the `-` dashes.)

```
ls -aF Documents
```

```
./      ../     .localized
```

Therefore we discovered that there are three hidden files in the directory. If you want you can check that all other directories harbor the same files even though they are "empty."

The `.localized` files are empty files specific to the Macintosh. They are used when a user has changed the default langage within the "International" preferences to reflect their local langage. For example a computer set to French would change on the fly the names `Users` and `Library` to their French equivallent of `Utilisateurs` and`Bibliothèque`.[3]

However, the two most important items are `.` (dot) and `..` (dot dot) which are directories themselves as shown by the trailing `/`.

They are the symbolic representation of the "current directory" (dot) and the "parent directory" (dot dot).

Notation	Spoken Name	Definition
.	dot	**Current** directory

[3]See https://discussions.apple.com/thread/252040 (archived at: http://bit.ly/1RgVXcX)

Notation	Spoken Name	Definition
`..`	dot dot	**Parent** directory: directory "above" containing the "current" directory.

We will use this knowledge in the following section.

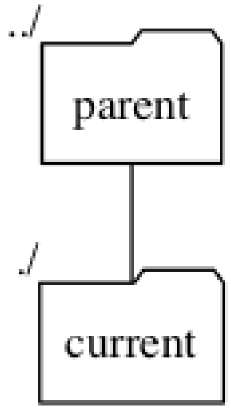

Figure 4: Current (dot) and parent (dot dot) directories.

Figure 4.

Note: Other common hidden files exist to customize the `bash` shell and would be located only within the home directory. They are typically called `.bash_profile`, `.bashrc` or `.bash_login`. The . prefix is used to prevent these configuration files from cluttering the terminal when a standard `ls` command is used.

7.3 Relative and absolute *path*

Referring to Figure 3 above we can see that starting from `root` the graphical "trail," "route," or "pathway" to go from `root` to arrive at any of the directories within your "home directory" (for example `Documents`) could be written as `/Users/YOU/Documents`. (Of course you would replace `YOU` with your `whoami` user name.)

The first forward slash `/` is the name of `root` and the subsequent `/` are the trailing separators as we saw above with the `ls -F` command. The separator is a *forward slahes* `/` and **not** a backslash `\` as is used in Windows.

The formal name for this written desctiption of pathway from `root` to final directory or file is ***path***.

The `path` can be **absolute** or **relative**.

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.

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

Since we already learned the command to change directory we can now *move into* the `Documents` directory for example and learn something about *path*:

```
cd Documents
```

We are now *within* the `Documents` directory. Therefore:

- the *current* directory is `Documents` also `./`
- the *parent* directory is the "home directory" also `../` and `~`

- the **relative** *path* to the `Downloads` directory is `../Downloads`
- the **absolute** *path* to the current directory is `/Users/YOU/Documents`

  Exercise: you can verify the validity of these statements with the commands:

```
ls -a
ls .
ls ./
ls ..
ls ../
ls ../Downloads
pwd
```

7.4 Privileges and permissions

On your own laptop or desktop you can make changes such as creating a directory, or moving a file from one place to another, usually by GUI. There are a few things that may require "*Admin*" or "*Administrator*" privileges for some operations such as installing software.

As a **user** you have the "*privilege*" to see, manipulate and change your own files and directories.

On shared computers it may be useful to prevent other users to see the files of a specific directory.

For this purpose directories and files can be tagged with the specific operation that a user can do, the "owner" of the file or another user.

We can "see" the privileges associated with files and directories simply by listing them with the `-l` flag to provide a **l**ong list with more details.

Remember that we already learned that the special symbol for the "home directory" is `~`. Therefore, whereever we are at the moment we can ask the listing of that directory:

```
ls -l ~
```

```
total 0
drwx------+  5 YOU  AD\Domain Users   170 Mar 15 18:36 Desktop
drwx------+  3 YOU  AD\Domain Users   102 Jun 19  2014 Documents
drwx------+  5 YOU  AD\Domain Users   170 Mar 15 18:36 Downloads
drwx------@ 43 YOU  AD\Domain Users  1462 Mar 15 18:36 Library
drwx------+  3 YOU  AD\Domain Users   102 Jun 19  2014 Movies
drwx------+  3 YOU  AD\Domain Users   102 Jun 19  2014 Music
drwx------+  3 YOU  AD\Domain Users   102 Jun 19  2014 Pictures
drwxr-xr-x+  5 YOU  AD\Domain Users   170 Jun 19  2014 Public
```

The privileges are described within the first ***column*** and will be explained below.

In order to organize privileges and permissions the Unix system is designed around the following definitions:

Table 3: Definitions of user groups. In the listing above they belong to user `YOU` and group `AD\Domain Users`.

User group	Definition
user	A user of the computer. Your specific user name is shown with `whoami`.
group	Multiple users can be assembled into a group *e.g.* from the same lab. The system administrator of the computer will create the group.
others	This is "anyone" else; on older system this was called "the world".
all	Contains everyone including **user**, **group** and **anyone** but not present in this listing.

Therefore a file "belongs" a user and a group.

The privileges and item nature are defined in the following table. The `rwx` privileges are shown in columns ordered by **user**, **group** and **others** from left to right. For each a set of `rwx` applies unless one of the privilege is not granted as expressed by `-`:

Table 4: Privilege tags

Privilege	Definition
d	This is a directory
r	The file can be **r**ead.
w	The file can be **w**riten or even over**w**riten.
x	The e**x**ecute privilege. For a directory it means its content can be listed.
-	The privilege within that column is not granted.

The `@` and `+` are rather new addition and part of the *Access Control List* (ACL) method added in the 80's. These are "extended" privileges that casual users should not interfere with and only useful to system administrators. For all info on this subject consult (Rubin 1989) (see online link in reference section.)

Therefore the listing:

```
drwxr-xr-x+  5 YOU  AD\Domain Users   170 Jun 19  2014 Public
```

can be read in plain English as: `Public` is a directory (`d`) that is owned by `YOU` and `AD\Domain Users` group. The owner has read, write and execute privileges (`rwx`). Privileges are only read and execute (`r-x`) for *group* and *other*. The directory information uses `170` bytes of hard drive space, and was created on June 19, 2014.

8 Advanced: Downloading files

In order to continue with the Software Carprentry files we need to download them.

While they have students download the files using the GUI and placing the result on the Desktop at the beginning of the tutorial, we'll accomplish that task by CLI to mimic a situation where we are connected to a remote computer without GUI:

If you are pressed by time and want to do the task by GUI here is the edited original information:

> Getting ready
> You need to download some files to follow this lesson:
>
> - Download *shell-novice-data.zip* and move the file to your Desktop.
> - The URL is: http://swcarpentry.github.io/shell-novice/data/shell-novice-data.zip
> - *Note*: Here is an alternate source in my DropBox: http://go.wisc.edu/38t26c
> - Unzip/extract the file (ask your instructor if you need help with this step). You should end up with a new folder called `data-shell` on your Desktop.

Now let's do this task by line command! We'll need two commands:

- one command to download the file: `curl`
- one command to unzip the file: `unzip`

> *Note*: On Linux the command `wget` could be used instead of `curl` but is not installed by default on the MacOS system.

First, let's go to the directory where we want to save the file: the `Desktop`.

```
cd  ~/Desktop
pwd
```

Now let's download the file. The program `curl` can be used to "transfer a URL". We'll need to specify the URL where to get the file, and specify the output name we want with the `-o` flag:

```
curl http://swcarpentry.github.io/shell-novice/data/shell-novice-data.zip -o shell-novice-data.zip
```

Now we can unzip the file:

```
unzip shell-novice-data.zip
```

This will unzip the content of the file and create a directory named `data-shell` located on the `Desktop` of your computer. You can probably also see it within the GUI on the Desktop of your computer.

We can now use commands we already learned to explore this directory.

We already know the flags `-F` and `-a`. Flag `C` forces the output into columns if that would not be the default. We also use the `~` shortcut for "home directory".

```
cd ~/Desktop/data-shell
ls -FaC
```

```
./          creatures/      notes.txt
../         data/           pizza.cfg
.DS_Store       molecules/      solar.pdf
.bash_profile       mycontent1.txt      sun_length.txt
Desktop/        mycontent2.txt      writing/
NewDir/         mydir/
NewDir2/        north-pacific-gyre/
```

We can explore the content of the directories but also change into any directory if we choose to with the `cd` command. We can go one level further down and list the content:

```
cd data
ls -F
```

```
amino-acids.txt     pdb/
animals.txt     planets.txt
elements/       salmon.txt
morse.txt       sunspot.txt
```

Here we can use the "dot dot" name for the "parent" directory to `cd` back "up" one level, and we don't have to know or type the actual name of the directory itself! (*Note*: the trailing `/` is not mandatory for the `cd` command.)

```
cd ../
ls -F
```

```
Desktop/        molecules/      pizza.cfg
creatures/      north-pacific-gyre/ solar.pdf
data/           notes.txt       writing/
```

9 Tab completion

So far we did not have to type much. However, somtimes file names can be long. We can use the `TAB` key to complete partially typed names of files or directories, and this will "traverse" the path and we can write a lot of text without typing.

For example, there is a file called `methane.pdb` within the `molecules` directory. We are going to list that file with minimal typing:

Let's try it:

Assuming we are within the `data-shell` directory (or provide the appropriate `cd` command!)

- type `ls`
- type `m` and the press the `TAB` key: this will complete the word `molecules` as there are no other words starting with the letter `m` so there is no ambiguity.
- Note that a trailing `/` was automatically added
- type `m` and press `TAB` this will complete the command as:

```
ls molecules/methane.pdb
```

What if there are ambiguities? There are two solutions: either type additional letters, or use the **double `TAB`** to find what the options are. For example there are two files starting with the letter `p` in the `molecules` directory. Pressing `TAB` twice will provide the list of available options. Lets try it:

- we are currently within the `data-shell` directory
- type `ls`
- type `m` and press `TAB` to complete `molecules`
- type `p` and press `TAB` - you will hear a sound and word is not completed after `p`
- type `TAB` again: a list of options is offered: `pentane.pdb  propane.pdb`
- you can now finish the command by typing one additional letter (either `e` or `r`) to finish the command.

Depending on your choice the final command will be either:

```
ls molecules/pentane.pdb
```

or

```
ls molecules/propane.pdb
```

10 Creating directories and files

So far we have only used existing files and directories.

10.1 Avoid blank space

> ***Important note***: white spaces or blanks should not be used for file or directory names on a Unix-style system. It is better to use dashes `-` or underscore `_` to separate words. However, if a file exists with white spaces, for example `File 1 of data` it is possible (but tedious) to use that name. There are two solutions: "escape" each white space with the backslash `\` or put the complete file name within quotes:
>
> - With backslash escape notation the file can be used as: `File\ 1\ of\ data`.
>
> - With quote notation the name can be used as: `"File 1 of data"`
>   White spaces are a common cause of error.

10.2 Make a directory: `mkdir`

On the GUI it is easy to create and move directories around. The same functionality exists with the CLI and it's easy provided we know where we are currenlty working (`pwd`).

Assuming we are within `data-shell` let's create a new directory called `mydir`. The command `mkdir` is used to **m**a**k**e a **dir**ectory:

```
cd ~/Desktop/data-shell
mkdir mydir
```

We now realize that's not the name we wanted! So, we can change it with the **mov**e `mv` command and give it a new name:

```
mv mydir mydata
```

This command would work in the same way on file names.

10.3 Text files: view and edit content

Shell commands exist to inspect the content of existing files and simple software exist to create new files.

10.3.1 Display file content

Different commands can be used to display all or portions of a text file.

For short files it is easy to type the complete file content onto the screen. For longer files we may want to see the beginning or the end of the file.

Make sure you are within `~/Desktop/data-shell`. We can display the content of the `notes.txt` file with the `cat` command:

```
cd ~/Desktop/data-shell
cat notes.txt
```

```
- finish experiments
- write thesis
- get post-doc position (pref. with Dr. Horrible)
```

For larger files we may be interested to see only the begining or the end of the file. Let's look for example at file `~/Desktop/data-shell/data/sunspot.txt`.

The commands `head` and `tail` respectively show the begining and end of the file, namely the first or last 10 lines. However, we can add a number flag to reduce this to the first 5 lines or last 3 lines as shown:

```
cd ~/Desktop/data-shell/data
head -5 sunspot.txt
```

```
(* Sunspot data collected by Robin McQuinn from *)
(* http://sidc.oma.be/html/sunspot.html         *)

(* Month: 1749 01 *) 58
(* Month: 1749 02 *) 63
```

```
cd ~/Desktop/data-shell/data
tail -3 sunspot.txt
```

```
(* Month: 2004 12 *) 18
(* Month: 2005 01 *) 31
(* Month: 2005 02 *) 29
```

From this we can conclude that observations listed for sun spots start in `1749` and end in `2005`.

There are two other, more sophisticated commands to explore longer files: `more` and `less` which are the older and newer versions of the same program respectively and allow to display only one screenful of text at a time. Let's try it.

```
less sunspot.txt
```

- Now press the `space bar` to go forward 1 full screen at a time.
- Press `return` to advance only one line at a time.
- `less` is newer and allows the use of the `up` and `down` arrows to explore content up and down the file.
- Press `q` to **quit**

Note: Files with very wide lines would "wrap" around and occupy more than one line on the screen terminal. The command `less -S` would prevent wrapping and the `left` and `right` arrows can be used to explore text side-ways on the screen.

10.3.2 Editing text file with `nano`

A full page text editor suitable for beginners called `nano` is now part of default available software and very useful to edit small text files.

Note: The old name of `nano` was `pico` so if you are working on a system that does not have `nano` try to use command `pico` instead. On a Mac both commands open the same `nano` software.

`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 ~/Desktop/data-shell/
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!          - - - -



^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 stay within the editing mode for further editing.

11 Getting help: manual pages

So far we have seen commands that are rather simple to remember as they are made with a few letters of an English description: `cd`, `mkdir`, `ls` etc.

The behavior of almost all commands can be altered with flags: `ls -F` and there are many others.

The ***manual pages*** contains a complete description of the commands as well as a list and description of all available flags. Sometimes examples are provided. The command function will be described on the first line.

The ***manual pages*** for a given command are called with `man` and the name of the command.

The information will be displayed with the `less` screen display and the information can be read following the `less` method:

- `space bar` to go down one screen
- `return` to go down one line
- `up` and `down` arrows to move up and down
- `q` to quit

  Exercise. You can try `man` on some of the commands we already know:

```
man man
man ls
man echo
man cd
man bash
man mkdir
```

Note: the ***manual pages*** are written in "geeky" language and may take some time to get used to. However, they provide a basis for immediate help or for quick understanding of the functionality of a command.

12 Concept: standard input and output

12.1 Standards

One of the most powerful Unix concept is the idea of **standard input** and **standard output** which are represented by your **keyboard** and your terminal **screen** respetively.

In fact **standard output** is internally split in "two channels" called **stdout** and **stderr** for the standard software output and standard error output in case errors are to be reported.

There are therefore three "streaming" channels for Input/Output (I/O)

Table 5: Understanding I/O streams numbers

Handle	Name	Description
0	**stdin**	Standard input
1	**stdout**	Standard output
2	**stderr**	Standard error

Figure 5

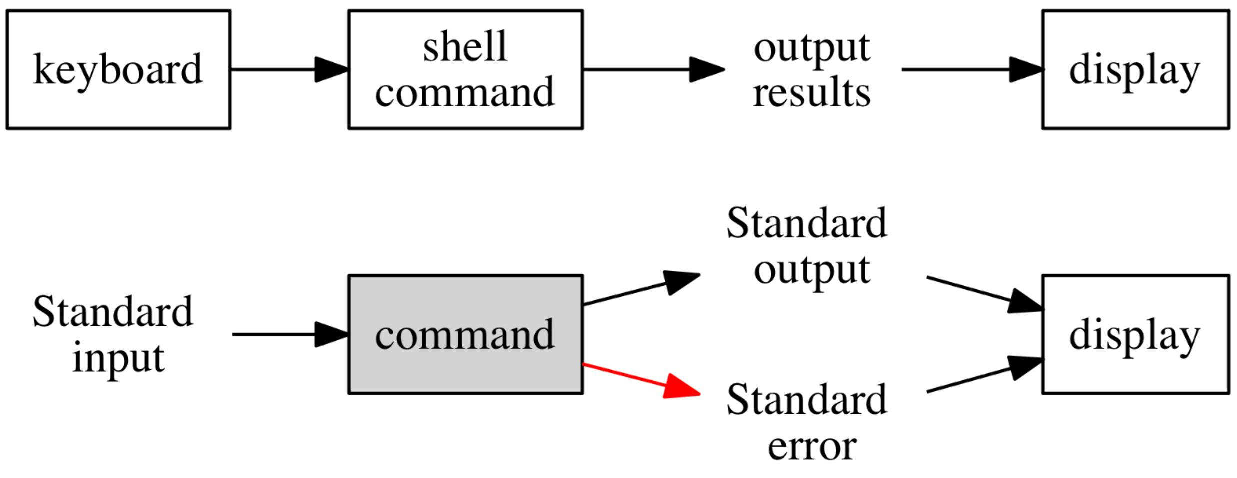

Figure 5: Standard Input/Output channels.

12.2 Redirect: capturing stdout

The "streaming" output can be "captured" so that instead of going to the screen display it will be "redirected" into a file or into another program. ***This is the basis for the power of Unix***.

12.2.1 Single redirect: >

When you type `ls` you see the results of listing the files onto your screen.

We can instead "redirect" this into a file as specified by the symbol `>` and the name of the file we want to create.

Let's try it! First let's do a "regular" `ls` and then let's redirect the content of the directory into a file called `mycontent1.txt` and a second time with `ls -C` into `mycontent2.txt`. We'll see why in a minute!

```
cd ~/Desktop/data-shell/
ls
ls > mycontent1.txt
ls -C > mycontent2.txt
```

The first `ls` will show the output on the screen and display the content in arranged columns, the default when output is to the screen.

Now we can see the content of the 2 files that we created by using the commands:

```
cat mycontent1.txt
cat mycontent2.txt
```

You will note that `mycontent1.txt` shows the results in a single column while `mycontent2.txt` shows the same column organization as the default screen display.

If you look within the `man` pages for `ls` you can see the following statements:

```
-1      (The numeric digit ``one''.)  Force output to be one entry per line.
             This is the default when output is not to a terminal.

-C       Force multi-column output; this is the default when output is to a terminal.
```

Therefore a "regular" `ls` is in fact equivallent to `ls -C` when used casually to display on screen display, but is equivalent to `ls -1` (number one) when redirecting to a file.

> Important note: `>` will redirect into a file. The file will be created if it does not exist (and if as a user you have the write privilege wihtin that directory). However, if the file already exists ***the file will be overwritten*** there will be no warning! And in addition there are ***no undo*** for this. Therefore, while this is very powerful it should be always used with care.

That said, the single redirect `>` can also be used to create simple text files "on the fly" without using any word processor. However, text can only be edited on the current line. The important thing to remember here is that it is necessary to tell the computer when you are done with the text input and want to return to the prompt: this is done by sending a `Control d` that signifies `EOF` or **end of file**.

> Exercise: Create a file called `mynotes.txt` containing about 2 lines as shown below:

```
cd ~/Desktop/data-shell/

cat > mynotes.txt
This is text that is going into the file.
It is a nice way to take notes on the fly.
The current directory content is:

Now hold the 2 keys together: CONTROL D
```

If you don't type `Control d` you will not get the `$` prompt back.

We will use this file in the next segment!

Note: I personally use this method often to leave "notes to myself" within directories where I work as a trail of information, commands that I used etc.

12.2.2 Double redirect: `>>` append to a file

The single redirect `>` will overwrite (clobber) the file if it already exists.

There are cases where we may want to **add** (append) more information at the **end** of the file and this can be accomplished with the help of the double redirect `>>`.

For example, let's complete the `mynotes.txt` file above by actually adding the content of the directory as stated in the file.

There are two ways we could accomplish that:

- we can do an actual `ls` or `ls -C` and double redirect the results `>>` into `mynotes.txt`
- Since we already have the directory content within files `mycontent1.txt` and `mycontent2.txt` we could redirect that instead.

The commands would be either:

```
cd ~/Desktop/data-shell/
ls -C >> mynotes.txt
```

or, with `cat` it would be:

```
cd ~/Desktop/data-shell/
cat mycontent2.txt >> mynotes.txt
```

We have now "appended" the content of file `mycontent2.txt` at the end of file `mynotes.txt`. You can verify that by simly typing the content of `mynotes.txt` to the screen!

```
cat  mynotes.txt
```

The 2 files have indeed been ***concatenated*** (glued) together and that is where the command `cat` gets its name: con***cat***enated!

If you look into the `man` pages for `cat` you will see that both functions that we just used are described in the first line of the description:

```
NAME
     cat -- concatenate and print files
```

12.3 Piping: sending output to other program

We just used the single `>` and double `>>` redirect method to "capture" standard output (`stdout`) and sending it to a file that we created or appended to an existing file.

But why stop there? Once the output stream has been redirected into a file it stops there. However, there is another method, piping with symbol `|` that allows the stream of output to be passed to another program. In fact multiple programs can be used in a "piping chain" for powerful data manipulation.

Let's see a simple example that will involve counting the number of lines, words and bytes within a file with the command `wc` (**w**ord **c**ount.)

First we can use file `~/Desktop/data-shell/data/animals.txt` which contains only 8 lines. You can verify that with `cat -n` that will number lines:

```
cat -n ~/Desktop/data-shell/data/animals.txt
```

```
     1  2012-11-05,deer
     2  2012-11-05,rabbit
     3  2012-11-05,raccoon
     4  2012-11-06,rabbit
     5  2012-11-06,deer
     6  2012-11-06,fox
     7  2012-11-07,rabbit
     8  2012-11-07,bear
```

Now let's use `wc` to see that it's true by "piping" the standard output provided by `cat` into the `wc` program:

```
cat ~/Desktop/data-shell/data/animals.txt  | wc
```

```
       8       8     136
```

The results shows that there are 8 lines, 8 words (there are no spaces on text within each line,) and 136 "bytes" which are all the visible letters with the addition of invisible (blanks) and non-printable characters such as `tab` or the end-of-line also called "newline."

Note that `wc -l` will only provide the very first number (instead of 3) representing only the number of lines:

```
cat ~/Desktop/data-shell/data/animals.txt  | wc -l
```

```
       8
```

The usefulness of this can be better understood with a larger file.

> Exercise: How many lines are in file `~/Desktop/data-shell/data/sunspot.txt` (Hint: `cat` and `wc -l`) ________________________________

12.4 Combining

Multiple pipes can be used and in addition the last output can also be redirected into a file! For example let's consider the following command:

```
cat ~/Desktop/data-shell/data/animals.txt  | wc -l > animals_length.txt
```

The final output is redirected into a new file that will contain the result of the command pipeline, in that case the number `8`.

Here is another example of 2 pipes and a redirect:

```
cat ~/Desktop/data-shell/data/sunspot.txt | head -50  | wc -l > sun_length.txt
```

Note: the use of `cat` is not mandatory in these previous examples and exercises, but it allows one more "layer" of process to understand the piping method.

13 Removing things: `rm` and `rmdir`

13.1 Removing a file: rm

We can remove the file just created above for example:

```
rm animals_length.txt
```

There is ***no warning and no undo***.

13.2 Removing directories

13.2.1 Empty directories

If a directory is empty (except for the standard `.` and `..`) it can be removed with the command `rmdir`. We can use that command to remove the directory `mydata` created above (it was `mydir` renamed as `mydata` by command `mv`.)

```
rmdir ~/Desktop/data-shell/mydata
```

There will be ***no warning. And there is no undo!***

13.2.2 Directories with content

If the directory contains other files or directories, which can be many levels deep, we need to use the `rm` command used for files but we need to make the command *recursive* which means to act on all sub-directories and sub-sub-directories etc.

The final command would be `rm -r` followed by the directory name.

However, if files are "locked" in some way but the user still have privileges to remove them the `-f` (force) flag may be extremely useful for a final command of `rm -rf` followed by the directory name. (Again: ***NO warnings and NO undo!***.)

14 Copying, moving and renaming: `mv`

We already encountered the `mv` command that we used to change the name of a directory. However, during the name change the location of the file can be changed as well.

Here are some example with `#` commented lines:

```
# Make 2 new directories:
mkdir ~/Desktop/data-shell/NewDir
mkdir ~/Desktop/data-shell/NewDir2
# Change into that directory:
cd ~/Desktop/data-shell/
# copy file mycontent1.txt into NewDir
cp mycontent1.txt  NewDir
# list content of NewDir
ls NewDir
# Copy file mycontent1.txt into NewDir2
# But change its name at the same time:
cp mycontent2.txt NewDir2/Copy-of-mycontent2.txt
# check content of NewDir2
ls NewDir2
```

15 Summary

15.1 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
Pipe	Take standard output and pass to next command as standard input

15.2 Symbols

Table 7: Symbols and filters

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

Table 8: 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

15.3 Commands learned:

Table 9: Commands in order of appearance in the text

Command	`man` page definition and/or example
`echo`	write arguments to the standard output. `echo $SHELL`
`whoami`	display effective user id.
`pwd`	return working directory name.
`cd`	change directory
`ls`	list directory contents. `ls -F`, `ls -FaC`
`curl`	transfer a URL.
`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`	concatenate and print files.
`head`	display first lines of a file.
`tail`	display the last part of a file.
`nano`	(Text editor) Nano's ANOther editor, an enhanced free Pico clone.
`wc`	word, line, character, and byte count.
`rm`	remove directory entries *i.e.* remove files. Remove non-empty dir with `rm -r`
`rmdir`	remove directories (empty dirs)
`cp`	copy files.

16 Resources

16.1 Online tutorials

There are many many online tutorials about Unix and any Google search will yield plenty of results. Here is a table with a few tutorials that seem to be reasonably well prepared with a target audience of beginner. The "Archived" column refers to the URL saved at archive.org if it exists.

Table 10: Online resources

Name.of.Tutorial	URL	Archived
UNIX Tutorial for Beginners	http://www.ee.surrey.ac.uk/Teaching/Unix/	http://bit.ly/1pixR8C
UNIX Tutorial	http://people.ischool.berkeley.edu/~kevin/unix-tutorial/toc.html	http://bit.ly/22374hN
A Practical Guide to Ubuntu Linux: The Shell	http://www.informit.com/articles/article.aspx?p=2273593&seqNum=5	http://bit.ly/1ZwILUA

Name.of.Tutorial	URL	Archived
Unix Tutorial	http://www2.ocean.washington.edu/unix.tutorial.html	http://bit.ly/1LUgiFM
Learn Unix	http://www.tutorialspoint.com/unix/	http://bit.ly/1YCh8ZN
Part1: Survival guide for Unix newbies	http://matt.might.net/articles/basic-unix/	http://bit.ly/2237l4k
Part2: Settling into Unix	http://matt.might.net/articles/settling-into-unix/	http://bit.ly/1LeFHd6
The Linux Command Line	http://linuxcommand.org/	http://bit.ly/223JcdO

16.2 Online courses

16.2.1 Students only (STS)

UW Students have access to free classes from the DoIT and may be able to access "Linux in a Day". The complete class offerings is at sts.doit.wisc.edu/classlist.aspx (NetID login required.)

16.2.2 All UW

All UW personel on the other hand can access the video courses from Lynda.com. See it.wisc.edu/services/online-training-lynda-com/

Searching for the word "*Linux*" currently yields 6 beginner level courses for various flavors of Linux.

A search for the word "*Unix*" only has one entry "Unix for Mac OS X Users" a beginner level course of 6h35min.

Another relevant search is "*bash*" which also yields a single beginner level course of 1h35min "Up and Running with Bash Scripting."

16.3 Software Carpentry

Software Carpentry is a volunteer organization whose goal is to make scientists more productive, and their work more reliable, by teaching them basic computing skills.

They provide two Unix tutorials versions, one in video format and one in text format. The video format is useful for pre-class learning.

Title	URL	Archived
The Unix Shell (Text)	http://swcarpentry.github.io/shell-novice/	http://bit.ly/1piCBel
The Unix Shell (video)	http://swcarpentry.github.io/v4/shell/index.html	http://bit.ly/1UVSQK6

17 R Session

This document was created with `R` and `RStudio` software with the followingconfiguration:

```r
sessionInfo()
```

```
R version 3.2.4 (2016-03-10)
Platform: x86_64-apple-darwin13.4.0 (64-bit)
Running under: OS X 10.11.6 (El Capitan)

locale:
[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
[1] pander_0.6.0     DiagrammeR_0.8.2 knitr_1.15.1

loaded via a namespace (and not attached):
 [1] Rcpp_0.12.3      codetools_0.2-14 visNetwork_0.2.1 digest_0.6.9
 [5] rprojroot_0.1-1  plyr_1.8.3       jsonlite_0.9.19  magrittr_1.5
 [9] evaluate_0.10    scales_0.4.0     stringi_1.0-1    rstudioapi_0.5
[13] rmarkdown_1.2    tools_3.2.4      stringr_1.0.0    htmlwidgets_0.6
[17] munsell_0.4.3    yaml_2.1.13      colorspace_1.2-6 htmltools_0.3.5
```

References

Rubin, Craig. 1989. *Rationale for Selecting Access Control List Features for the Unix System.* 2nd ed. FORT GEORGE G. MEADE, MARYLAND 20755-6000: NATIONAL COMPUTER SECURITY CENTER. NCSC-TG-020-A. DIANE Publishing. http://fas.org./irp/nsa/rainbow/tg020-a.htm.