# Command-Line-Intro-MSB

March 8, 2019

## 0.1 # The Command Line - An Introduction

Graphical user interfaces are fast, often more than fast enough to suit our needs. GUIs are feature rich, can be intuitive, and often filter out a lot of stuff we don't need to know about and aren't interested in. Nearly everything we need to do can be done simply and quickly using a GUI.

## 0.2 But some things benefit from batch processing..

For example, cleaning all the PDFs off of a cluttered desktop. Backing up files and data. Getting file and permissions info. Anything we do more than a couple of times on a regular basis, and especially error prone, repetitive tasks.

**The command line is a great resource for speeding up and automating routine activities without using a lot of processing power. In some cases, it can be better for**

- Searching for files
- Searching *within* files
- Reading and writing files and data
- Network activities

Some file and data recovery processes can **only** be executed from the command line.

## 0.3 Overview

This morning we will focus on using a command line client to navigate the filesystem, read and write files, and practice stringing together commands using pipes. For various good reasons, the emphasis will be on UNIX shell commands which are used on Linux and Mac OS operating systems. Most if not all of the commands covered will not work within the standard Windows command line client. Windows users can and are encouraged to avail themselves of UNIX command line clients:

- Git provides an excellent client: `https://git-scm.com/`
- *Minimalist GNU for Windows (mingw)* underlies the Git client: `http://www.mingw.org/`
- Cygwin is another popular option: `https://www.cygwin.com/`
- For Windows 10 Anniversary Edition and later try out the Windows Subsystem for Linux

**[note]** in all of the executable code blocks below the `%%bash` line at the beginning of the block tells the notebook we're using to execute the commands in the block as if they were typed in at the `bash` command prompt. *This is not required when you are typing in commands from the `bash` command prompt.*

### 0.3.1 Live Coding

Today we are going to be executing commands together and therefore need to have a **command line** environment on our systems. If you are already running the Mac OS or Linux you already have one - the **terminal**. If you are running Windows you need to install a UNIX command environment such as those listed above.

**Check for operating command line environment**

### 0.3.2 Getting Help and More Information . . .

Many shell commands have flags or options that can be utilized to refine the execution of the command. In order to find out more about a specific command and its options, three resources (one or more of which may be available for most commands) are the `--help` flag, the manual pages available through the `man` command, and command information available through the `info` command.

```
ls --help
```

OR

```
man ls
```

OR

```
info ls
```

**Give it a try**

1. Follow along with me executing the above commands
2. Look up the help information for the `pwd` and `cd`

### 0.3.3 Navigation

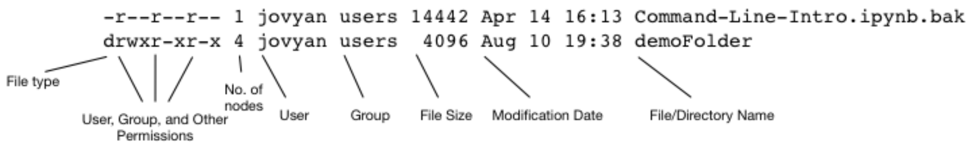
**Where am I and what's here?**

```
pwd
```

`pwd` is short for 'present working directory.' The output of this command is the absolute path to the directory a user is currently in. Often, knowing the absolute path is necessary in order to move or copy files, or to run scripts.

```
ls
```

`ls` lists the contents of a directory or directory tree. It is commonly executed with the '-l' (for long output) and/or the '-a' (for a listing of all files - including hidden ones) flag. If `ls` is executed without providing a specific location (path) it will list the contents of the present working directory. If a path is provided it will provide the list of files from that location.



```
-r--r--r-- 1 jovyan users 14442 Apr 14 16:13 Command-Line-Intro.ipynb.bak
drwxr-xr-x 4 jovyan users  4096 Aug 10 19:38 demoFolder
```

elements of the `ls -l` command output

**Give it a try**

- Use the `pwd` command to see the path of the *present working directory*
- Use the `ls` command to see the contents of the *current directory*
- Use the `ls ..` command to see the content of the *parent* of the current directory

The `..` character sequence is a special reference to the parent of the current directory. This character combination can be used multiple times to represent higher levels relative to the current directory. For example `../..` refers to the parent of the current directory's parent (grandparent), and `../../..` refers to the great grandparent of the current directory. This shorthand reference can be used in many other commands where you would provide a *path* for a command to operate on.

- Use the `ls -l` command to see the detailed listing of the contents of the current directory

The `-l` flag used above stands for "long list" when used with `ls`. Compared with the list of filenames produced by the unflagged `ls` command, the long list provides some useful information, including whether a list item is a file or directory, file/directory permissions, the owning user, the owning group, the size and the time when the file was last modified.

- Use the `ls -la` command to see the detailed listing of the contents of the current directory - including hidden and special files and pointers

The `-a` option, or just `a` when combined with another option such as in this example tells the `ls` command to display all files, including hidden files and the special reference shortcuts `..` (for the parent directory of the current directory), and `.` for the current directory. Any file or directory name that begins with a `.` (such as a file named `.ipynb_checkpoints`) is treated as a hidden file and won't be displayed in an ls command unless the -a flag is provided.

Notice the `.`, `..` items that are now in the file listing. These items are not displayed by default, as they are *system* provided representations of the current directory location `.`, or the parent directory location `..`. Any file or directory name that begins with a `.` (such as the `.ipynb_checkpoints` file) is treated as a hidden file and won't be displayed in an `ls` command unless the `-a` flag is provided.

**How do I get out?**

There are some shortcuts to reference specific locations in the computer's file system:
`/` is a shortcut for the top directory in the file system
~ (tilde) is a shortcut for the current user's home directory
`.` is a shortcut for the present working directory
`..` is a shortcut for the parent of the present working directory
`-` is a shortcut for the previous directory

`cd`

`cd` is one command that will also work in the Windows shell, and stands for… *change directory*.

The fastest way to move up within a folder hierarchy is to use the *dot-dot* notation for the parent directory:

**Give it a try**

- Use the `cd ..` command to move up one directory
- Use the `cd ../..` comand to move up two directories
- Use the `cd /` command take you to the topmost directory
- Use the `cd ~` (or just plain `cd`) command to take you to the current user's home directory

**Quick Check:**
**How far up can we go from here?**

```
pwd
cd ..
pwd
```

**Higher?**

```
pwd
cd ../..
pwd
```

**Getting moving**

```
# Navigate to the root directory ('/'), generate a list of its contents, and then navigate bac
pwd

# Set pwd to a variable for later use
DEMO=$(pwd)
echo $DEMO

cd /
pwd

ls -la

cd $DEMO
pwd
```

Three good things to know:

- `clear`: Clear the screen.
- *tab completion:* Auto-complete file and directory names.
- *up- and down-arrow:* Scroll through previous commands.

**Relative versus Absolute Paths** At first, navigating across directories within the shell may seem slower and more cumbersome than simply using mouse clicks and a GUI. Using tab completion with absolute OR relative paths is an excellent way to increase efficiency.

The *relative path* is the path to a directory or file from the context of a specific starting point (usually the present working directory).

The *absolute path* is the path to a directory or file from the filesystem root ('/').

As an example, to navigate to the root directory from our present working directory, we can use

*cd ../../../* (using the relative path)

**OR**

*cd /* (using the absolute path)

The second option is plainly faster in this case. In other contexts, the relative path would be faster.

Relative and absolute paths can also be used to list the contents and read, write, or delete files in other directories.

**Give it a try**

```
# what is my current directory?
pwd

# save the current directory for use later
DEMO=$(pwd)

# Move up two directories in the file structure using the relative path.
cd ../..
pwd

# Return to the saved directory and then navigate to the root directory using the absolute pat
cd $DEMO
pwd
cd /
pwd

# Go home
cd
pwd

# Go to a specific directory with an absolute path
cd /Users/kbene
pwd
```

## 0.4 What Next

Some additional learning materials

- Lynda.com *Learn the Linux Command Line: The Basics* - particularly:
  - Introduction

    - 1. Command-Line Basics
    - 2. Files, Folders, and Permissions

- Webmonkey "Unix Guide"
- Linux Command Line Cheatsheet
- Bash Guide for Beginners
- An A-Z Index of the Bash command line for Linux

Some additional commands of immediate interest:

- find
- grep
- curl