

Using a Command Line Interface A CoS126 Introduction

Long ago, before operating systems with graphical user interfaces (GUIs), everyone used command line interfaces (CLIs). For example, instead of starting a program by clicking the icon for your program in the Launchpad or whatever new-fangled mawhoozawhatzit, you instead typed the name of your program. And instead of clicking on menus in your program to adjust settings, open files, etc, you could specify these things as command line arguments.

In fact, expert computer users still use command prompts quite heavily. It's an incredibly useful abstraction that you're now going to begin making use of.

A specific instance of the command line interface is called a "terminal window" by Mac OS X (which utilizes a version of Unix under the hood), and the term "command prompt" is used by Windows. The terms are virtually interchangeable. We will use the term "command line interface" throughout this document.

Learning to fully utilize a command line interface is something that could (and does) fill an entire book. In this tutorial, we'll address the key ideas, give a few examples, and leave you to learn the rest on your own as you choose.

The Basics - Opening the Command Line Interface

In Mac OS X, the terminal application can be found in Applications->Utilities->Terminal.

In Windows, it can be found in Programs->Accessories->Command Prompt.

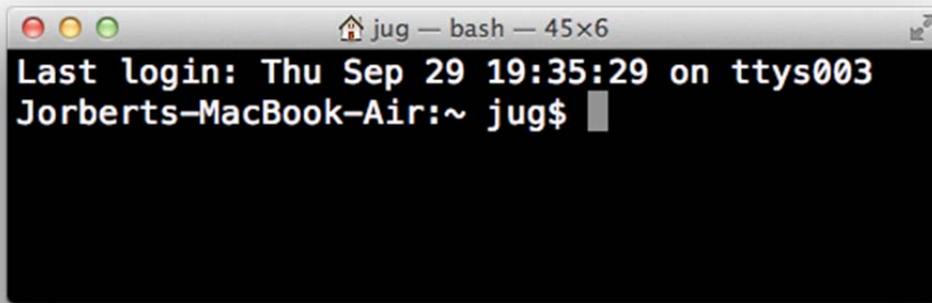
Note, there are many other ways of accessing the command line interfaces. All of them are fine to use.

Prompt text

When you open your CLI, the first thing you'll see is a prompt, i.e. a string of text that lets you know the computer is waiting for you, the user, to take action.

In Windows, the default prompt is the entire path to your *present working directory* (we'll discuss what that means later) followed by a > sign. For example:

In Mac OS X, the default command prompt is your computer name, followed by the name of your present working directory, followed by your user name, followed by a \$. For example:

A screenshot of a macOS terminal window. The title bar at the top shows a home icon, the text 'jug — bash — 45x6', and a close button icon. The terminal content is white text on a black background. The first line reads 'Last login: Thu Sep 29 19:35:29 on ttys003'. The second line reads 'Jorberts-MacBook-Air:~ jug\$' followed by a small grey cursor block.

Here, my computer is named Jorberts-Macbook-Air;, I am in the “~” directory [more on this later], my user name is jug, and there’s that dollar sign.

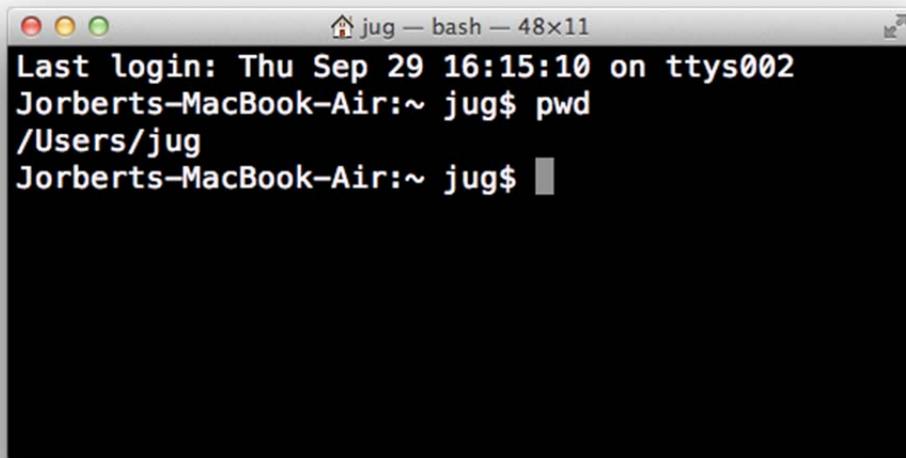
Both of these prompts can be changed. I leave it to you to google this.

Present Working Directory

Your CLI can be thought of as being “in” a particular directory. We call this directory the *present working directory*.

In Windows, you always know your present working directory, because the default prompt tells you the entire path to the present working directory. In the example below, we see that the command prompt starts off in a directory called “Hvagvarfis”, and the whole path is “E:\users64\Hvagvarfis”.

In Mac OS X, you must run a program called *pwd* in order to see your present working directory. To do this, type “pwd”. For example, on my computer, if I use *pwd* right after opening a terminal window, I get:

A terminal window titled 'jug - bash - 48x11' with a home icon. The text inside the terminal reads: 'Last login: Thu Sep 29 16:15:10 on ttys002', 'Jorberts-MacBook-Air:~ jug\$ pwd', '/Users/jug', and 'Jorberts-MacBook-Air:~ jug\$' followed by a cursor bar.

```
jug — bash — 48x11
Last login: Thu Sep 29 16:15:10 on ttys002
Jorberts-MacBook-Air:~ jug$ pwd
/Users/jug
Jorberts-MacBook-Air:~ jug$ █
```

This means that the name of the present working directory is *jug*, and the full path of the present working directory is */Users/jug*. Note that in Unix based operating systems, the “~” directory is shorthand for your home directory.

One important distinction is that in Windows, all paths start with a drive letter and then a colon symbol. In Unix based operating systems, everything simply starts with */*.

Listing the Files in a Directory

To list files in the present working directory in Windows, use the *dir* command:

In Mac OS X, use the *ls* (that’s lowercase LS) command:

```
Jorberts-MacBook-Air:~ jug$ pwd
/Users/jug
Jorberts-MacBook-Air:~ jug$ ls
Desktop          Pictures
Documents        Public
Downloads        Scuts
Library          introcs
Movies           work
Music
Jorberts-MacBook-Air:~ jug$
```

Here we see that there are 11 things inside of my /users/Jug directory. In Mac OS X, subdirectories are listed in exactly the same format as files. You can tell Mac OS X to mark folders with a / by typing “ls -F”.

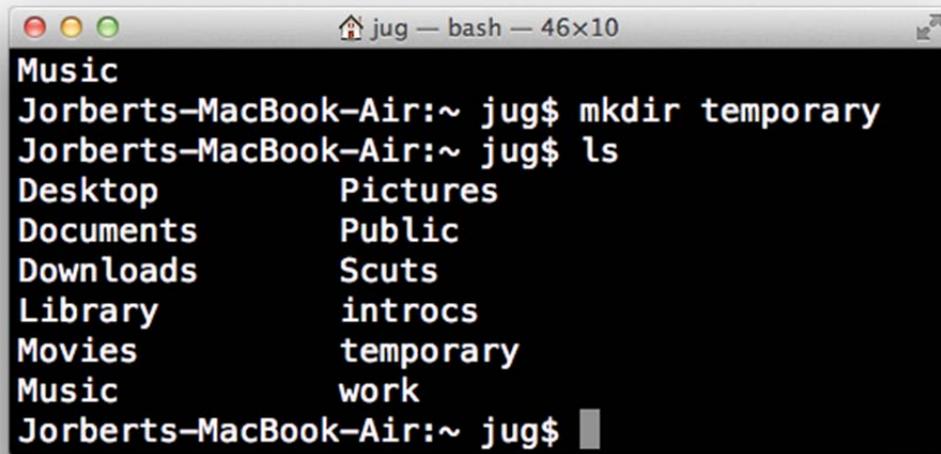
```
Jorberts-MacBook-Air:~ jug$ ls -F
Desktop/          Pictures/
Documents/        Public/
Downloads/        Scuts/
Library/          introcs/
Movies/           temporary/
Music/            work/
Jorberts-MacBook-Air:~ jug$
```

Creating, Removing, and Navigating between Directories

In both Mac OS X and Windows, you can create a new directory by using the *mkdir* command.

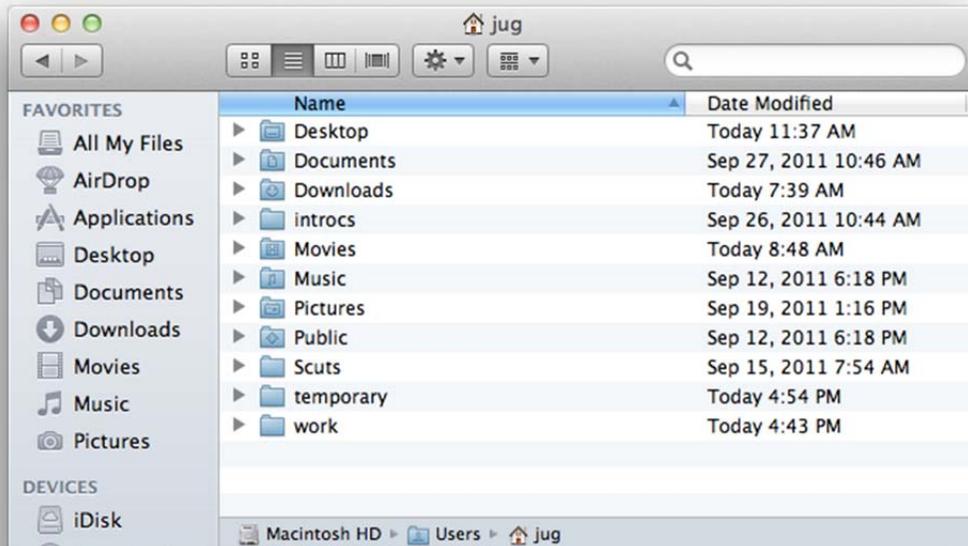
In this section, we recommend that you follow along.

First, type “mkdir temporary” and press enter. Now type “ls” or “dir” to get the directory listing. You’ll see that a new folder named “temporary” exists.



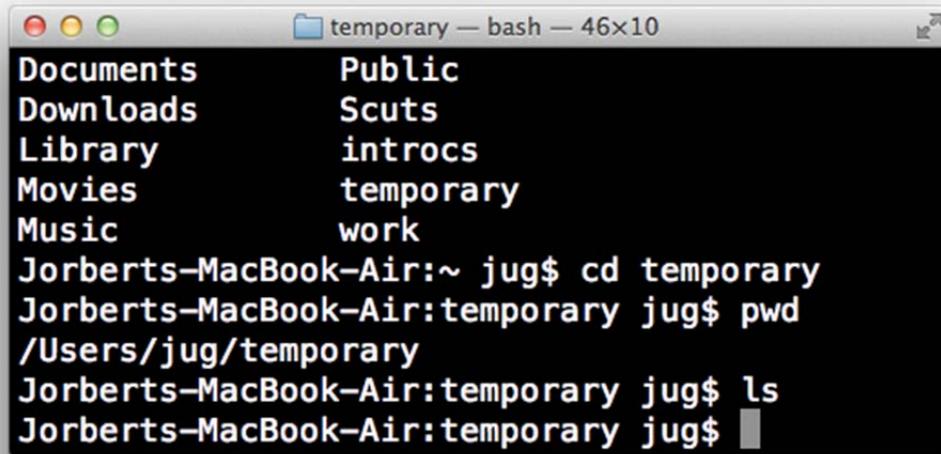
```
jug — bash — 46x10
Music
Jorberts-MacBook-Air:~ jug$ mkdir temporary
Jorberts-MacBook-Air:~ jug$ ls
Desktop          Pictures
Documents        Public
Downloads        Scuts
Library          intros
Movies           temporary
Music            work
Jorberts-MacBook-Air:~ jug$
```

In fact, you can even go to your Finder or Explorer Window, and you’ll see that the GUI can see the folder you’ve made. For example, on a Mac, we have:



To change directories in Mac OS X or QWindows, one uses the command *cd*. Try it out by typing “cd temporary”.

In Mac OS X, type *pwd* and *ls*, and you will see the following:

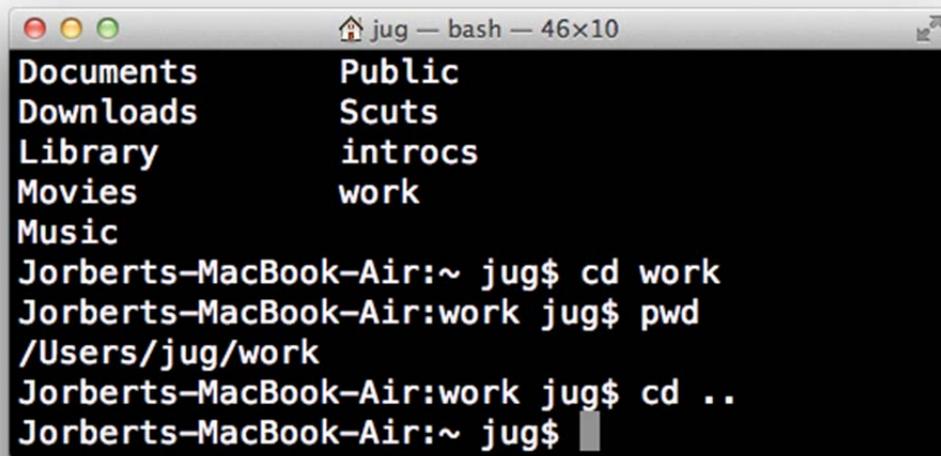
A terminal window titled "temporary — bash — 46x10" with a dark background and white text. It shows a directory listing of the home directory with two columns: Documents, Downloads, Library, Movies, Music, Public, Scuts, introcs, temporary, and work. Below the listing, the user runs the command 'cd temporary', then 'pwd' which outputs '/Users/jug/temporary', and finally 'ls' which shows no output.

```
Documents      Public
Downloads      Scuts
Library        introcs
Movies         temporary
Music          work
Jorberts-MacBook-Air:~ jug$ cd temporary
Jorberts-MacBook-Air:temporary jug$ pwd
/Users/jug/temporary
Jorberts-MacBook-Air:temporary jug$ ls
Jorberts-MacBook-Air:temporary jug$
```

There are no files, unsurprisingly. We also note that the “~” has changed to “temporary”, which means that our current directory has changed from our home directory to a directory with the name “temporary”.

In Windows, the command prompt automatically shows you what folder you’re in. Type “dir” and you will also see that there are no files in the temporary directory:

How does one go back, you may ask? Simply type “cd ..” to go back one directory. For example:

A terminal window titled 'jug — bash — 46x10' showing a directory listing and navigation commands. The listing shows 'Documents', 'Downloads', 'Library', 'Movies', 'Music', 'Public', 'Scuts', 'intros', and 'work'. The user then runs 'cd work', 'pwd' (outputting '/Users/jug/work'), and 'cd ..' to return to the home directory.

```
Documents      Public
Downloads      Scuts
Library        intros
Movies         work
Music
Jorberts-MacBook-Air:~ jug$ cd work
Jorberts-MacBook-Air:work jug$ pwd
/Users/jug/work
Jorberts-MacBook-Air:work jug$ cd ..
Jorberts-MacBook-Air:~ jug$
```

Running a Program

To run a program, simply type the name of the program. Programs in Windows typically (but don't always) have the extension `.exe` (for executable). In Mac OS X, they typically have no extension at all.

For this to work, the program must be either in the present working directory, or on the *system path* (see appendix).

Running Programs Using the Full Path

If you want to run something that's not on the system path, and not in the present working directory, you can instead type the full path of the file.

For example, if we type `/usr/bin/ls` at the command prompt, we get an error, because the computer tries to find `ls` in the wrong location:

```
jug — bash — 48x7
Last login: Thu Sep 29 17:20:40 on ttys006
Jorberts-MacBook-Air:~ jug$ /usr/bin/ls
-bash: /usr/bin/ls: No such file or directory
Jorberts-MacBook-Air:~ jug$
```

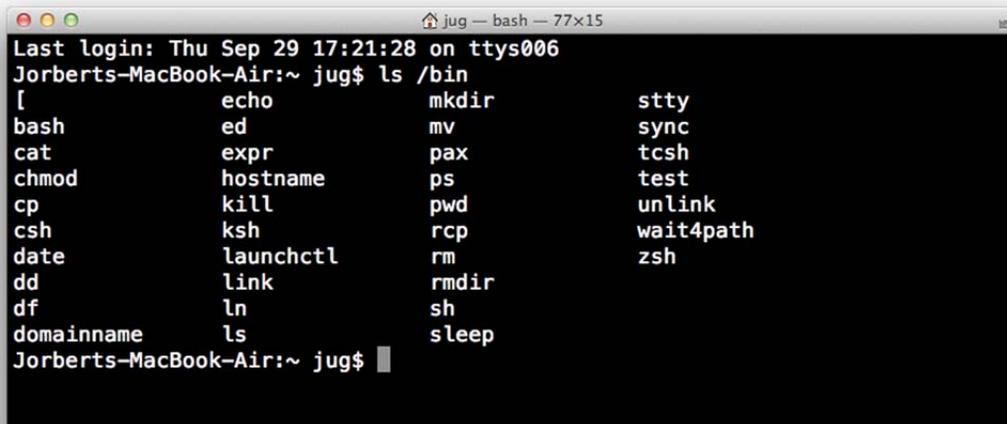
/bin/ls works just fine, by contrast

```
jug — bash — 48x7
-bash: /usr/bin/ls: No such file or directory
Jorberts-MacBook-Air:~ jug$ /bin/ls
Desktop          Movies           Scuts
Documents        Music           introcs
Downloads        Pictures        temporary
Library          Public          work
Jorberts-MacBook-Air:~ jug$
```

Command Line Arguments

You can provide additional information to a program about what you'd like to do with command line arguments. For example, if you give the *ls* or *dir* programs a directory as a command-line argument it will instead list the contents of the directory name that you give.

For example, try `ls /bin`, and you should get something like:

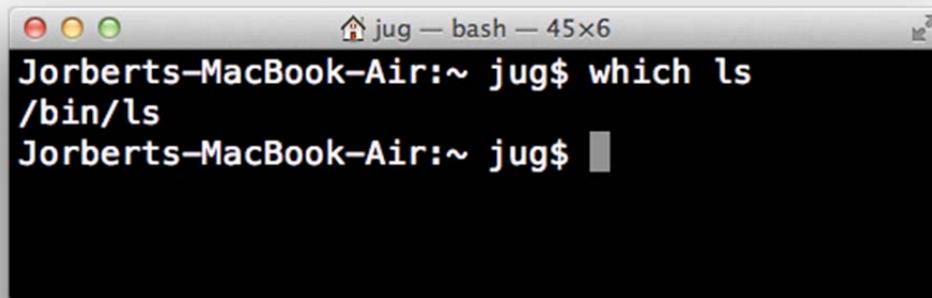


```
jug — bash — 77x15
Last login: Thu Sep 29 17:21:28 on ttys006
Jorberts-MacBook-Air:~ jug$ ls /bin
[
bash          echo          mkdir         stty
cat           ed            mv            sync
chmod         expr          pax           tcsh
cp            hostname     ps            test
csh           kill          pwd           unlink
date          ksh           rcp           wait4path
dd            launchctl    rm            zsh
df            link          rmdir
domainname   ln            sh
ls           sleep
```

Or in Windows, try “`dir C:\`” and you should get something like:

Finding Where a Program is Located

In Mac OS X, you can type *which* “[name of program]”, and it will tell you where that program is located. On Windows, use *where* instead. For example:



```
jug — bash — 45x6
Jorberts-MacBook-Air:~ jug$ which ls
/bin/ls
Jorberts-MacBook-Air:~ jug$
```

Redirecting StdOut

Ordinarily, programs print their text output to the screen. However, you can redirect the output of most programs to a file. To do this, one simply types:

Program name [command line arguments] > [filename]

The > indicates to the operating system that output should be sent to a file instead of the screen.

For example, try “ls > curdir.txt” or “dir > curdir.txt” depending on your operating system.

This will output the results of ls to the file *curdir.txt*.

Redirecting StdIn

Ordinarily, when a program requests input from StdIn, this information is gathered from the keyboard. However, one can redirect StdIn to refer to a file. To do this, you type:

Program name [command line arguments] < [filename]

Now, all calls to StdIn will be referred to the file given by *filename*.

And there's more... much more...

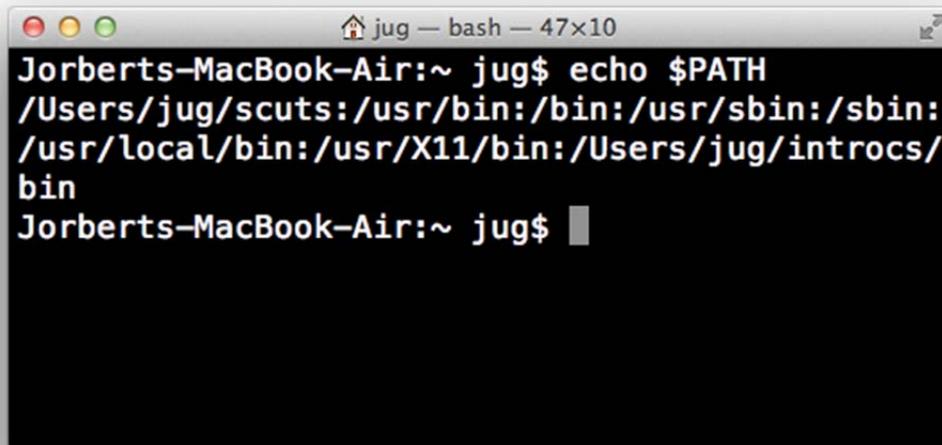
We have scratched the surface of what it's possible to do using a command line interface. There are many tutorials on the web that you might find interesting to explore.

Appendix: System Path

The system path is a special list of directories that the computer always checks when you ask it to run a program.

To see the system path in Windows, type:
echo %PATH%

In Mac OS X, type *echo \$PATH*

A terminal window titled "jug — bash — 47x10" on a Mac. The prompt is "Jorberts-MacBook-Air:~ jug\$". The command "echo \$PATH" has been entered and executed. The output is: "/Users/jug/scuts:/usr/bin:/bin:/usr/sbin:/sbin:/usr/local/bin:/usr/X11/bin:/Users/jug/introcs/bin". The prompt "Jorberts-MacBook-Air:~ jug\$" is shown again with a cursor.

```
Jorberts-MacBook-Air:~ jug$ echo $PATH
/Users/jug/scuts:/usr/bin:/bin:/usr/sbin:/sbin:
/usr/local/bin:/usr/X11/bin:/Users/jug/introcs/
bin
Jorberts-MacBook-Air:~ jug$ █
```

Here we see that the path on my computer includes the directories /Users/jug/scuts, /usr/bin, /bin, /usr/sbin, etc.

As an example of using the path, "ls" is actually a program, and it is located at /bin/ls. If we did not have /bin/ls in our path, then typing "ls" would result in an error.