Tuesday, Thursday
9:05 (ILR 315)
11:15 (Call Aud)

CS 1110: Introduction to Computing Using Python
Fall 2018

Main

About:
Announcements
Course Staff
Times & Places
Calendar

Materials:
Lectures
Texts
Python
Text Shell

Assessment:
Grading
Assignments
Labs
Exams

Resources:
CMS
Piazza
AEWs
FAQ
Python API
Introcs API
Python Tutor

Style Guide
Terminology

Academic Integrity

Using Text-Based Commands

Python first became popular as a *scripting language*, which is used to automate repetitive tasks on a computer. As a scripting language, it is heavily geared toward being used in text-based *command shell*. In order to learn how to get the most of out Python, you need to learn how to use your command shell.

Command shells are operating system dependent. They are called different things depending on whether you are using Windows, MacOS, or Linux. The commands that you are allowed to type in them also depend on your choice of OS. Below we describe how to use the command shell in each of the popular computer operative systems.

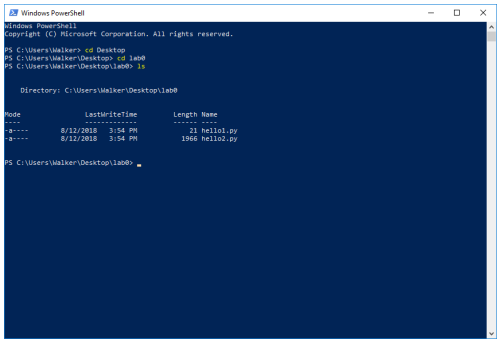
The specific commands on the various operating systems are similar, but they can also be very, very different (particularly between Windows and MacOS/Linux). If you have never used a command shell before, we highly recommand that you **pick one operating system and use it for the entire semester**. Otherwise, this can get very confusing. In particular, if you have a Macintosh laptop, we suggest that you bring it to the lab sections, as the lab computers are all Windows machines.

As with the Python installation instructions, this tutorial is very step-by-step. This is to give you a quick introduction to the command shell. But we are not expecting you to master the command shell immediately. You will get a lot of practice with it over the course of the semester.

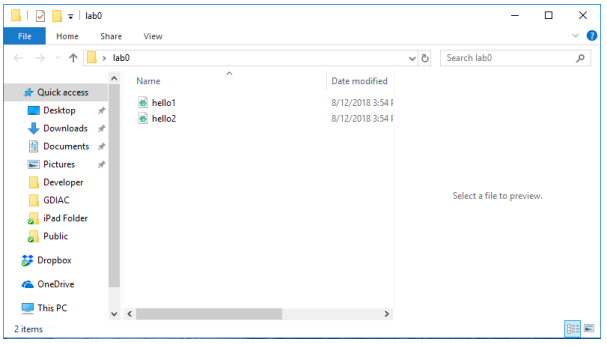
[Overview] [Windows] [Macintosh] [Linux]

Overview

Before learning the OS specific commands, it is important to understand what all command shells have in common. A command shell is a text-based version of a file manager. So it is the equivalent of Windows Explorer on Windows, or the Finder on MacOS. At any given time it is open to a specific folder (or *directory*) on your computer. We call the folder that is currently open in the shell the *working directory*. The pictures below show a command shell and a graphical file manager with the same working directory.



Windows PowerShell

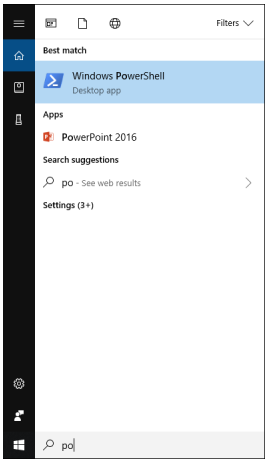


Windows Explorer

From within the command shell, you can do everything that you could do with a graphical file manager. You can move, rename, and copy files. You can change the current directory. You can even run programs. In a graphical file manager, you run a program by double-clicking on it; in a command shell, you type the name of the program. We cover this in more detail in the Python tutorial.

Every computer user has what is known as a *home directory*. This is the folder that has your name. In Windows 10, this is a hidden folder which contains all your other folders, like Downloads or the Desktop. In MacOS, it is the folder you typically see when you ask for a new Finder window. Whenever you open a new command shell, it always starts in the home directory. As your homework and lab assignments will often be in different folders, the first thing you need to learn about the command shell is how to change directories. That is the focus of the tutorials below.

Windows



In Windows, the command shell is called the **PowerShell**. There is another, older command shell called the **Command Prompt**. However, official Microsoft policy since Windows 10 is to use the newer PowerShell, and that is what we recommend. It is much closer to the MacOS and Linux versions we show in class, and so you will be less confused.

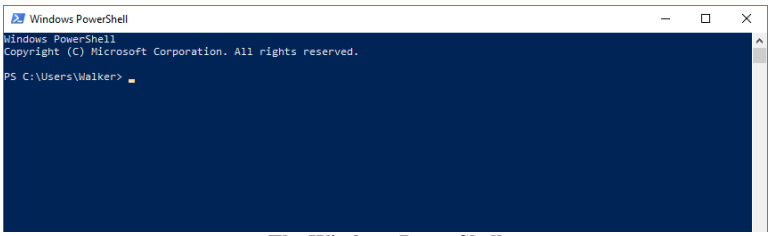
To find the PowerShell, just search for "PowerShell" in the search box at the bottom of the Start Menu. When you start up the PowerShell, you get a Window that looks something like the illustration below. At any given time, the bottom line of the PowerShell is the *working directory* followed by a > symbol.

To get the PowerShell to do something, simply type in a command, and hit **Return**. The shell will then process the command, either doing something or printing out an error message. When done, it will present the prompt again, ready for you to type in a new command.

As we mentioned above, the PowerShell works like the Windows Explorer. At any given time it is open to a specific folder (or *directory*) on your computer, which we call the *working directory*.

When working on a Python assignment, you want to make sure that the working directory is the directory that contains the `.py` files you are currently editing. Many a student has found themselves editing a `.py` folder while testing one (of the same name) in

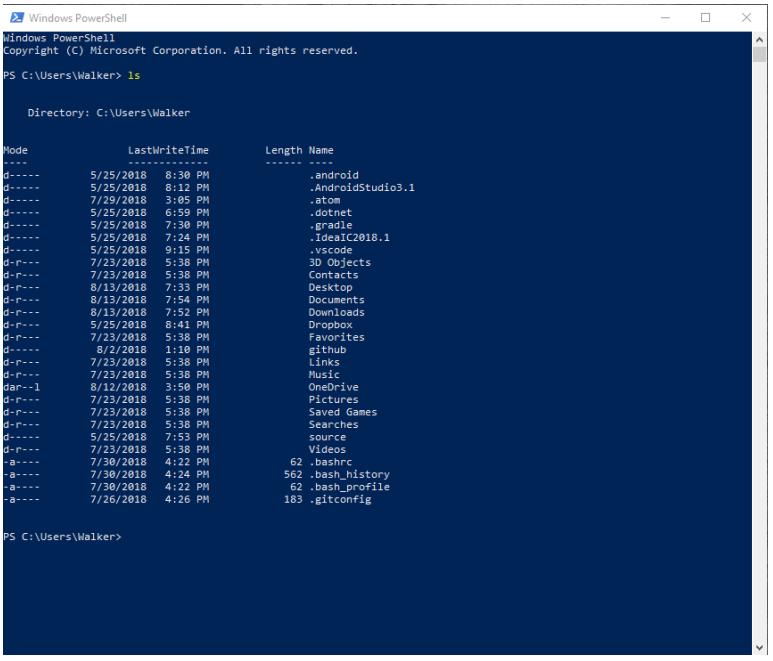
a different folder. You might be tempted to just put everything in your home directory. However, this is a bad idea and as the folder will get very cluttered as the semester progresses.



The Windows PowerShell

Navigating Directories

The two most important commands to know in Windows are **ls** and **cd**. Typing in **ls** displays the current working directory as well as all of its contents. An example of the **ls** command is shown below.



The **cd** command is an abbreviation for "change directory". It is how you move from one folder to another. When you type this command into the PowerShell, you must give it two things: the command **cd** and the name of the folder you wish to go to. Using the example above, suppose we wish to switch the working directory to Desktop. Then you would type

```
cd Desktop
```

Try this out and then type **ls**. See the difference?

There are a couple of important tricks to know about the **cd** command.

Backing Out of a Directory

The simplest form **cd** can only move to a folder that is "sees" (e.g. is a folder inside the working directory). If you change to directory (such as Desktop), you can no longer see the original directory (your home directory); it is outside of the current working directory. So how do you back-out if you go into a folder by mistake?

The solution is that there is a special folder called "`..`". This refers to the folder that contains the current one. Type

```
cd ..
```

and see what happens. If you typed it just after moving into the Desktop folder (from the previous example), then you should be back in your home directory.

Combining `cd ..` with regular uses of the **cd** command are enough to allow you to move up and down the directory hierarchy on your computer.

Tab Completion

If you are new to the PowerShell, you might find yourself quickly getting tired of all the typing that you have to do. Particularly when you have a directory with a very long name. A slight misspelling and you have to start all over again.

Fortunately, Windows has *tab completion* to speed things up. Go to your home directory and type (**but do not hit Return**)

```
cd D
```

Now hit the tab key. See what happens? Windows turns "D" into the first folder that it can find that starts with that letter (which is likely to be Desktop, and not Documents, as it comes first alphabetically).

Changing Multiple Directories at Once

Suppose you are currently in the your home directory; you want to move to the folder "Favorites" which is inside of "Documents". You could do this with two **cd** commands. But to do it with a single command, you just connect the folders with a \, as follows:

```
cd Documents\Favorites
```

When you combine this with `..`, you can do some rather clever tricks. Suppose you are currently in the Desktop directory, and you want to move in the Documents directory (which is contained in your home directory). You can do this with the command

```
cd ..\Documents
```

We refer to these expressions as *paths*; they are are a "path" from the working directory to the directory that you want to go to.

Absolute Paths

The paths that we have shown you are more properly called *relative paths*. They show how to get from the working directory to your new directory. The correct path to use depends on exactly which directory is the current working directory.

Absolute paths are paths that do not depend on the working directory; instead they depend on the disk drive. They always start with name of the drive. For example, suppose you inserted a USB drive into the computer, and you wanted to open that drive in the PowerShell. The USB drive will (typically) be the the E: drive, so you simply type

```
cd E:
```

You can combine this with the \ symbol to move anywhere you want on the USB stick. If the USB stick has a folder called "Python" on it, simply type

```
cd E:\Python
```

Any time that you need to change disk drives, you need to use absolute paths. If your user account is called "Sally", then you return to your home directory by typing

```
cd C:\Users\Sally
```

Folder Names with Spaces

The PowerShell breaks up the commands that you type in by spaces. That means that if you have a folder with spaces in the name, it will break it up into references to two different folders. For example, suppose you have a folder called "Python Examples", and you type

```
cd Python Examples
```

You will get an error saying that Windows cannot find that path.

To solve the problem, put the directory in quotes. The following should work correctly.

```
cd "Python Examples"
```

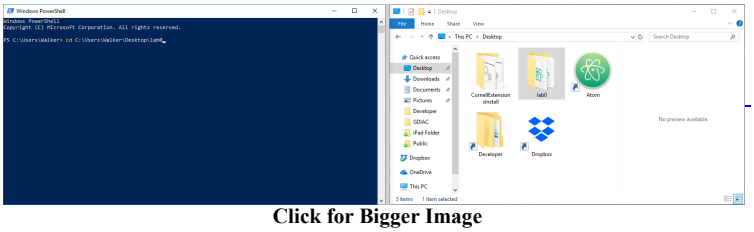
If you are changing multiple directories then you need to put the entire path in quotes (not just the folder). For example, if you want to go to "Program Files" on the C drive, type

```
cd "C:\Program Files"
```

The Drag-and-Drop Shortcut

If you do not learn anything else about the PowerShell, you should learn this one trick (which works on MacOS and most Linux systems as well). If you take a folder and drag-and-drop it onto the PowerShell, it will fill the window with the absolute pathname of that folder. Therefore, to quickly move the PowerShell to a a specific folder, do the following:

- Type `cd` followed by a space.
- Drag and drop the folder on to the PowerShell.
- Hit **Return**.



Click for Bigger Image

This is a very useful skill and you will see your instructor use it often in class.

Manipulating Files (OPTIONAL)

The PowerShell allows you to do everything that Windows Explorer can do (and more). You can use the PowerShell to make folders, move files, and delete files. However, none of this is necessary for you to learn. For this class, **you never need to understand how to do anything other than navigate directories**. You can do everything else in Windows Explorer (or some other program) if you wish.

Make a Directory

To make a new folder or directory, use the command **mkdir** followed by the name of the new folder. For example:

```
mkdir MyFolder
```

The new folder will appear in the current working directory.

You can also delete a directory with the **rmdir** command. For example, to delete the folder we just made, type

```
rmdir MyFolder
```

The PowerShell will only delete empty directories. If there is anything in a directory, it will not let you delete it. You have to delete the contents first.

Move (or Rename) a File

You move files with the **move** command. The way this command works is that you give it two file names. It searches for a file with the first file name; once it finds it, it makes a copy with the new file name and then deletes the original.

For example, suppose you wanted to rename the file `test.py` to `assignment3.py`. Then you would type

```
move test.py assignment3.py
```

(this by the way, illustrates why paths cannot have spaces in them).

If the second filename is path to a file, then it will move the the file into the correct directory. For example, suppose we now wanted to move `assignment3.py` to the Desktop (which is a folder in the current working directory), and rename it `completed.py`. Then we would type

```
move assignment3.py Desktop\completed.py
```

If we want to keep the name as `assignment3.py`, you could shorten this to

```
move assignment3.py Desktop
```

In this case, the PowerShell will move `assignment3.py` into Desktop, but keep the name of the file unchanged.

Copy a File

The **move** command will always delete the original (name of) the file when it is done. Sometimes we want to make a copy of a file. We do that with the **copy** command. Suppose that `assignment3.py` is in the working directory and we want to put a copy on the Desktop without deleting the original. Then we would type

```
copy assignment3.py Desktop\assignment3.py
```

Delete a File

Files are deleted with the **del** command. In our running example, to delete the file `assignment3.py`, you would type

```
del assignment3.py
```

Be very careful with this command. It completely erases the file. **It does not move the file your Recycle Bin**. You cannot retrieve a file deleted this way.

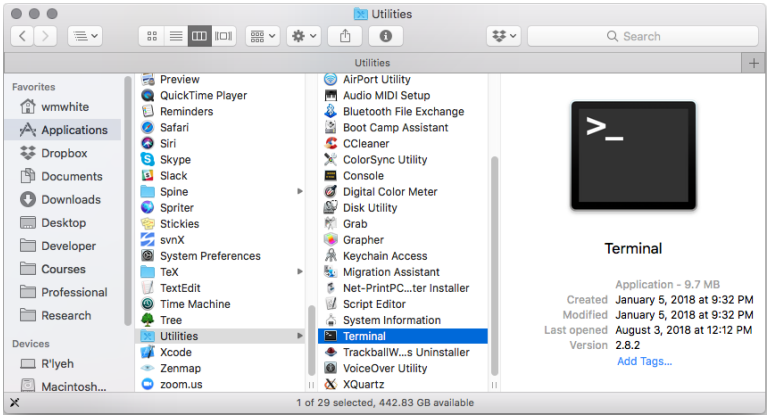
Getting Help

There are hundreds of resources out there on how to learn to use the PowerShell in Windows. If you want to learn more, we suggest this tutorial as a starting point.

If have are having difficulty with the PowerShell, please see one of the course staff. They are available to help.

Macintosh

On the Macintosh, the command shell is called the **Terminal**. If it is not in your Dock (where it belongs!), it can be found in the **Applications** > **Utilities** folder as shown below. We recommend putting it in your Dock immediately.



When you start up the Terminal, you will get some message about the "last login" (a holdover of the days in which Terminals were used to connect machines over the network) followed by a line with a cursor that looks like a box. The left side of the line will depend on your settings, but the last symbol will likely be either a $ or a >. This symbol is called the prompt, and it is a cue for you to type something into the Terminal.

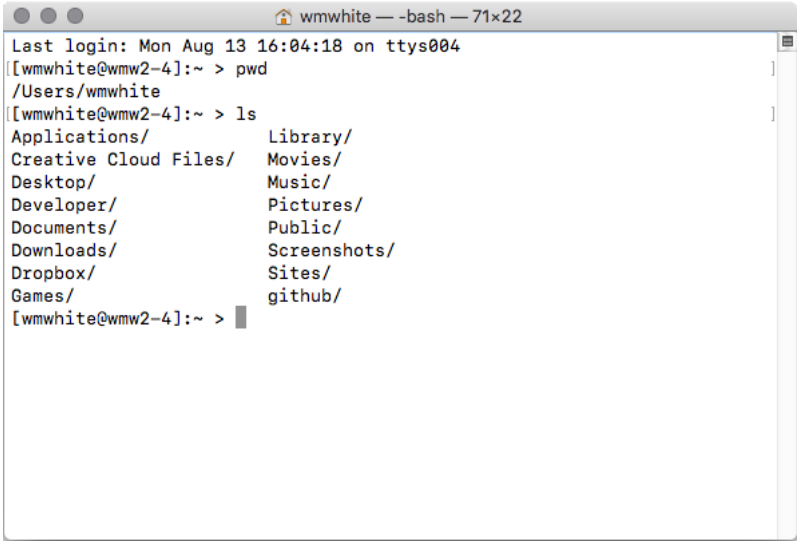
To get the Terminal to do something, simply type in a command, and hit **Return**. The shell will then process the command, either doing something or printing out an error message. When done, it will present the prompt again, ready for you to type in a new command.

As we mentioned above, the Terminal works a lot like the Finder. At any given time it is open to a specific folder (or *directory*) on your computer, which we call the *working directory*.

When working on a Python assignment, you want to make sure that the working directory is the directory that contains the `.py` files you are currently editing. Many a student has found themselves editing a `.py` folder while testing one (of the same name) in a different folder. You might be tempted to just put everything in your home directory. However, this is a bad idea and as the folder will get very cluttered as the semester progresses.

Navigating Directories

Because you often need to change your working directory, the three most important commands to know in the Terminal are **pwd**, **ls**, and **cd**. Typing in **pwd** displays the current working directory. The command **ls** lists the contents (files and folders) in the working directory. An example of these two commands is shown below.



The **cd** command is an abbreviation for "change directory". It is how you move from one folder to another. When you type this command into the Terminal, you must give it two things: the command **cd** and the name of the folder you wish to go to. Using the example above, suppose you wish to switch the working directory to Desktop. Then you would type

```
cd Desktop
```

Try this out and then type **ls**; see the difference?

There are a couple of important tricks to know about the **cd** command.

Backing Out of a Directory

The simplest form **cd** can only move to a folder that is "sees" (e.g. is a folder inside the working directory). If you change to directory (such as Desktop), you can no longer see the original directory (your home directory); it is outside of the current working directory. So how do you back-out if you go into a folder by mistake?

The solution is that there is a special folder called "`..`". This refers to the folder that contains the current one. Type

```
cd ..
```

and see what happens. If you typed it just after moving into the Desktop folder (from the previous example), then you should be back in your home directory.

Combining `cd ..` with regular uses of the **cd** command are enough to allow you to move up and down the directory hierarchy on your computer.

It is also possible to type `cd` by itself, without a directory name. If you do this, it will immediately put you back in your home folder. This is very helpful should you ever get lost while using the Terminal.

Tab Completion

If you are new to the Terminal, you might find yourself quickly getting tired of all the typing that you have to do. Particularly when you have a directory with a very long name. A slight misspelling and you have to start all over again.

Fortunately, MacOS has *tab completion* to speed things up. Go to your home directory and type (**but do not hit Return**)

```
cd Desk
```

Now hit the tab key. See what happens? It completes the work "Desk" to "Desktop", because it is the only thing in your home folder that starts with "Desk" (if you actually do have something else in your folder that starts with "Desk", this example will not work).

As another example type (but do not hit Return)

```
cd D
```

and hit tab again. There are at least two things in your home directory that start with D: Desktop and Documents. MacOS does not know which one to complete to, so it lists the possibilities for you. Tab autocompletion only works when the Terminal has enough information to uniquely pick one option from the current folder. Try doing this again with

```
cd De
```

What happens?

Changing Multiple Directories at Once

Suppose you are currently in the your home directory and you want to move to the folder "iTunes" which is inside of "Music". You could do this with two **cd** commands. But to do it with a single command, you just connect the folders with a `/`, as follows:

```
cd Music/iTunes
```

When you combine this with `..`, you can do some rather clever tricks. Suppose you are currently in the Desktop directory, and you want to move in the Documents directory (which is contained in your home directory). You can do this with the command

```
cd ../Documents
```

We refer to these expressions as *paths*; they are are a "path" from the working directory to the directory that you want to go to.

Absolute Paths

The paths that we have shown you are more properly called *relative paths*. They show how to get from the working directory to your new directory. The correct path to use depends on exactly which directory is the current working directory.

Absolute paths are paths that do not depend on the working directory. In MacOS (and all Unix systems), absolute paths start with a `/`. This `/` represents the *root* directory that contains everything else. For example, if you wanted to go to your Applications directory (which is just inside the root directory), you would type

```
cd /Applications
```

Absolute paths are very important when you are trying to navigate to a different disk drive. In MacOS, when you plug in a new disk drive it is added to the `/Volumes` folder (note the `/` indicating that Volumes is just inside the root folder). Suppose you have a Kingston USB drive from the Campus store named `KINGSTON`. To view the contents of this drive in the terminal, type

```
cd /Volumes/KINGSTON
```

To drive home the difference between relative and absolute paths, create a folder called "Applications" in your home directory. Make sure the terminal is in the home directory (go home by typing `cd` by itself) and type

```
cd Applications
```

Look at the contents with **ls**. Now go back to the home directory again and type

```
cd /Applications
```

Look at the contents with **ls**. See the difference?

Folder Names with Spaces

The Terminal breaks up the commands that you type in by spaces. That means that if you have a folder with spaces in the name, it will break it up into references to two different folders. For example, suppose you have a folder called "Python Examples", and you type

```
cd Python Examples
```

You will (likely) get an error saying that the folder "Python" does not exist.

To solve the problem, put the directory in quotes. The following should work correctly.

```
cd "Python Examples"
```

If you are changing multiple directories then you need to put the entire path in quotes (not just the folder). For example, if "Python Examples" were on the Desktop, you would type

```
cd "Desktop/Program Files"
```

Alternatively, you can represent a space using the *escape character* `\` which we talked about in class. For example, the following should also work correctly:

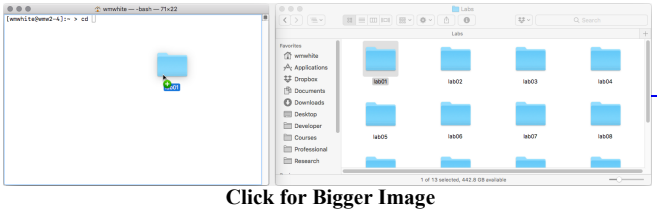
```
cd Python\ Examples
```

If you use Tab Completion a lot, you will notice that this is the prefered way of handling spaces.

The Drag-and-Drop Shortcut

If you do not learn anything else about the Terminal, you should learn this one trick (which works on Windows as well). If you take a folder and drag and drop it onto the Terminal, it will fill the window with the absolute pathname of that folder. Therefore, to quickly move the Terminal to a a specific folder, do the following:

- Type `cd` followed by a space.
- Drag and drop the folder on to the Terminal window.
- Hit **Return**.



Click for Bigger Image

This trick works on Windows and Linux as well. However, MacOS has an even faster trick that is unique to its operating system. Simply take the folder icon and drop it onto the Terminal *icon* (in your Dock), and it will open a new Terminal window with that folder as its working directory. This is a very useful skill and you will see your instructor use it often in class.

Manipulating Files (OPTIONAL)

The Terminal allows you to do everything that Finder can do (and more). You can use the Terminal to make folders, move files, and delete files. However, none of this is necessary for you to learn. For this class, **you never need to understand how to do anything other than navigate directories**. You can do everything else in the Finder (or some other program) if you wish.

Make a Directory

To make a new folder or directory, use the command **mkdir** followed by the name of the new folder. For example, type:

```
mkdir MyFolder
```

The new folder will appear in the current working directory.

You can also delete a directory with the **rmdir** command. For example, to delete the folder you just made, type

```
rmdir MyFolder
```

The Terminal will only delete empty directories. If there is anything in a directory, it will not let you delete it. You have to delete the contents first.

Move (or Rename) a File

You move files with the **mv** command. The way this command works is that you give it two file names. It searches for a file with the first file name; once it finds it, it makes a copy with the new file name and then deletes the original.

For example, suppose you wanted to rename the file `test.py` to `assignment3.py`. Then you would type

```
mv test.py assignment3.py
```

(this by the way, illustrates why paths cannot have spaces in them).

If the second filename is path to a file, then it will move the the file into the correct directory. For example, suppose you now wanted to move `assignment3.py` to the Desktop (which is a folder in the current working directory), and rename it `completed.py`. Then you would type

```
mv assignment3.py Desktop/completed.py
```

If you want to keep the name as `assignment3.py`, you could shorten this to

```
mv assignment3.py Desktop
```

In this case, the Terminal will move `assignment3.py` into Desktop, but keep the name of the file unchanged.

Copy a File

The **mv** command will always delete the original (name of) the file when it is done. Sometimes you want to make a copy of a file. We do that with the **cp** command. Suppose that `assignment3.py` is in the working directory and we want to put a copy on the Desktop without deleting the original. Then you would type

```
copy assignment3.py Desktop/assignment3.py
```

Delete a File

Files are deleted with the **rm** command. In our running example, to delete the file `assignment3.py`, you would type

```
rm assignment3.py
```

Be very careful with this command. It completely erases the file. **It does not move the file your Trash**. You cannot retrieve a file deleted this way.

Getting Help

There are many resources out there on how to learn to use the Terminal in OS X. If you want to learn more, we suggest this tutorial as a starting point.

If have are having difficulty with the Terminal, please see one of the course staff. They are available to help.

Linux

Let's be honest here. If you use Linux, do you really need to learn how to use the command shell? How is it possible to do anything in Linux without knowing how to use the command shell?

On the off chance that you honestly have never used a command shell in Linux, the hard part is finding the program that provides access to the shell. Which program to use depends on your choice of GUI.

- X11: xterm
- Gnome: Gnome Terminal
- KDE: konsole

Once you have that running, simply refer to the instructions for Macintosh. While the programs are not exactly the same (particularly if you are running a shell other than Bash), they are close enough for purposes of this class.

If you need more help with using the command shell in Linux, please see one of the course staff.

Course Material Authors: D. Gries, L. Lee, S. Marschner, & W. White (over the years)