GOOGLE IT SUPPORT SPECIALIZATION
COURSE 3: OS POWER USER
01 NAVIGATING THE SYSTEM

Table des matières

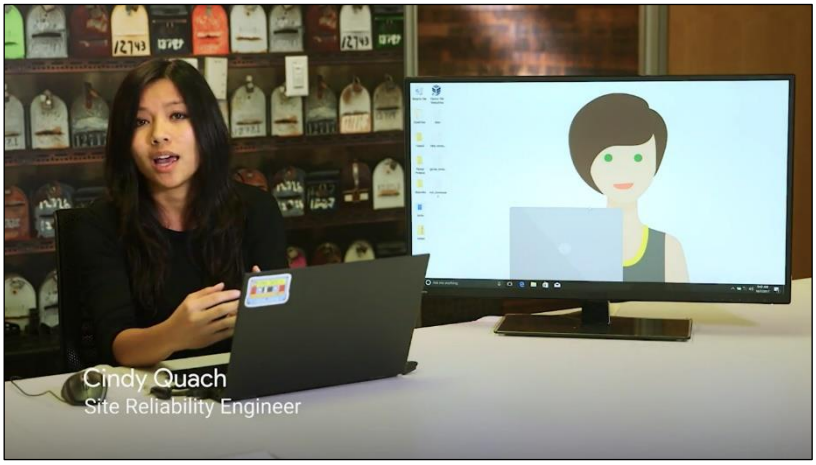
1 Navigating the system

1.1 Introduction to Operating systems and becoming a power user

1.1.1 Course introduction

You've already learned the basics of computing and you just finished learning about the bits and bytes of computer networking. Now it's time to navigate the Windows and Linux Operating Systems or OSs. But before we dive in, I'd like to introduce myself. We met way back in the first course.

But for those of you who might have forgotten or skipped those lessons, my name is Cindy Quach, and I'm a site reliability engineer at Google.



The team I work on is responsible for the management and support of Google's entire internal mobile fleet; Android OS, iOS, and chrome OS. Before focusing on mobile, I was a systems administrator on the Linux team, and before that, I was an operations engineer. But like a lot of the Googlers you've met and will meet, I started my career as an IT support specialist. I've been working in IT for seven years now.

The first time I can remember interacting with computers was in middle school, when my teacher brought them into our classrooms so we could create fun video and multimedia projects. It was my brother who brought technology into our house. My parents were immigrants from Vietnam and we didn't have a lot of money growing up, so we had to be creative if we wanted to play with a computer at home. I can remember we were spending hours with my brother as he assembled a computer and I would just ask a million questions.

Eventually, I wanted to try and build my own computer. So, I gathered up some old parts and saved money to buy new components. I finally put all the parts together from what I remembered my brother doing. But it didn't work out. It turns out I use some incompatible parts but through a lot of trial and error, troubleshooting, and long search sessions on the internet. I finally got it to work. The feeling I got when I heard my computer boot up for the first time was amazing, and before I knew it, I was hooked on computers. I really enjoyed the intense concentration and problem solving required in IT. But I didn't think a career in tech was even possible but then.

Once I got to college, I had to find a job to help pay for tuition. And that job was an IT support specialist on campus. That's when I realized that tech is actually something I could pursue as a career. I've been working with computers for as long as I can remember and much of my IT knowledge was based on my own troubleshooting experiences over the years. I was great at troubleshooting issues with operating systems or so I thought, it wasn't until I became a systems administrator on Google's Linux team that I realized just how little I knew about operating systems. I was surrounded by brilliant teammates who maintain code for a large open source operating system projects. Some even had Wikipedia pages written about them. So it was hard not to feel inadequate at times. Like I was learning to walk again as I dove more into Linux, I just wasn't used to working on the command line and it felt overwhelming to use it to troubleshoot obscure issues that popped

up. I had to constantly look up commands and figure out where to find certain files, but I didn't let it get the best of me. I took things day by day, and after a year of being on the team, I realized I had progressed incredibly far.

One year later, I was building and packaging my own tools then deploying them for everyone to use. As contributing code directly to open source software. Using the command line, had become second nature. There's so much to learn about operating systems and it's one of the reasons why I'm passionate about teaching this course. Learning Linux doesn't have to be scary. It's not impossible to use Windows commands and it's certainly not difficult to get started. So let's just do that and get started. While this course will have some conceptual learning. Will focus more on the practical aspects of the operating system. Not only when you learn how to use the Windows and Linux OSs, we'll also teach you how to interact with these operating systems through the command line. Remember that the command line inputs text commands instead of relying on a graphical user interface or GUI.

If this is your first time using a command line for any OS, you may find this a little intimidating at first. That's totally normal, but you'll be well on your way to become a command line wizard by the end of this course. As always, we'll help guide you every step of the way, and you can always re-watch the lessons if you need to take a refresher. Take your time, you got this. We're not only going to teach you how to use the command line in Windows and Linux. You'll also learn how file systems work, and you'll be able to assign different user permissions and roles, which is a super important task in any support role. You'll be able to understand how to use package managers and consider the trade offs between different package managers for Windows and Linux. We'll also teach you about process management, so you understand the nuances of running programs. That could save you valuable time when troubleshooting in the workplace. We'll also take a deeper dive into the remote connection tools you've already been using to help you access other computers, when you're working at a distance.

Finally, we'll teach you about OS deployment or how to install OSs on a lot of machines at once. By the end of this course, you'll become a real OS power user, in both the Windows and Linux operating systems. This is an invaluable skillset for anyone pursuing a career as an IT support specialist. After all, we spend most of our time within an operating system. But remember, you'll need to practice, practice, and practice some more to get a firm grip on operating systems. Just like with any skill, you need to really apply yourself to get good at it. Eventually, navigating the operating system will seem like second nature to you. We strongly recommend that you follow along in this course with a computer using one if not both of these operating systems. Navigating a real operating system while following along this course, is a much more efficient way to learn these concepts. If you don't have access to them that's totally okay. You'll be doing active learning exercises in an application called Qwiklabs, to help simulate what it's like to use the Windows and Linux OS. I'm super excited to teach you about Windows Linux OSs. So let's get started.

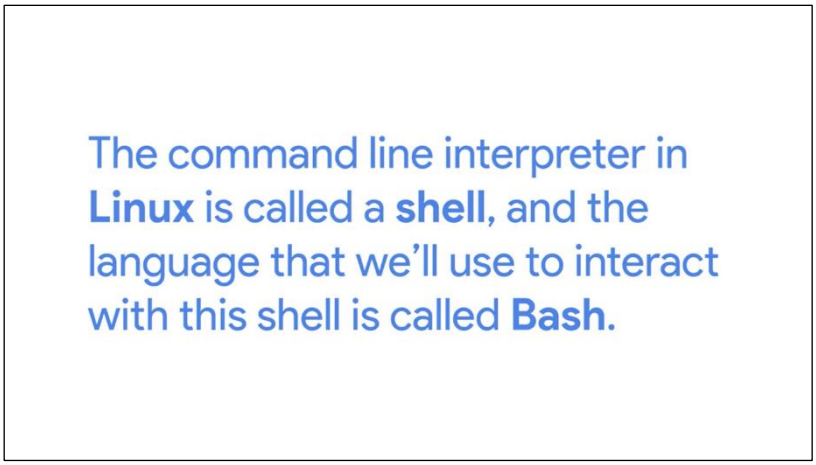
1.2 Basic commands

1.2.1 Lesson overview practice tips

We dipped our toes in the Windows and Linux OS's in the first course of this program. Now, let's jump right in and learn how to perform all the common navigational tasks of both operating systems.

For Windows, we're going to learn how to navigate the operating system using the GUI and using the command line interpreter or CLI.

For Linux, we're only going to focus on learning the command line. The command line interpreter in Linux is called a shell, and the language that we'll use to interact with the shell is called Bash.



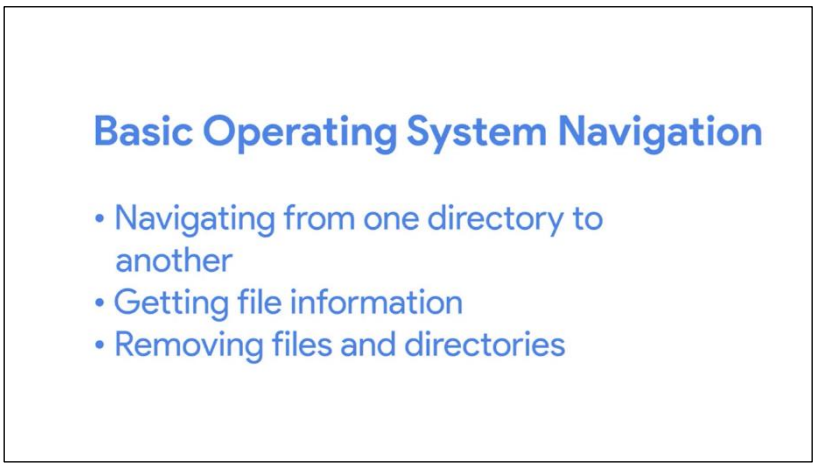
It's worth calling out that these two operating systems are very similar to one another. So, even if you don't know how to use the Linux GUI, as long as you know how to navigate the Windows GUI, you'll be able to apply those tools to the Linux GUI. It's possible that you'll only be using the Windows GUI in the workplace. Even so, if you learn how to use the Windows command line, this will set you apart from other IT support specialists.

You'll soon discover that using the command line in any operating system can actually help you complete your work faster and more efficiently. We surely encourage you to follow along and actually perform the task we do in this course yourself. If you can, pause a video and do the exercises that

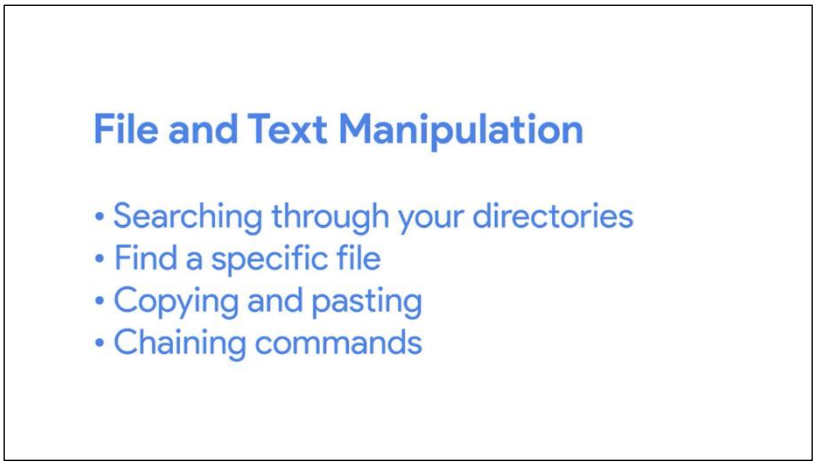
we do or type out any of the commands we introduce. It will be much easier for you to understand them in this way. We also recommend that you document all the commands that we show you. Either write them down with an old-fashioned pen and paper notebook, or type them out in a doc or text editor. Just write them on a stone if you have to, we just want you to write them down somewhere.

You probably won't remember all the commands immediately when we first introduced you to them, but with a little practice, typing the commands will become second nature to you. You can also use the official Windows CMI and Bash documentation that we've provided for you in the supplemental reading, right after this video for reference, if you need to.

In this lesson, the content is broken down into two themes. The first is basic operating system navigation, like navigating from one directory to another, getting file information, and removing files and directories.



The second theme is file and text manipulation, like searching through your directories to find a specific file, copying and pasting, chaining commands and more.

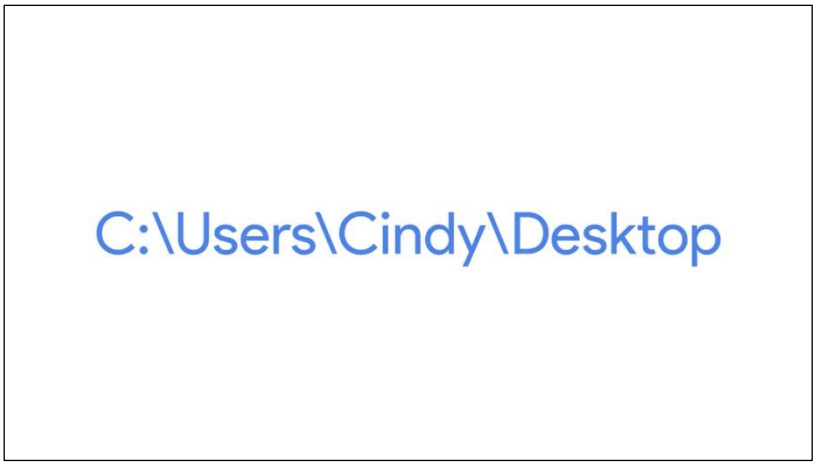


Okay. Enough chit-chat. Let's get started.

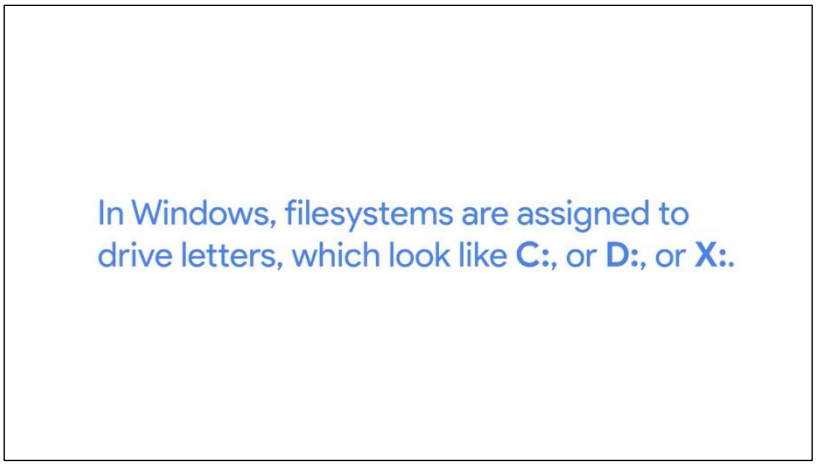
1.2.2 List directories in a GUI

In operating systems, files and folders or directories are organized in a hierarchical directory tree. You have a main directory that branches off and holds other directories and files. We call the location of these files and directories, paths.

Most paths in Windows looks something like this C:\Users\Cindy\Desktop.

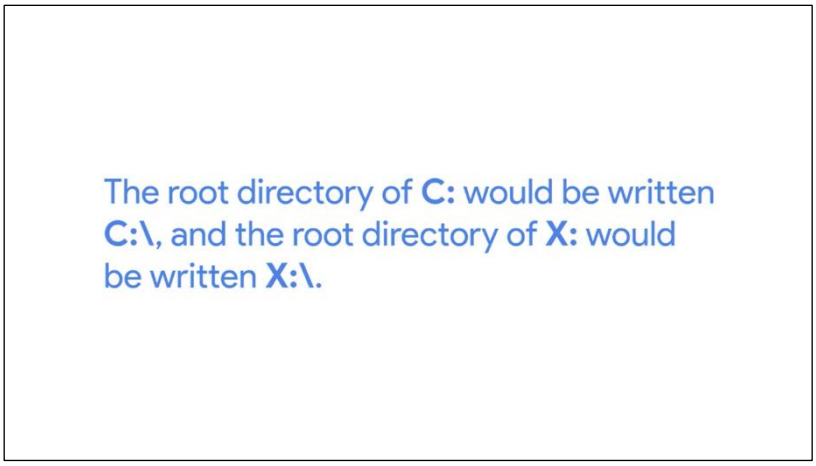


In Windows, file systems are assigned to drive letters which look like C:, or D:, Or X:.

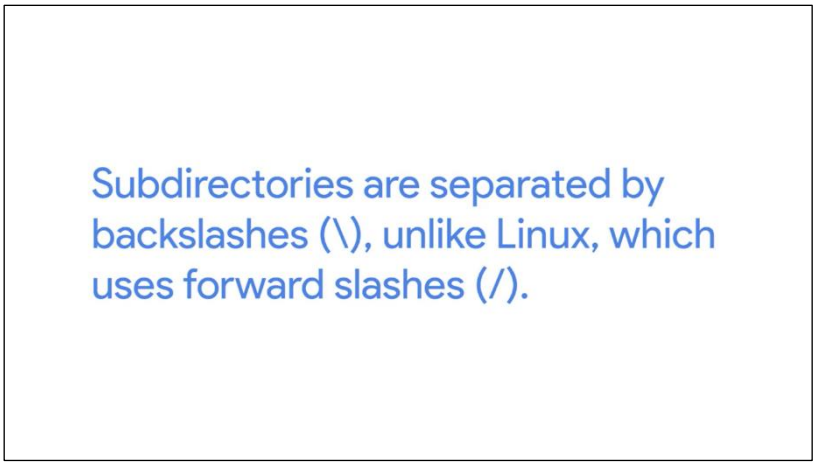


Each drive letter is a file system. Remember that file systems are used to keep track of files on our computer. Each file system has a root directory which is the parent for all other directories in that

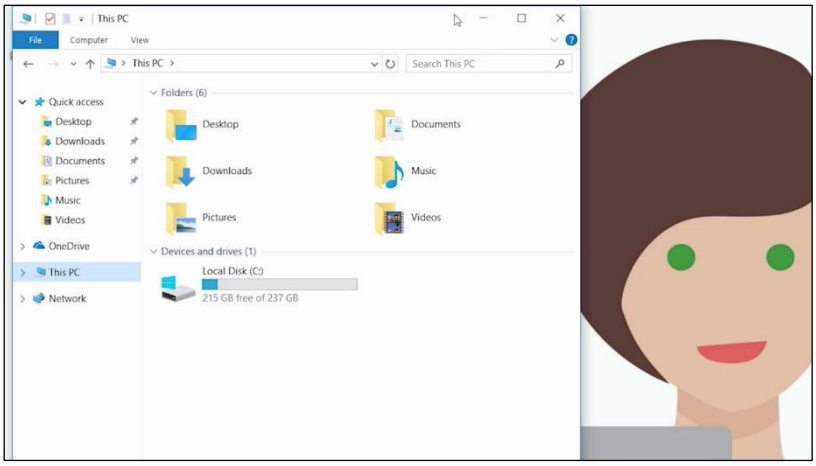
file system. The root directory of C: would be written C:\, and the root directory of X: would be written X:\.



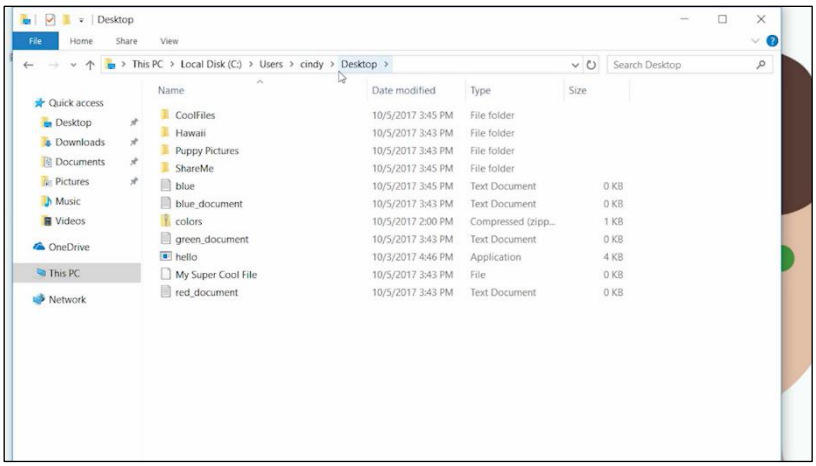
Subdirectories are separated by backslashes, unlike Linux, which uses forward slashes.



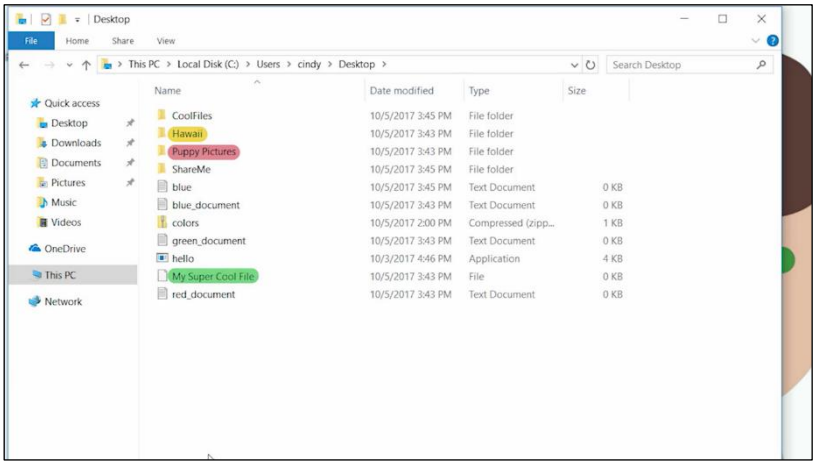
A path starts at the root directory of a drive and continues to the end of the path. Let's open up this PC and navigate to our main directory. The main directory in a Windows system is the drive that the file system is stored on. In this case, our file system is stored on Local Disk C.



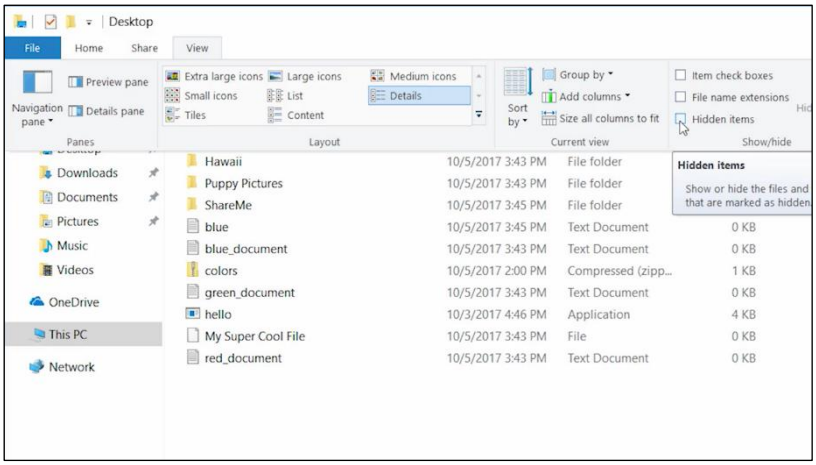
From here, I'm going to go to Users, then my User folder cindy, and finally to Desktop. If you look at the top here, you can see the path I'm in. Local disk, Users, cindy, Desktop. That wasn't too hard, right?



You can see here in our desktop directory that we have a few folders and files. We have a Puppy's Pictures folder, a Hawaii folder, and a file called My Super Cool File.

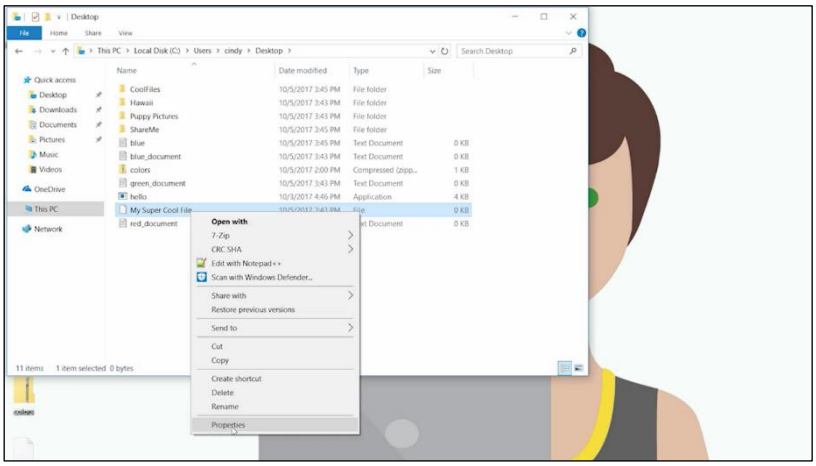


There are also some files on here that you can't see. We call these hidden files. They're hidden for a few reasons. One is that we don't want anyone to see or accidentally modify these files. They could be critical system files or configs or even worse, embarrassing pictures of you in grade school rocking a mullet. It's okay, you aren't the first person who like their hair to be business in the front and party in the back. Just for fun, let's see what kind of hidden files we have in here. We'll go to the top and click View, then check the hidden items checkbox. Now we can see all the hidden files on our system.

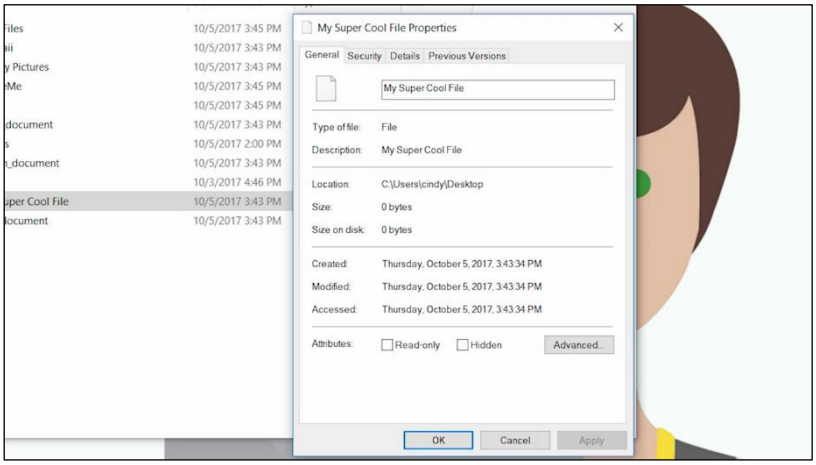


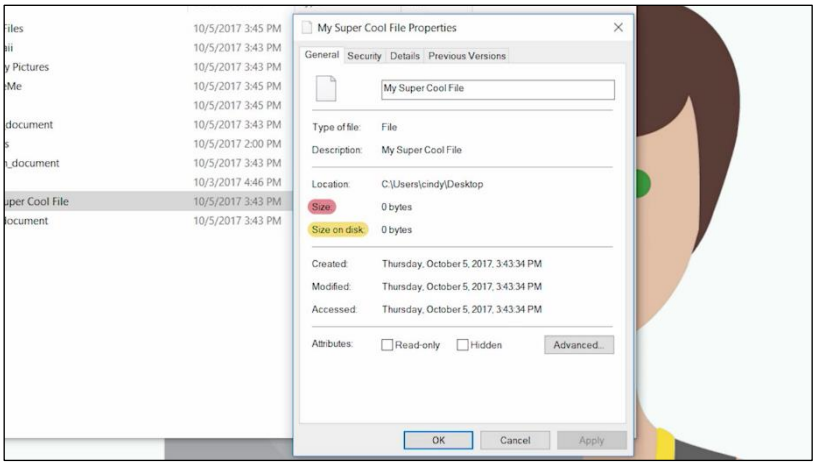
Oh, interesting. There is a file named secret_file. As much as I'd like to take a peek at it, whoever created it probably doesn't want us to see what's inside so we're going to leave it alone. Let's go ahead and revert this option so we don't accidentally change something else.

Okay, so what if we wanted to view information about a file? Well, to do this, we can actually just right click and choose Properties. Let's try this for My Super Cool File.



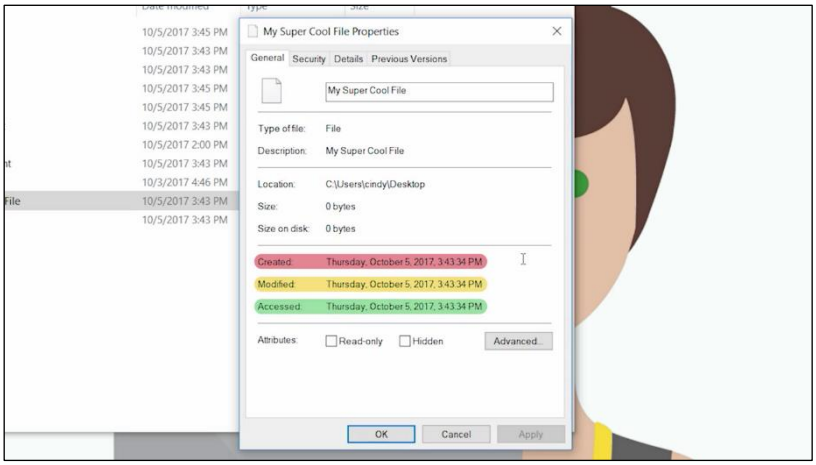
This pop up dialog has a lot of information displayed here. Let's break it down. In the general tab, we can see the file name, the type of file, what applications we use to open it, and the location path of the file which is C\Users\cindy\Desktop, then we have the size of the file, and the size on disk.



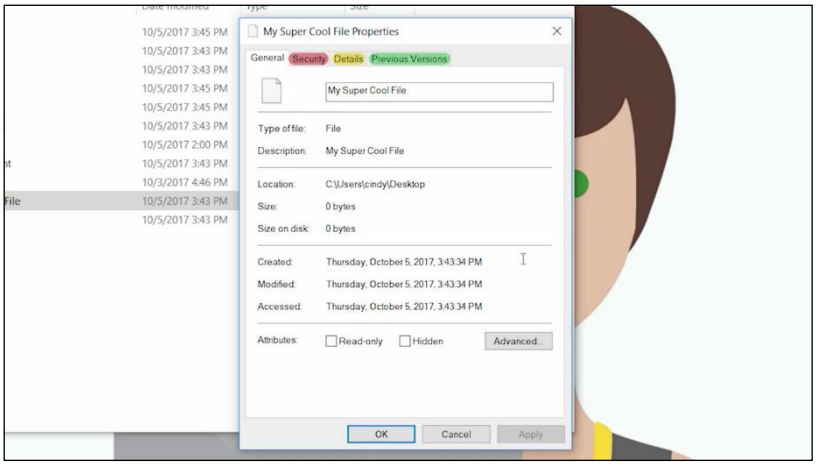


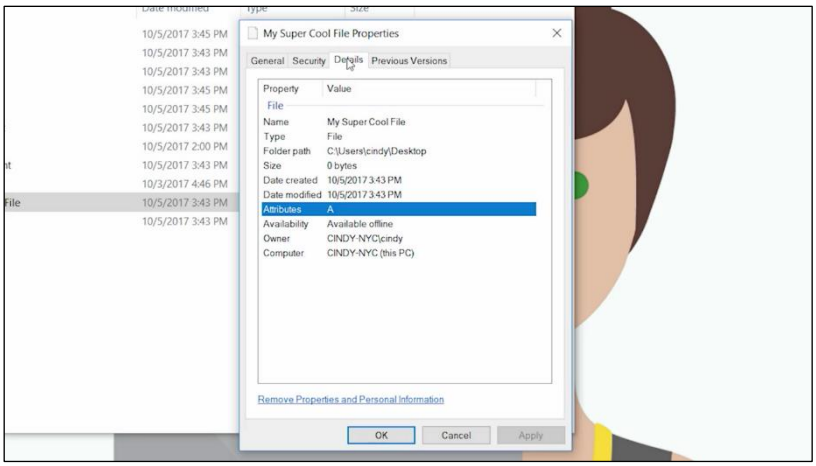
This can be a little confusing. The size of the file is actually the amount of data that it takes up, but size on disk is a little different. It's not something you need to know right now but if you want to learn more about it, you can check out the next supplemental reading. All right, let's move on.

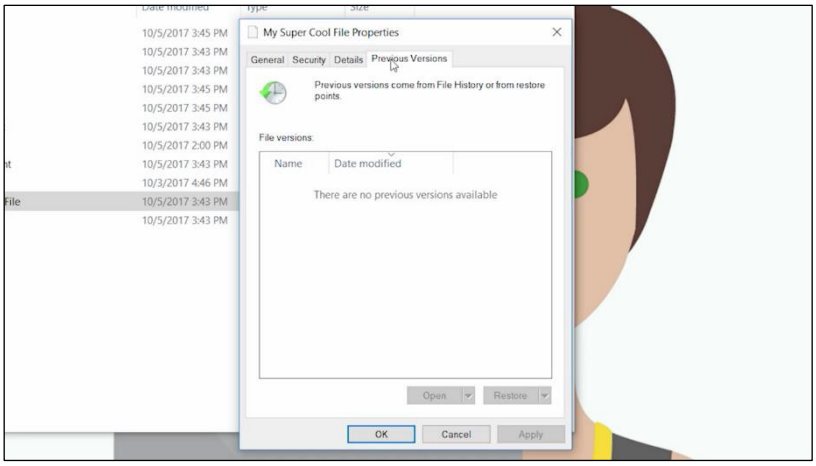
Next you have timestamps of when the file was created, last modified, and last accessed. After that our file attributes we can enable for our file. We have Read-Only and Hidden. You might guess that if you check hidden, our file will be hidden and only visible if we enable show hidden items.



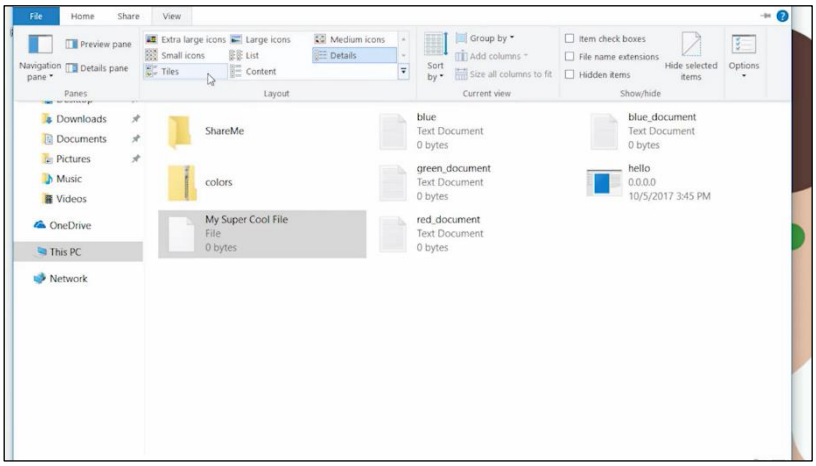
There are some advanced options too but we won't touch those for now. You'll also notice a few other tabs here at the top. Security, Details, and Previous Versions. We'll talk more about the security tab in a later lesson. The Details tab, basically, tells us the information we just discussed about a file. The Previous Versions tab lets us restore an earlier version of a file so if you made a change to a file and wanted to revert to that change, you could go back to that version.







To sum up listing the directories in the Windows GUI, we can see the list of files and folders by default here. You can even change how you want to view them using icons or even a list. Then if you want to get more information about a file, you can look at its properties. Next up, let's see how to view all this information through the Windows CLI.

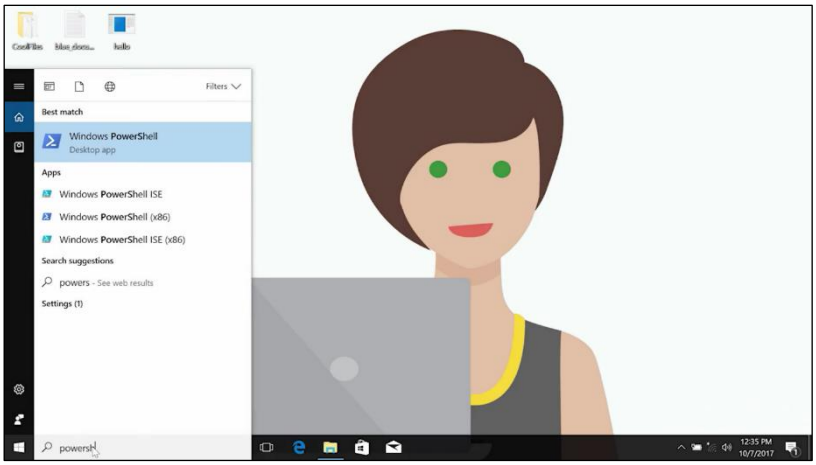


1.2.3 Windows list directories in a CLI

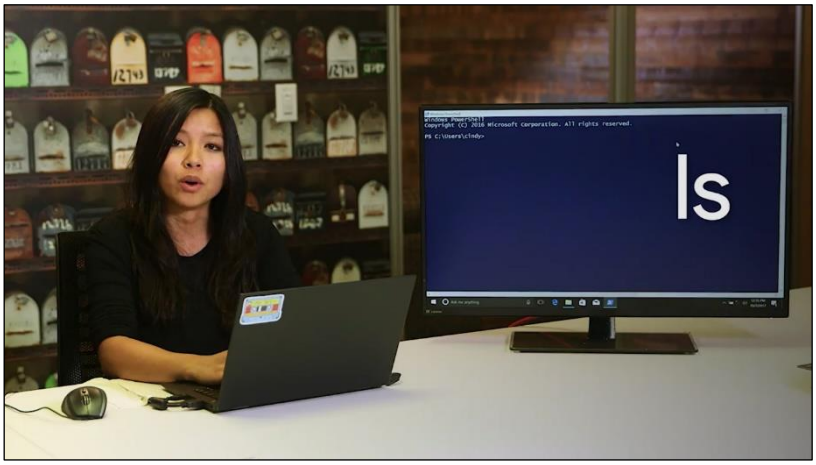
It's important to know that there are a couple of command line interfaces or CLIs available in Windows. The first one is called the Command Prompt, command.exe. The second one is PowerShell or powershell.exe.

The command prompt has been around for a very long time. It's very similar to the Command Prompt that was used in MS DOS. Since PowerShell supports most of the same commands as Command Prompt and many, many more, we're going to use PowerShell for the exercises in this module. I want to call out that many PowerShell commands that we use are actually aliases for common commands in other shells. An alias is sort of like a nickname for a command.

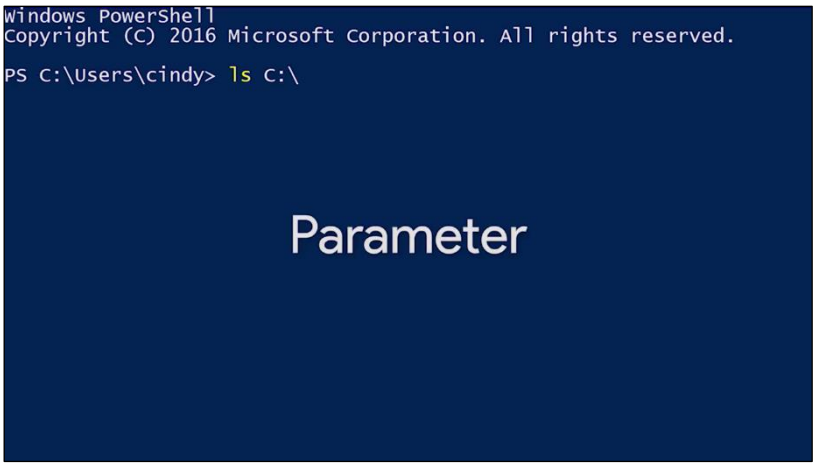
The first command that we'll use is for listing files and directories. Let's start by listing the directories in the root of our C: drive. The C: drive is where the Windows operating system is installed. For many of you, it might be the only hard drive that you have in your computer. To get to the PowerShell CLI, just search in your application's list PowerShell.



From here, we can go ahead and launch the PowerShell program. We're going to use the ls or list directory command and give it the path of where we want to look.

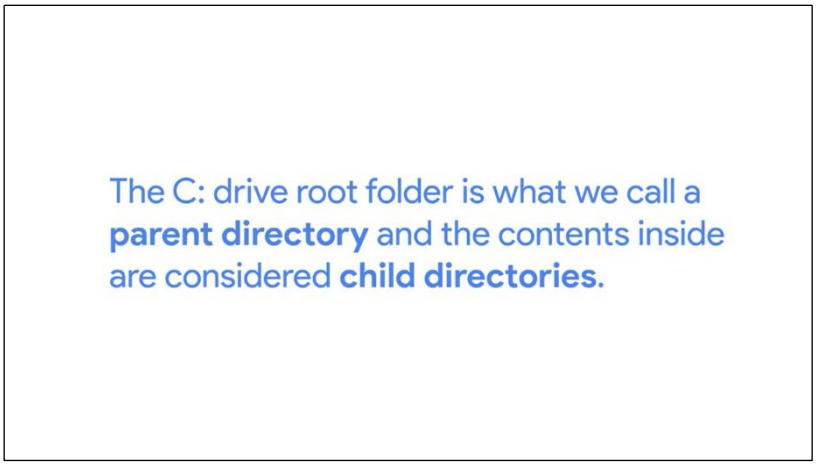


The path is not actually part of the command but it is a command parameter. You can think of parameters as a value that's associated with a command.



```
Windows PowerShell
Copyright (C) 2016 Microsoft Corporation. All rights reserved.

PS C:\Users\cindy> ls C:\


    Directory: C:\


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        10/5/2017   3:40 PM                Intel
d-----        3/18/2017   2:03 PM                PerfLogs
d-r---        10/5/2017   3:46 PM                Program Files
d-r---        10/5/2017   3:29 PM                Program Files (x86)
d-r---        10/5/2017   3:38 PM                Users
d-----        10/5/2017   3:44 PM                Vacation Pictures
d-----        10/5/2017   3:42 PM                Windows


PS C:\Users\cindy>
```

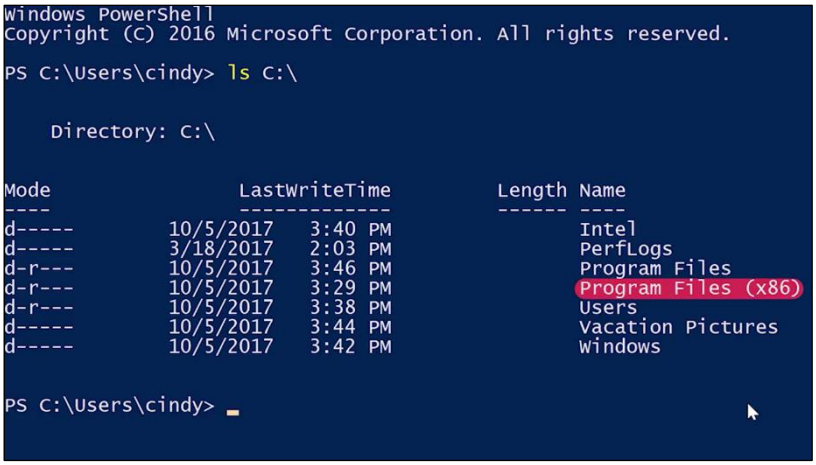
Now you can see all the directories in the root of your C: drive. You might just see a few or a whole bunch of directories. It all depends on what your computer is used for. The C: drive root folder is what we call a parent directory and the contents inside are considered child directories.



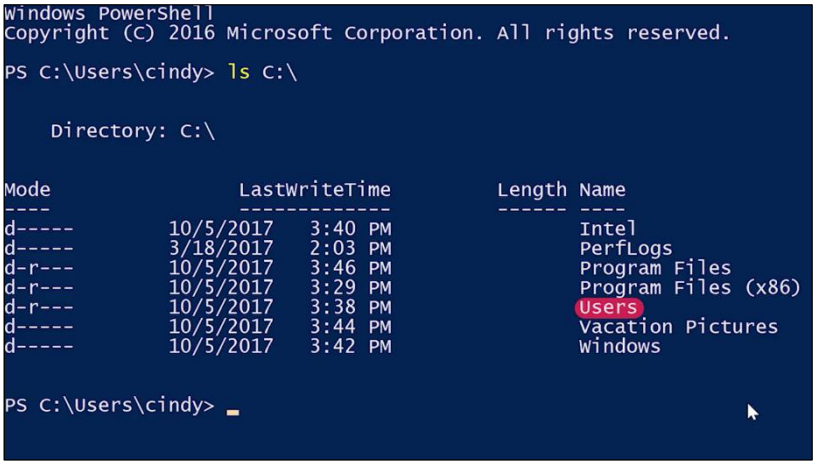
As you continue to work with operating systems, you'll encounter terms that may seem a bit out of place at first but they actually make a lot of sense. Parents and children are common terms that stand for hierarchical relationships in OS's.

If I have a folder named dogs and a second folder nested within that folder called Corgi, dogs would be the parent directory and Corgi would be the child directory.

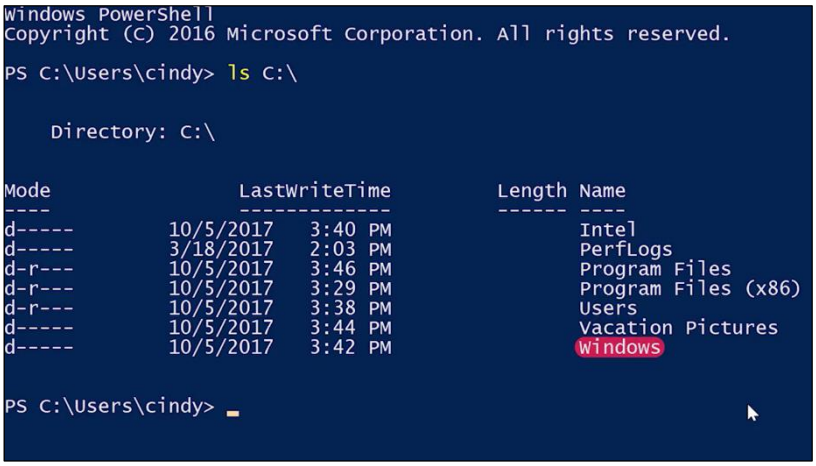
Let's look at a few of the common child directories in this folder. Program Files x86. These directories contain most of the applications and other programs that are installed in Windows



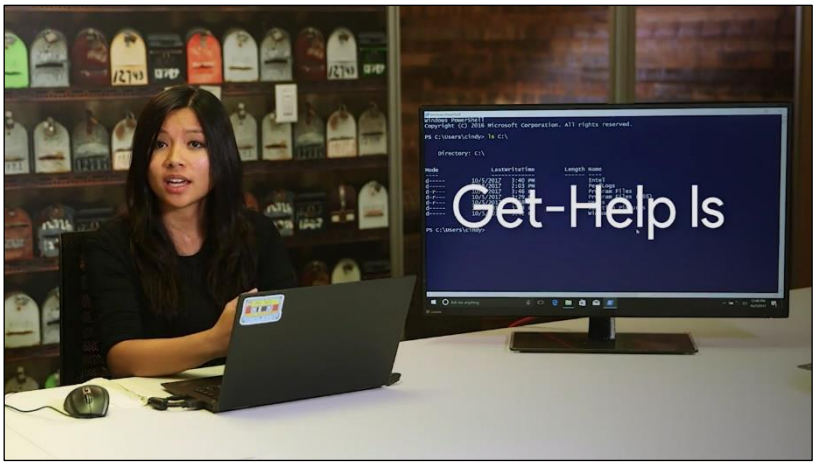
Users. This contains the user profile directories or home directories. Each user who logs into this Windows machine will get their own directory here.

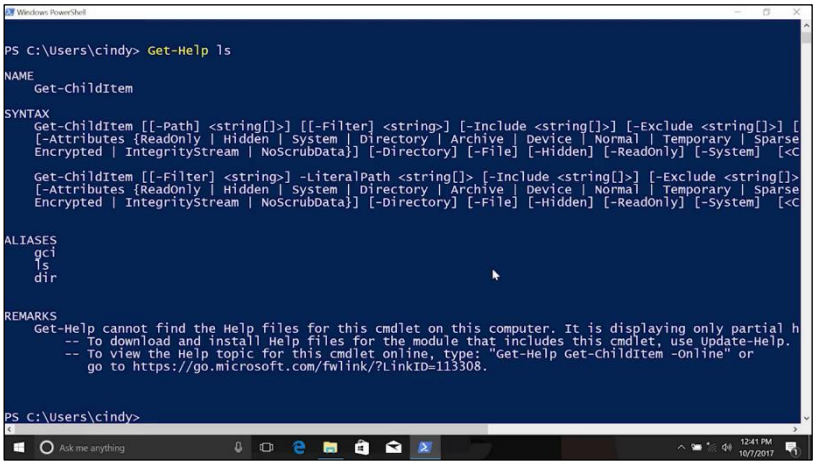


Windows, this is where the Windows operating system is installed.

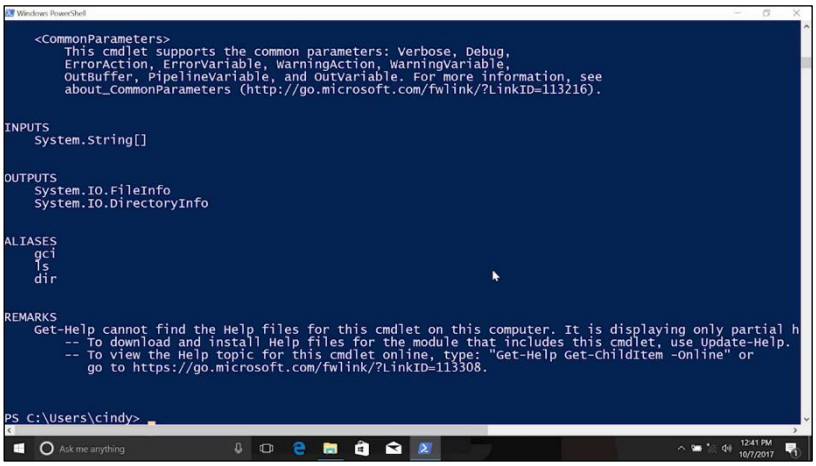


If we open a PowerShell and run Get-Help ls, we'll see the text describing the parameters of the ls command. This will give us a brief summary of the command parameters.



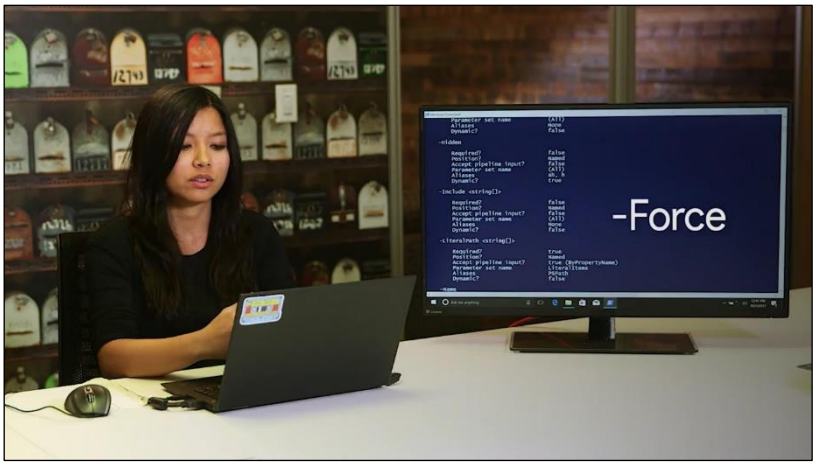


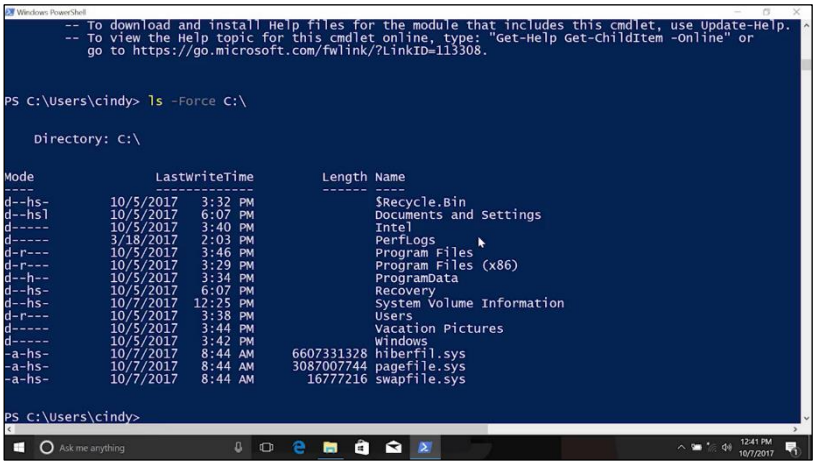
But if you want to see more detailed help, try Get-Help ls -Full.

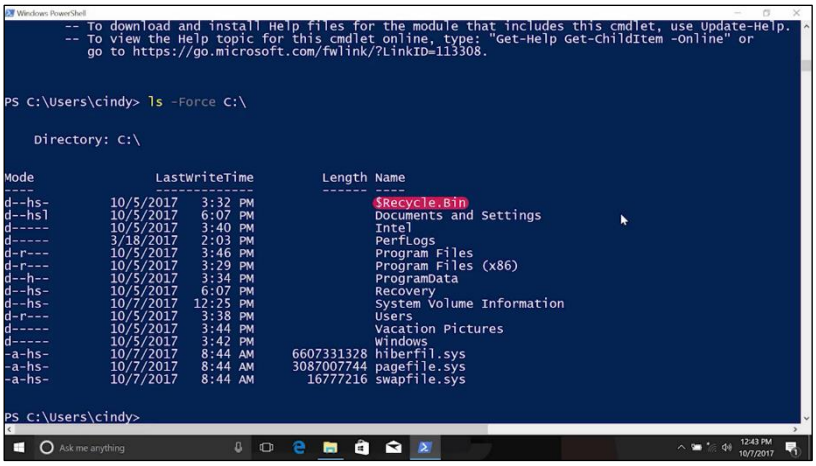


Now you can see a description of each of the parameters and some examples of how to use the command. What if we wanted to see all the hidden files in this directory?

Well, we can use another useful parameter for the ls command, -Force. The -Force parameter will show hidden and system files that aren't normally listed with just ls. Now you can see some important files and directories like Recycle Bin. This is where the Recycle Bin lives.

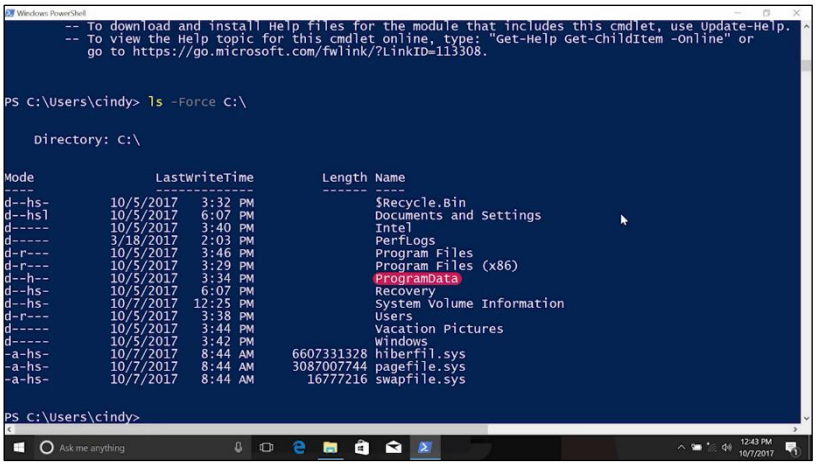






When you move files to the Recycle Bin, they're move to this directory instead of being deleted immediately.

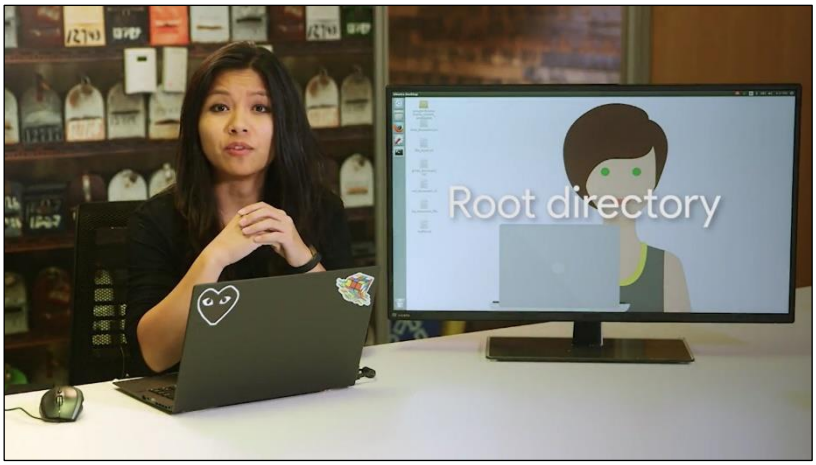
Program data, this directory contains lots of different things. In general, it's used to hold data for programs that are installed in Program Files.



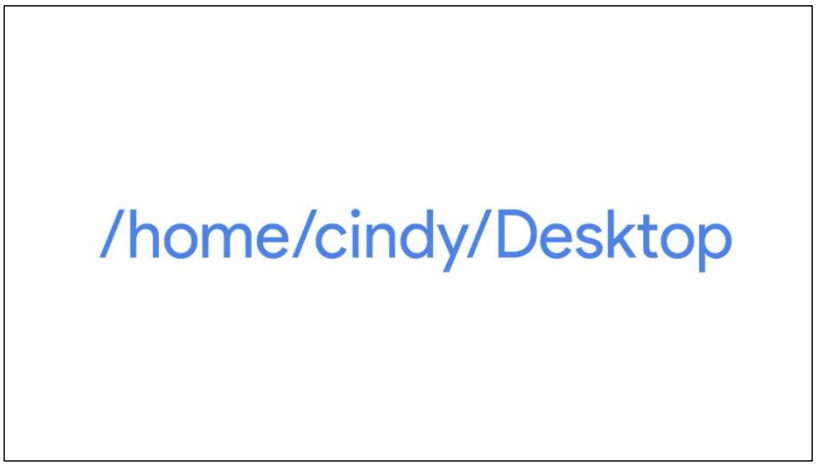
All right, now that you've seen how to take a look around the file system in Windows, lets see what this process looks like in Linux.

1.2.4 Linux list directories

In Linux, the main directory that all other stem from is called the root directory.

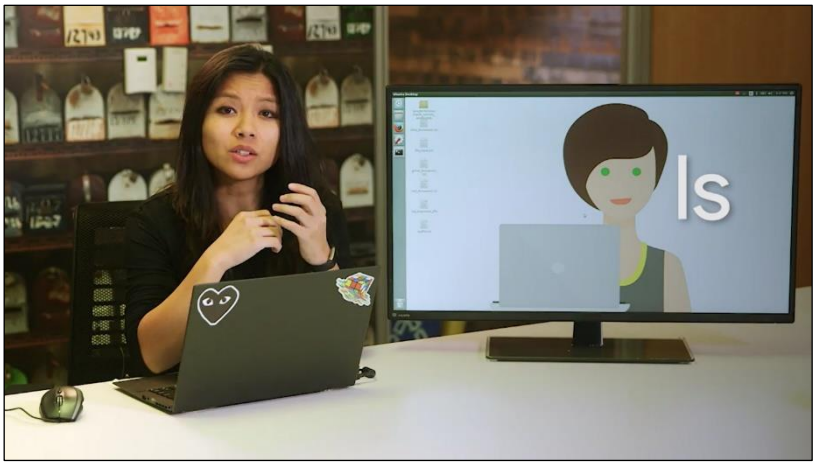


The path to the root directory is denoted by a slash or forward slash. An example of a path in Linux that starts from the root directory is /home/cindy/Desktop.

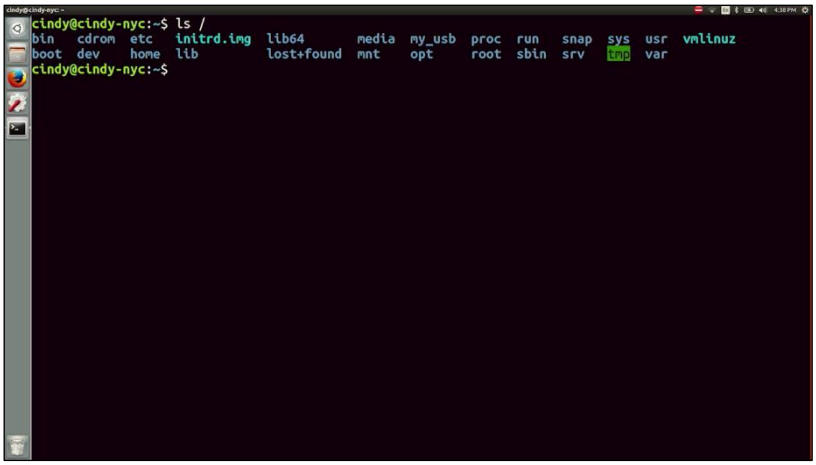


Just like c:\users\cindy\desktop in Windows.

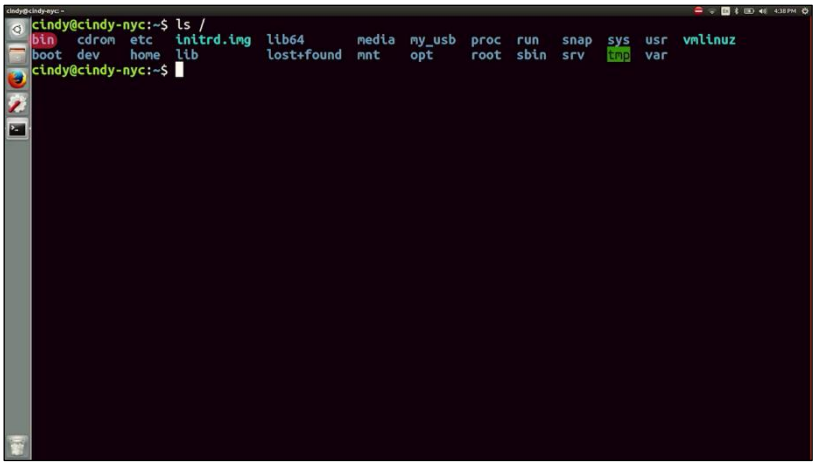
Let's go ahead and see what's under the root directory. We're going to be using the ls or list directory contents command. We also want to give this command, the path, the directory that we want to see. If we don't provide a path, it will just default to the current directory we're in.



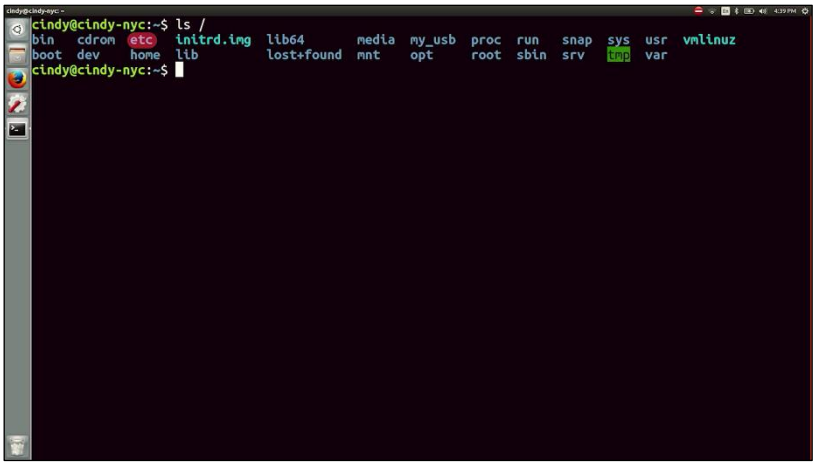
So ls slash. All right, now we can see all the directories that are listed under the root directory. There are a lot of directories here, and they're all used for different purposes. We won't go through them all, but let's talk about a few of the important ones.



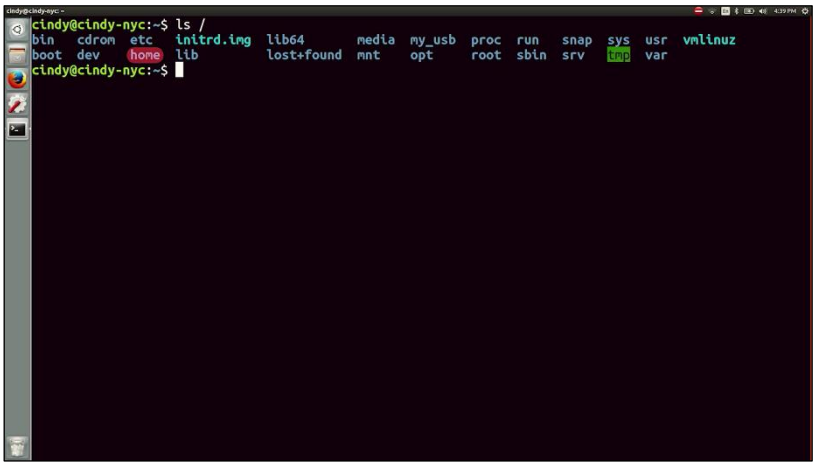
Slash bin, this directory stores our essential binaries or programs. The ls command that we just used is a program, and it's located here in the slash bin folder. It's very similar to our Windows program files directory.



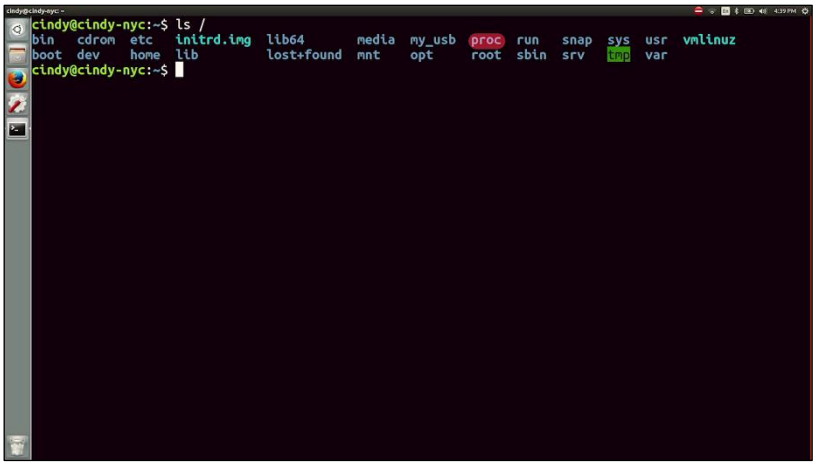
Slash etc, this folder stores some pretty important system configuration files.



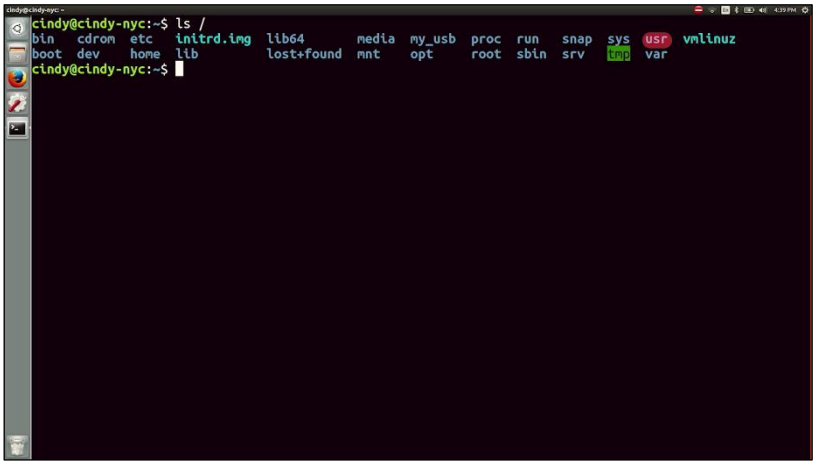
Slash home, this is the personal directory for users. It holds user documents, pictures, and etc. It's also similar to our Windows users directory.



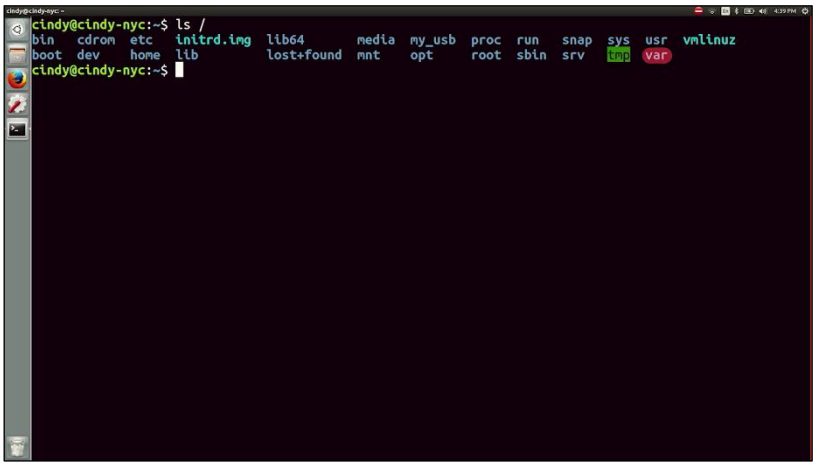
Slash proc, this directory contains information about currently running processes. We'll talk more about processes in an upcoming lesson.



Slash user, the user directory doesn't actually contain our user files like our home directory. It's meant for user installed software.



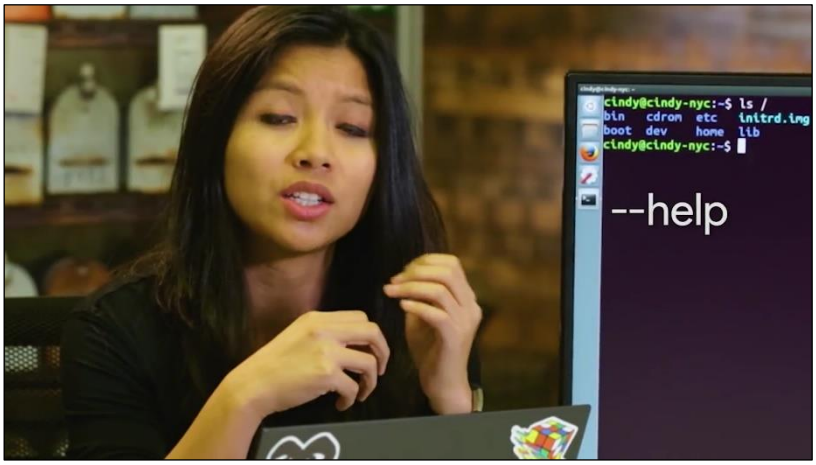
Slash var, we store our system logs and basically any file that constantly changes in here.



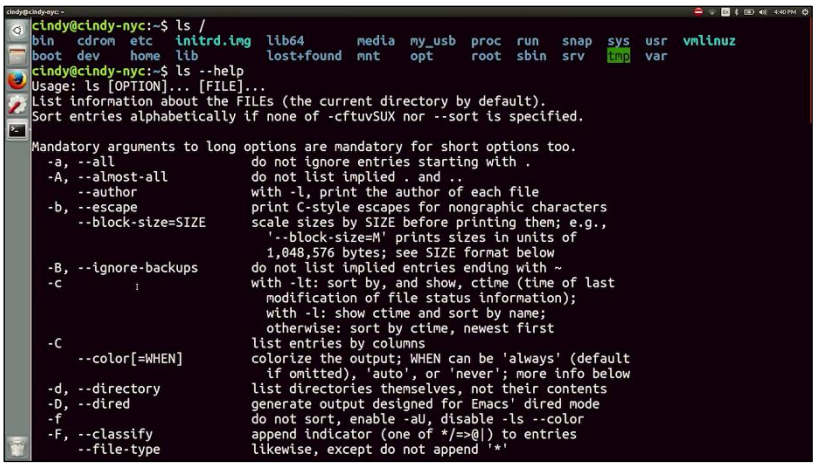
The ls command has a couple of very useful flags that we can use too. Similar to Windows command parameters, a flag is a way to specify additional options for a command. We can usually specify a flag by using a hyphen then the flag option.

Similar to Windows command parameters, a **flag** is a way to specify additional options for a command.

This varies depending on the program, though. Every command has different flag options. You can actually view what options are available for a command by adding the dash, dash help flag.



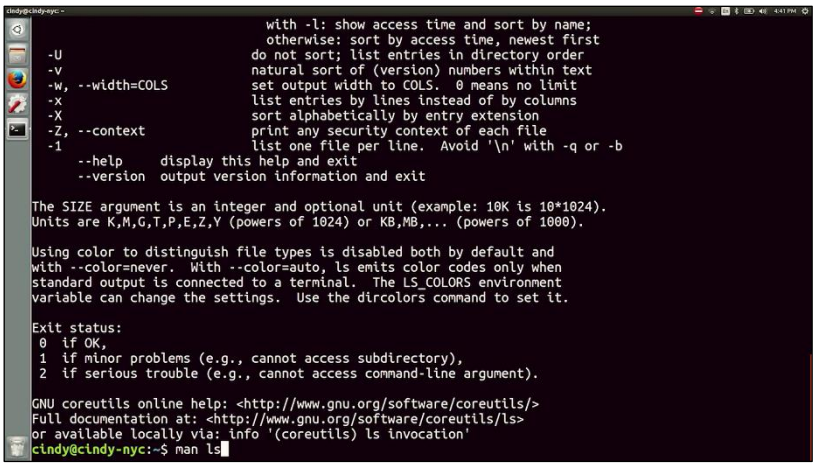
Let's see this in action. There's an incoming wall of text, but don't panic. You don't have to memorize these options. This is mainly used for reference. For now, let's just quickly go through the help menu



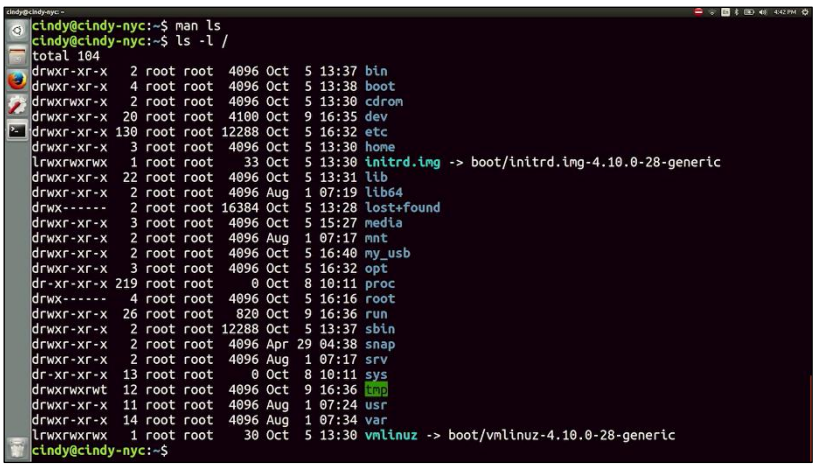
At the top here it tells you what format to put the command in. And here it gives you a description of what the command does. This huge chunk of text lists the options that we can use. It tells us what command flags are available and what they do. The dash, dash help flag is super useful, and even experienced OS users refer to it every so often.

Another method that you can use to get information about commands is the man command from manual. It's used to show us manual pages, in Linux we call them man pages. To use this command, just run man, then the command you want to look up.

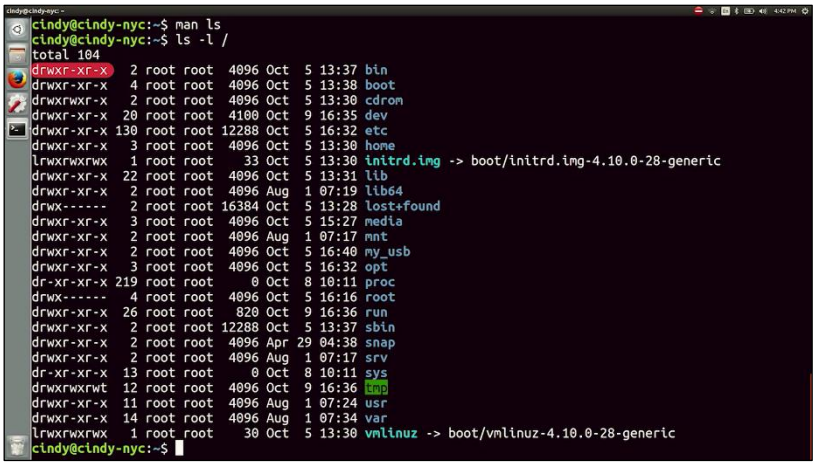
So let's look up man ls. And here we get the same information as dash, dash help, but with a little more detail.



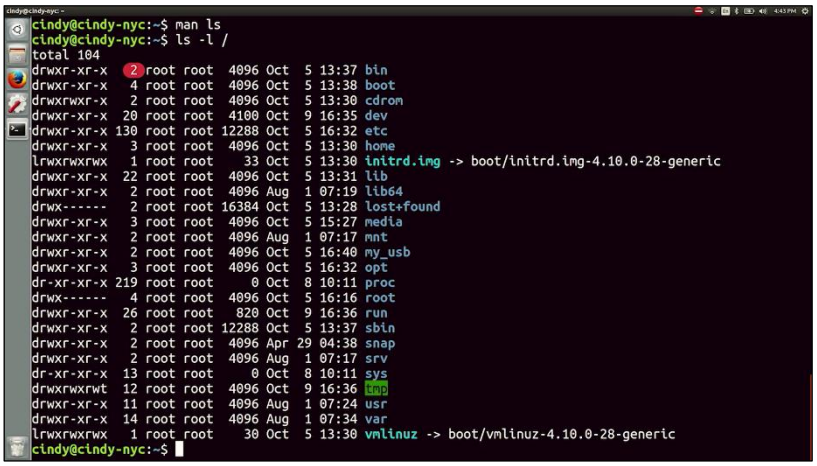
Okay, back to using the ls command. Right now, it's not quite friendly to read. So let's make our directory list more readable with the dash l flag for long. This shows detailed information about files and folders in the format of a long list. Now we can see additional information about our directory and the files and folders in them. Similar to the Windows show properties, the ls command will show us the detailed file information.



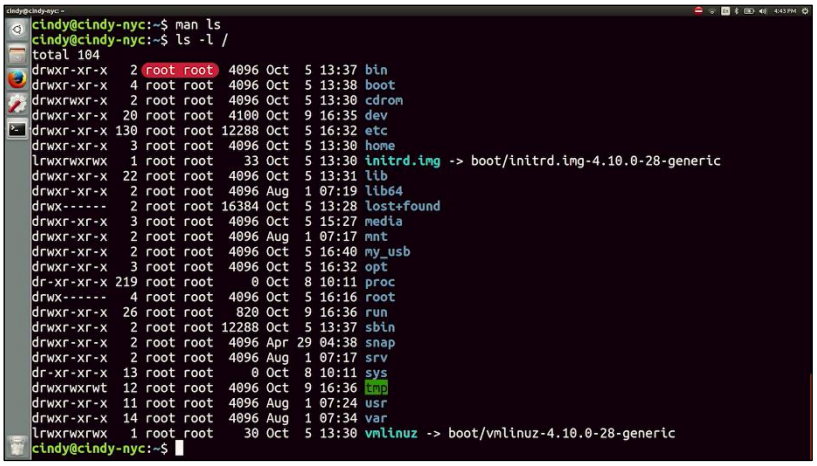
Let's break down this output starting from the left. The first column here are file permissions, side note, we're going to cover file permissions in an upcoming lesson.



Okay, next up is the number of links a file has. Again, we'll discuss this is more detail in a later lesson.



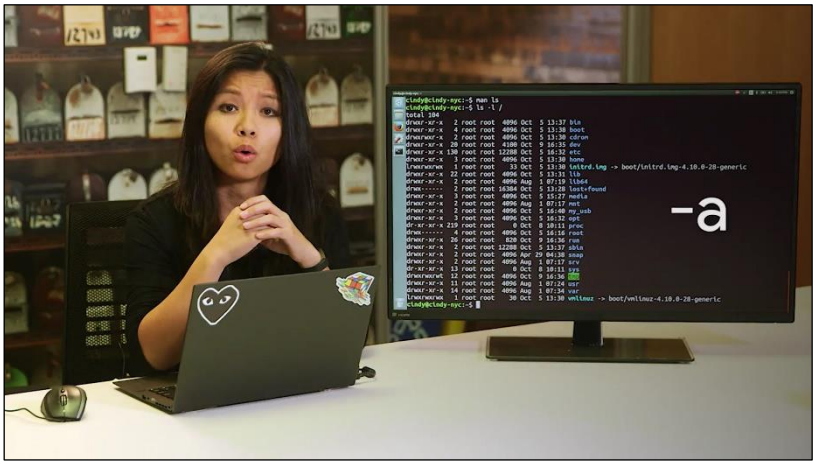
Next, we have the file owner, then the group the file belongs to. Groups are another way we can specify access, we'll talk about this in another lesson too.



So then we have the file size. The time stamp of last modification, and finally, the file or directory name.

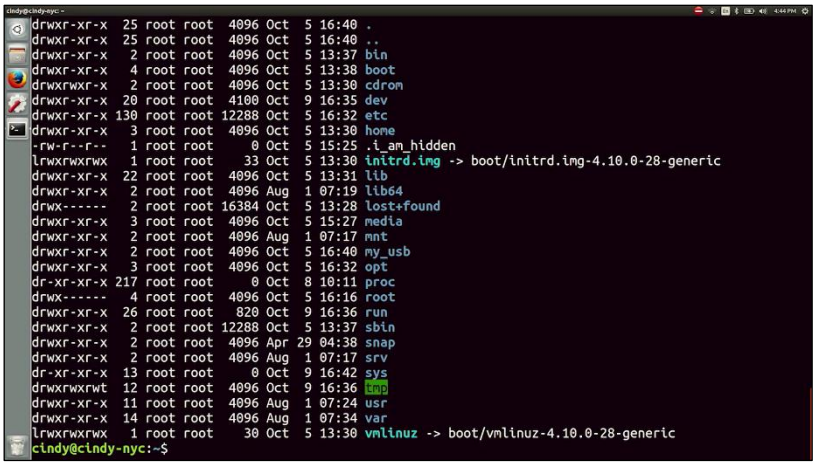
```
cindy@cindy-nyc:~$ man ls
cindy@cindy-nyc:~$ ls -l /
total 104
drwxr-xr-x   2 root root  4096 Oct  5 13:37 bin
drwxr-xr-x   4 root root  4096 Oct  5 13:38 boot
drwxrwxr-x   2 root root  4096 Oct  5 13:30 cdrom
drwxr-xr-x  20 root root  4100 Oct  9 16:35 dev
drwxr-xr-x 130 root root 12288 Oct  5 16:32 etc
drwxr-xr-x   3 root root  4096 Oct  5 13:30 home
lrwxrwxrwx   1 root root    33 Oct  5 13:30 initrd.img -> boot/initrd.img-4.10.0-28-generic
drwxr-xr-x  22 root root  4096 Oct  5 13:31 lib
drwxr-xr-x   2 root root  4096 Aug  1 07:19 lib64
drwx------   2 root root 16384 Oct  5 13:28 lost+found
drwxr-xr-x   3 root root  4096 Oct  5 15:27 media
drwxr-xr-x   2 root root  4096 Aug  1 07:17 mnt
drwxr-xr-x   2 root root  4096 Oct  5 16:40 my_usb
drwxr-xr-x   3 root root  4096 Oct  5 16:32 opt
dr-xr-xr-x 219 root root     0 Oct  8 10:11 proc
drwx------   4 root root  4096 Oct  5 16:16 root
drwxr-xr-x  26 root root   820 Oct  9 16:36 run
drwxr-xr-x   2 root root 12288 Oct  5 13:37 sbin
drwxr-xr-x   2 root root  4096 Apr 29 04:38 snap
drwxr-xr-x   2 root root  4096 Aug  1 07:17 srv
dr-xr-xr-x  13 root root     0 Oct  8 10:11 sys
drwxrwxrwt  12 root root  4096 Oct  9 16:36 tmp
drwxr-xr-x  11 root root  4096 Aug  1 07:24 usr
drwxr-xr-x  14 root root  4096 Aug  1 07:34 var
lrwxrwxrwx   1 root root    30 Oct  5 13:30 vmlinuz -> boot/vmlinuz-4.10.0-28-generic
cindy@cindy-nyc:~$
```

The last flag that we'll discuss for the ls command is the dash a or all option. This shows us all the files in the directory including the hidden files.



You'll notice that I appended two different flags together. This is the same thing as ls -l -a /. Both work the exact same way. The order of the flag determines which order it goes in. In our case, it doesn't matter if we do a long list first or show all files first.

```
cindy@cindy-nyc:~$ man ls
cindy@cindy-nyc:~$ ls -l /
total 104
drwxr-xr-x   2 root root  4096 Oct  5 13:37 bin
drwxr-xr-x   4 root root  4096 Oct  5 13:38 boot
drwxrwxr-x   2 root root  4096 Oct  5 13:30 cdrom
drwxr-xr-x  20 root root  4100 Oct  9 16:35 dev
drwxr-xr-x 130 root root 12288 Oct  5 16:32 etc
drwxr-xr-x   3 root root  4096 Oct  5 13:30 home
lrwxrwxrwx   1 root root    33 Oct  5 13:30 initrd.img -> boot/initrd.img-4.10.0-28-generic
drwxr-xr-x  22 root root  4096 Oct  5 13:31 lib
drwxr-xr-x   2 root root  4096 Aug  1 07:19 lib64
drwx------   2 root root 16384 Oct  5 13:28 lost+found
drwxr-xr-x   3 root root  4096 Oct  5 15:27 media
drwxr-xr-x   2 root root  4096 Aug  1 07:17 mnt
drwxr-xr-x   2 root root  4096 Oct  5 16:40 my_usb
drwxr-xr-x   3 root root  4096 Oct  5 16:32 opt
dr-xr-xr-x 219 root root     0 Oct  8 10:11 proc
drwx------   4 root root  4096 Oct  5 16:16 root
drwxr-xr-x  26 root root   820 Oct  9 16:36 run
drwxr-xr-x   2 root root 12288 Oct  5 13:37 sbin
drwxr-xr-x   2 root root  4096 Apr 29 04:38 snap
drwxr-xr-x   2 root root  4096 Aug  1 07:17 srv
dr-xr-xr-x  13 root root     0 Oct  8 10:11 sys
drwxrwxrwt  12 root root  4096 Oct  9 16:36 tmp
drwxr-xr-x  11 root root  4096 Aug  1 07:24 usr
drwxr-xr-x  14 root root  4096 Aug  1 07:34 var
lrwxrwxrwx   1 root root    30 Oct  5 13:30 vmlinuz -> boot/vmlinuz-4.10.0-28-generic
cindy@cindy-nyc:~$ ls -la /
```

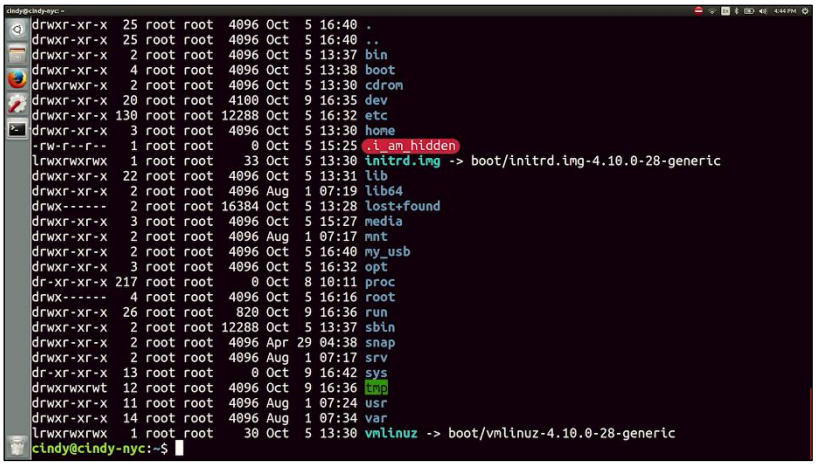


This is the same thing as: ls -l -a /.
Both work the exact same way.

Check out how there are some new files are visible when we use these flags. The dash a or all flag, shows all files including hidden ones.

The **-a** or **all** flag shows all files including hidden ones.

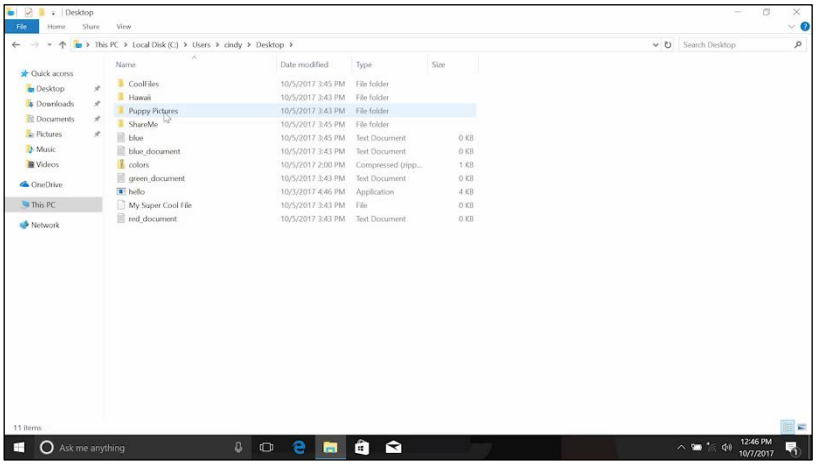
You can hide a file or directory by pre-pending a dot to it. Like the file shown here .I_am_hidden.



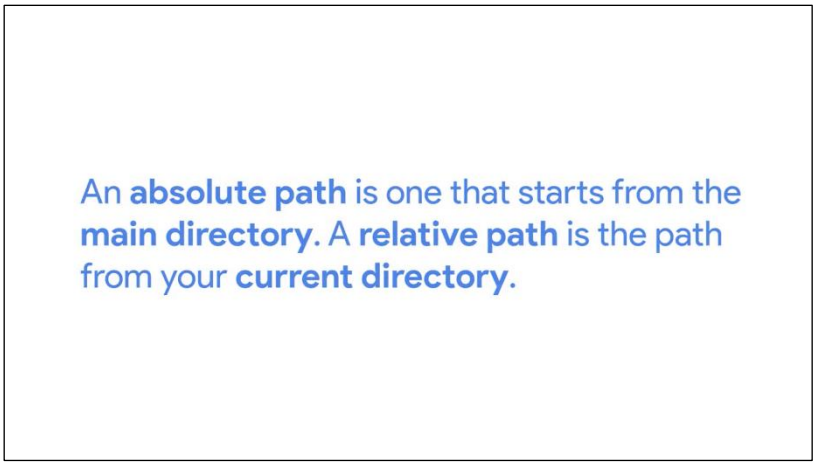
We've covered a lot in this video, we've learned how to view detailed information about files with the ls command. We also started using computer paths and we learned how to get help with commands using the dash dash help flag and man pages. We even took a sneak peek at our Linux files system. If I went through any of this a little too quickly, just rewatch the video. We'll meet back up in the next one, where we'll start changing directories in the GUI. See you there.

1.2.5 Windows changing directories in the GUI

Okay. Now that we know how directories are laid out, let's start moving from one directory to the next. You probably change directories in your GUI a lot without even realizing it. Even if that's not the case, we're going to go ahead and show you how to do it. Knowledge is power.



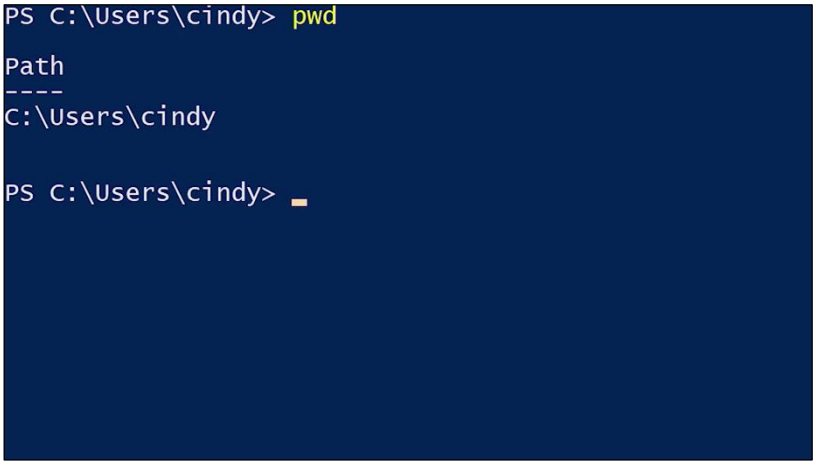
There, that was pretty simple, right? We can move freely between any directory in any path on our systems. One thing to call out is that there are two different types of paths, absolute and relative. An absolute path is one that starts from the main directory. A relative path is the path from your current directory.



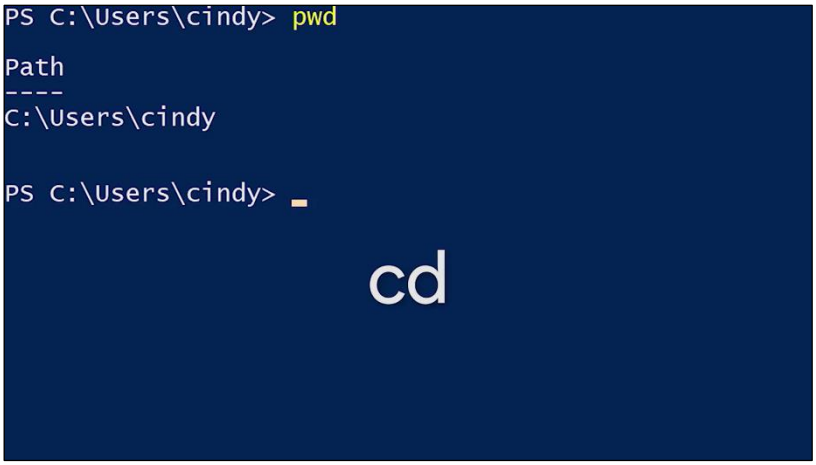
These two distinctions aren't as important when we're working in a GUI, but they're important when you work in a shell. So let's see what this looks like in the Windows CLI.

1.2.6 Windows changing directories in the CLI

When you first open PowerShell, you'll usually be in your home directory. Your prompt shows you which directory you're currently in, but there's also a command that will tell you where you are. PWD or print-working directory tells you which directory you're currently in.



If we want to change the directory that we're in, we can use the CD or change directory command.

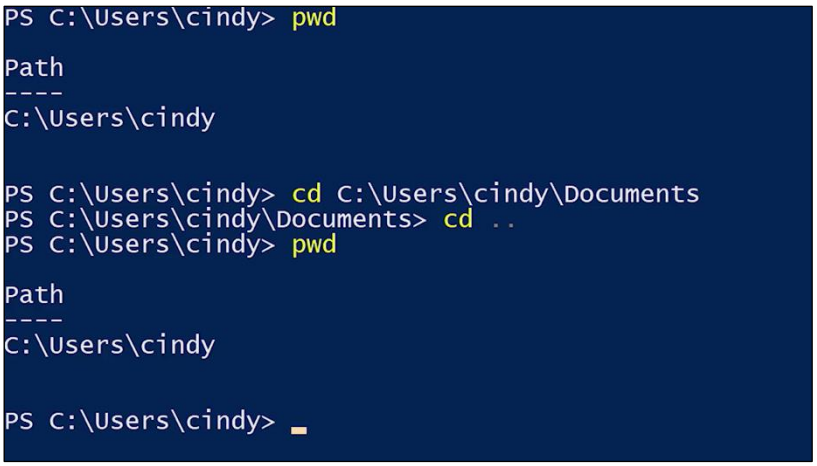


To use this command, we'll also need to specify the path that we want to change to. Remember, this path can be absolute, which means it starts from this drive letter and spells out the entire path.

On the flip side, it can be relative, meaning, that we only use part of the path to describe how to get to where we want to go relative to where we're currently are. I'll show you what I mean in a minute. So right now, we're in C:\Users\cindy. Let's say that instead, I want to go to C:\Users\cindy\documents, what do you think the command would look like here? Here it is, cd C:\Users\cindy\documents. And now we've changed to the documents directory. We use an absolute path to get to this directory, but this can be a little cumbersome to type out.

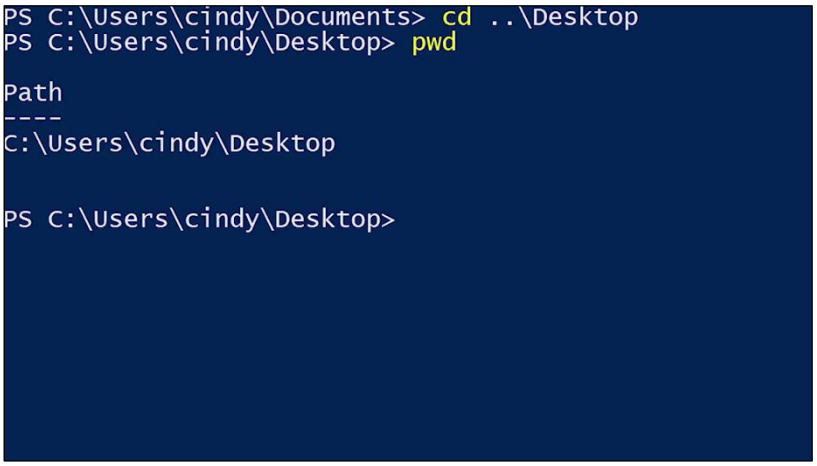
We know that that documents directory is under the cindy folder, so can't we just go up one level to get to that folder? We absolutely can. There's a shortcut to get to the level above your current directory, CD dot dot.

Let's run the PWD command one more time. Now, we can see that I'm in C:\users\cindy, the parent directory of where I was before.

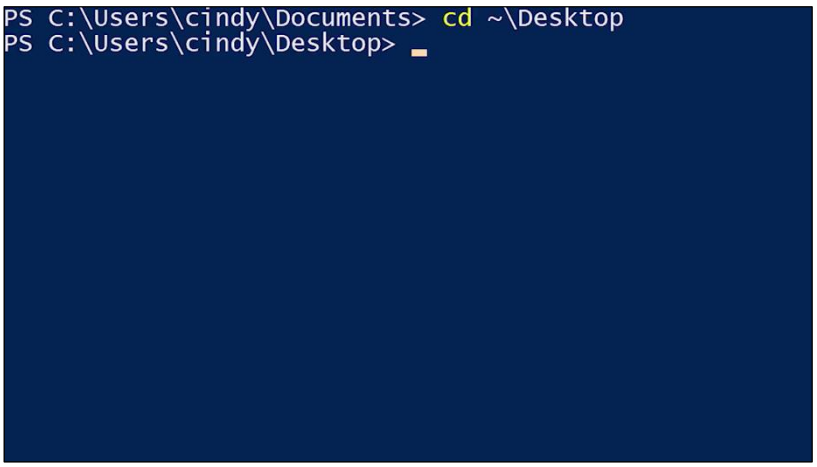


The dot dot is considered a relative path because it'll take you up one level relative to where you are. Let's go back to the documents folder and try this again, except this time, let's go to the desktop folder using the new command we learned.

We know that the desktop and document directories are under the home directory, so we could run CD dot dot then CD desktop, but there is actually an easier way to write this, cd..\Desktop. Let's check PWD one more time. PWD now shows that were in the Desktop folder. Sweet.



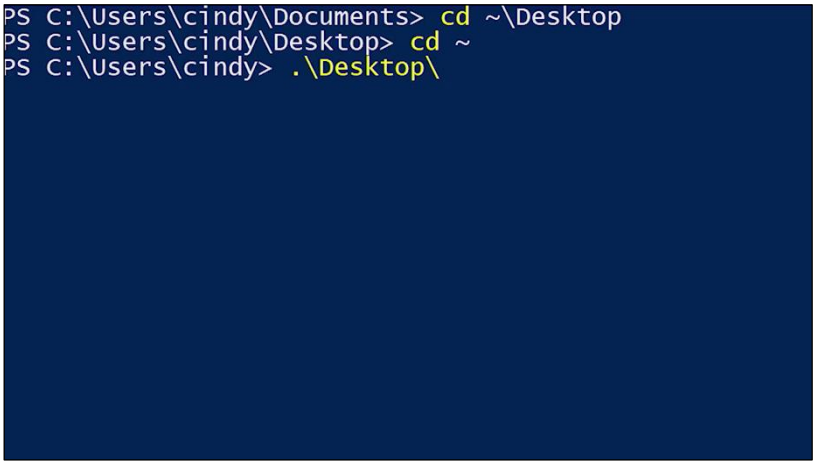
Another cool shortcut for CD that you can use is CD~. The tilde is a shortcut for the path of your home directory. Let's say I want to get to the desktop directory in my home folder. I can do something like this, cd~\Desktop.

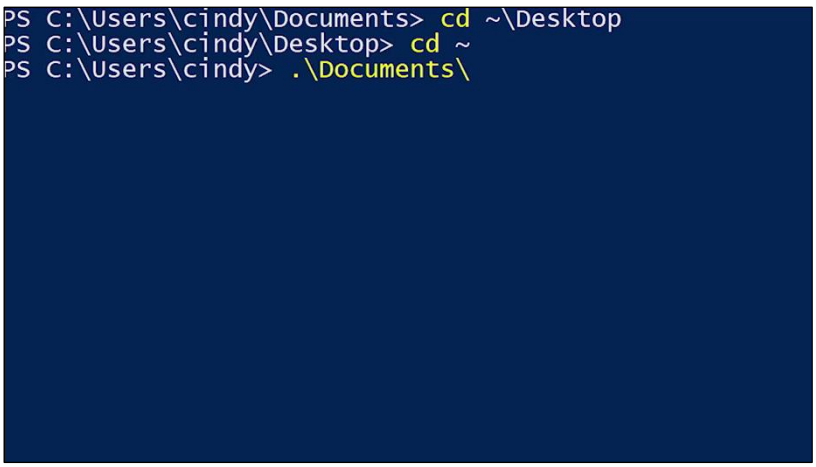


We've done quite a bit of typing so far, you might actually be wondering, what would happen if we messed up while typing these directory names? How are we supposed to memorize where everything is, and if it's spelled correctly? Fortunately, we don't have to do that. Our shell has a built-in feature called tab completion.

Tab completion lets us use the tab key to auto-complete file names and directories. Let's use the tab completion to get to our desktop from our home directory, if I type D and then tab, the first file or directory starting with D will now complete.

Now, if this isn't the file or directory that I was looking for, I can continue to press tab, and the path will rotate through all the options that complete the name that I started to type. So I'll see desktop, and then documents, and then downloads.





```
PS C:\Users\cindy\Documents> cd ~\Desktop
PS C:\Users\cindy\Desktop> cd ~
PS C:\Users\cindy> .\Downloads\
```

Take note, that the dot in front of the path of.\Desktop just means the current directory. If I erased this and instead type DE then the only directory that matches is desktop.

Tab completion is an awesome feature that you'll be using more and more as you continue to work with commands.

1.2.7 Linux changing directories in bash

Let's do the same thing in Bash. From our desktop we're going to navigate to the documents folder. The commands we used earlier in PowerShell are exactly the same here in bash. Print working directory or PWD again shows us the current path we're in.

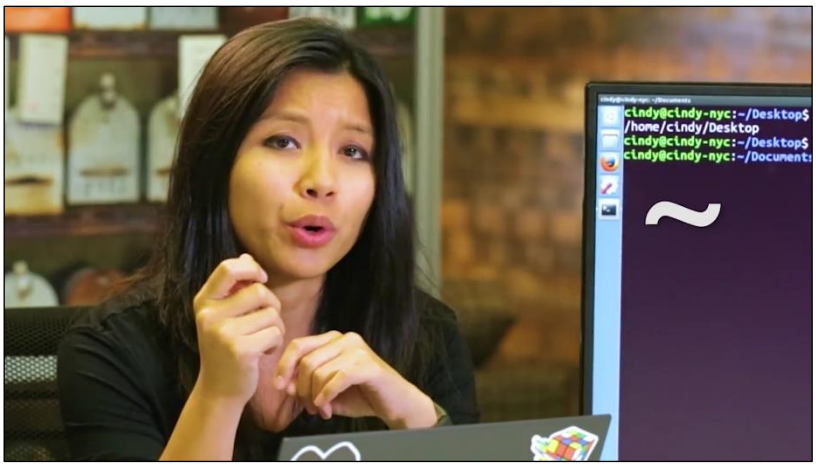
Yep, looks good. We're currently in our desktop directory, which you can see from /home/cindy/Desktop.

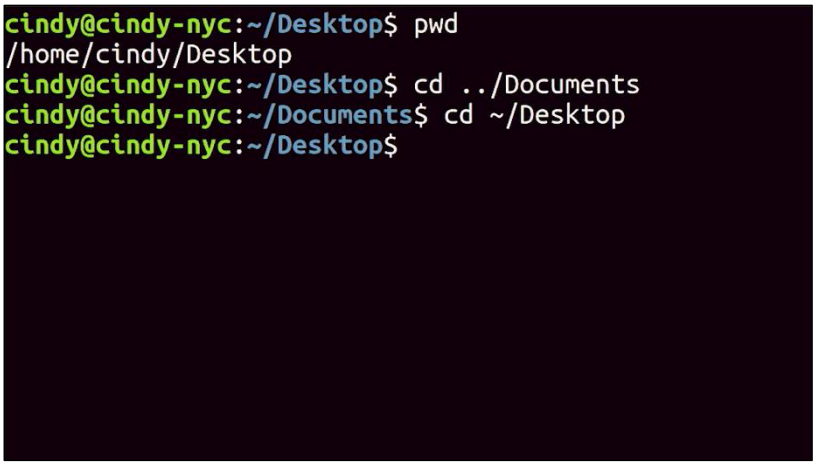
```
cindy@cindy-nyc:~/Desktop$ pwd
/home/cindy/Desktop
cindy@cindy-nyc:~/Desktop$
```

To navigate around, we use the CD command just like with Windows. We can give it an absolute path like this cd/home/cindy/Documents, or we can give it a relative path like this cd../Documents.

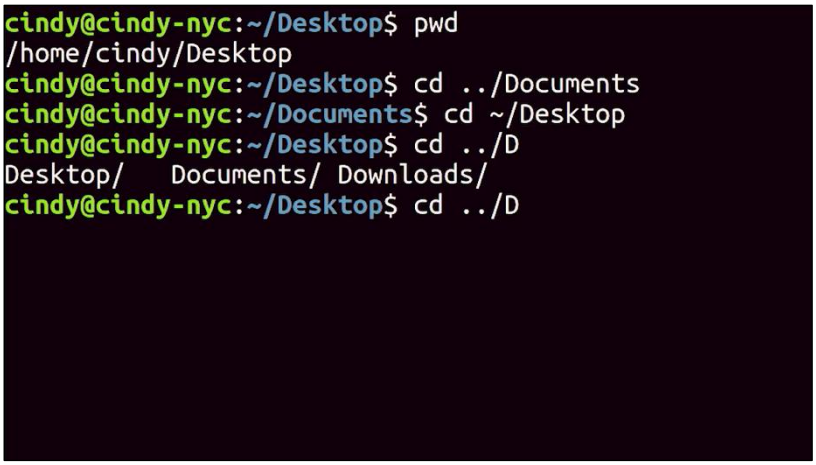
```
cindy@cindy-nyc:~/Desktop$ pwd
/home/cindy/Desktop
cindy@cindy-nyc:~/Desktop$ cd ../Documents
cindy@cindy-nyc:~/Documents$
```

In Bash, the tilde is used to reference our home directory. So, cd~/Desktop will take us back to our desktop, and guess what?





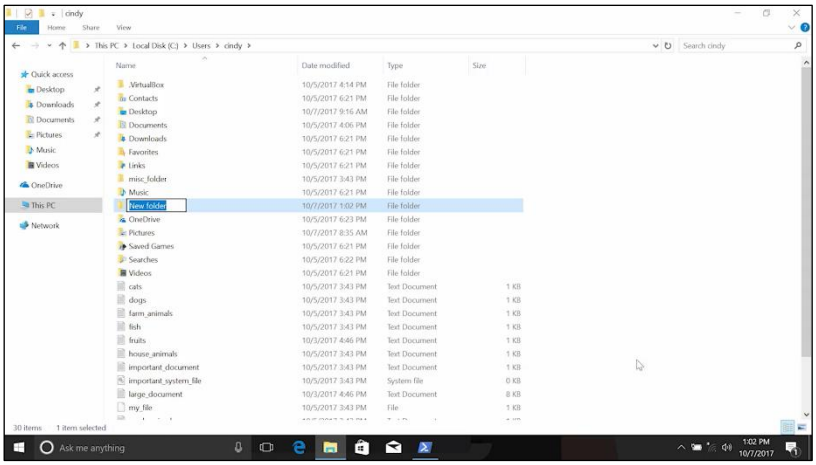
We still have that useful tab completion feature in Bash. The difference between Bash tab complete and Windows tab complete is that if we have multiple options, it won't rotate through the options, but instead will show us all options at once like this.



We can already start connecting the bridge between Windows and Linux.

1.2.8 Windows make directories in the GUI-CLI

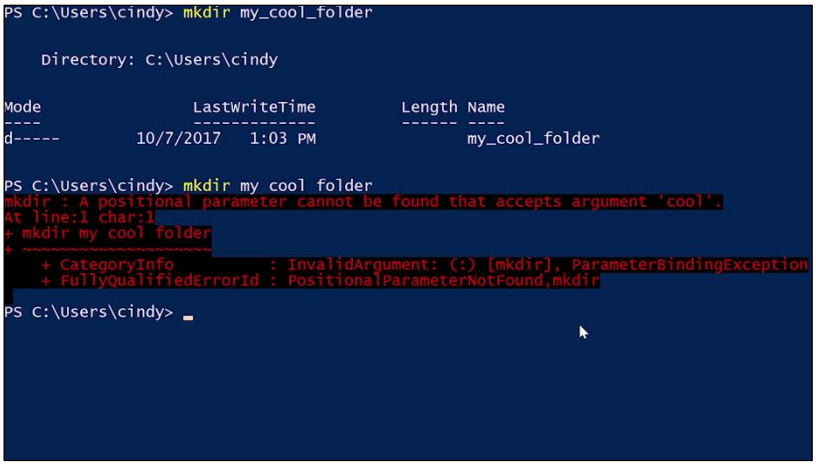
Now that we've covered listing and changing directories, let's learn how to add new directories. We can do this in the GUI in a super simple way. Just right-click, new, then folder, and bam, we have a new folder.



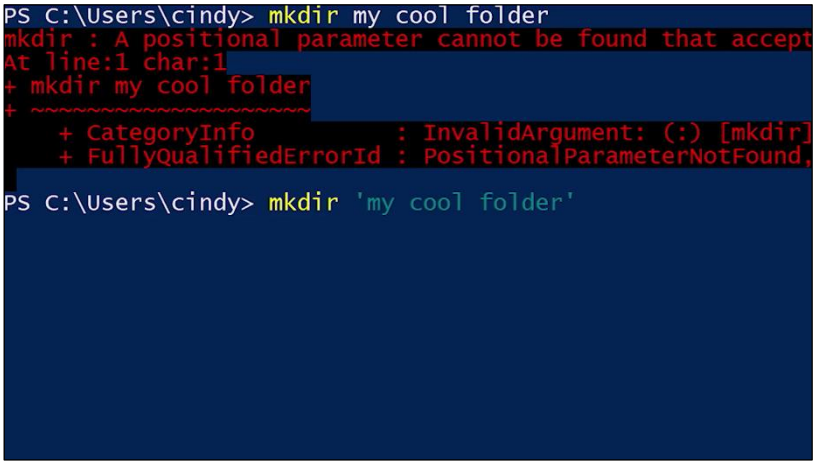
Now, what if we wanted to do this in the CLI? In PowerShell, the command to make a new directory is called mkdir or make directory.

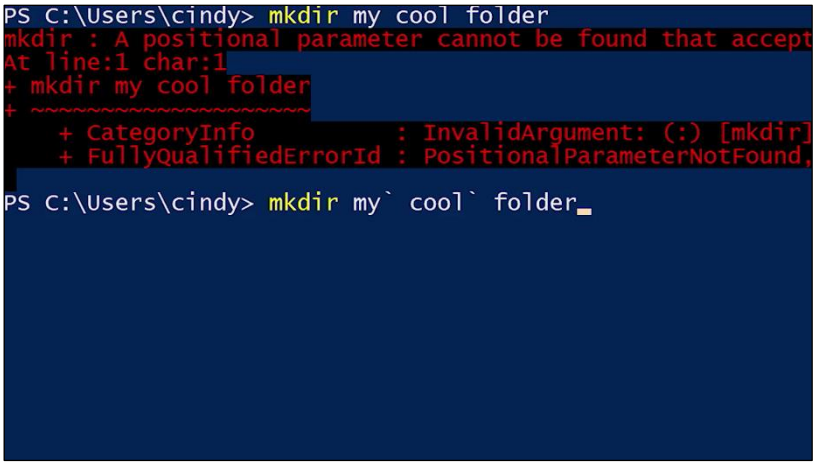

```
PS C:\Users\cindy> mkdir my_cool_folder


    Directory: C:\Users\cindy

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        10/7/2017   1:03 PM                my_cool_folder

PS C:\Users\cindy> _
```

Let's make a new directory called my_cool_folder and there it is. That was easy. What if we wanted to use spaces in our folder name instead of underscores? What do you think would happen if I did this instead? Mkdir my cool folder. That's an error. Mkdir is trying to interpret cool and folder as other parameters to the mkdir command. It doesn't understand those words as valid parameters.

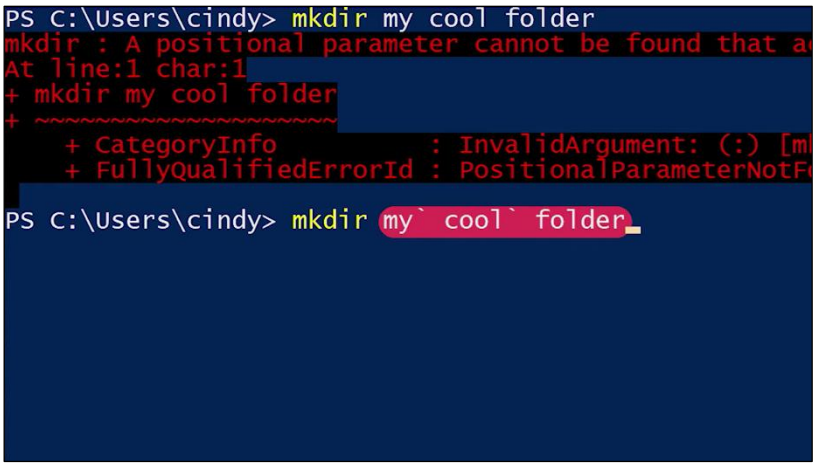


Turns out that our shell doesn't interpret spaces the way we do. So, we need to tell it explicitly that this folder name is one single thing. We can do this in a variety of ways. We can surround the name with quotes like, mkdir 'my cool folder', or we can escape to space by using the back tick character, mkdir my` cool` folder.





Escaping characters is a pretty common concept when dealing with code. It means that the next character after the back tick should be treated literally. In our example, escaping the space tells the shell that the space after the back tick is part of our filename.

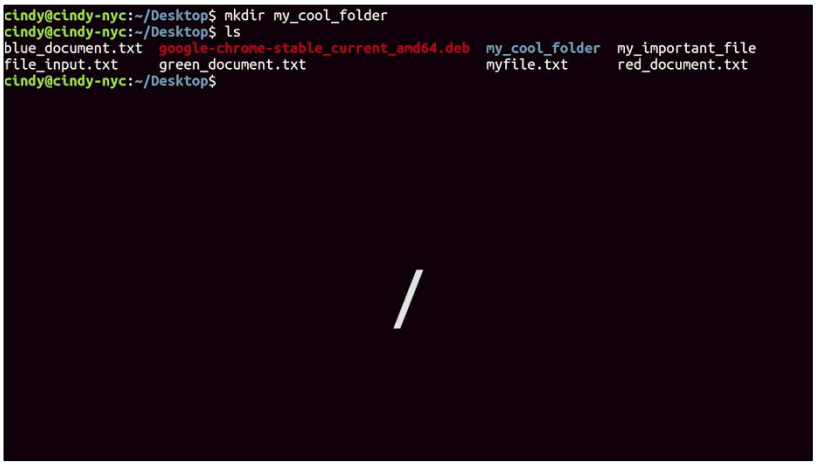


While the back tick is the escape character in PowerShell, other shells and programming languages may use another character as an escape character. You'll see this in the next video.

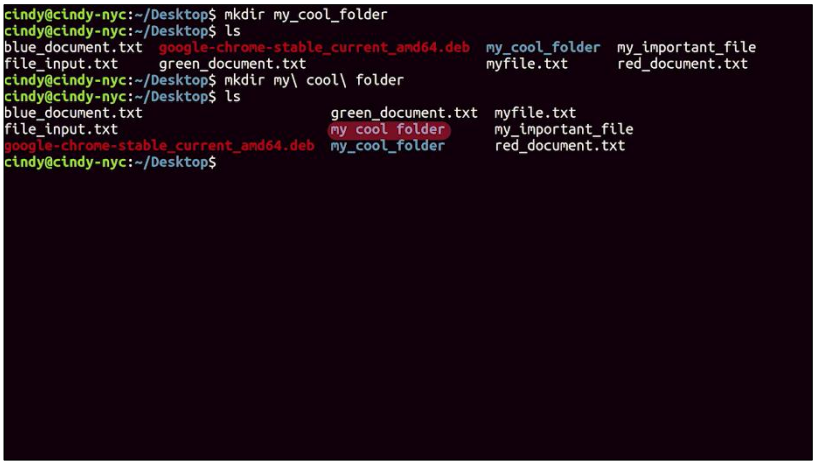
1.2.9 Linux make directories in bash

In Bash, the command to make a new directory is the same as in Windows. Let's make a new directory called my cool folder with the mkdir or make directory command. And now, we can verify my cool folder is in our desktop.

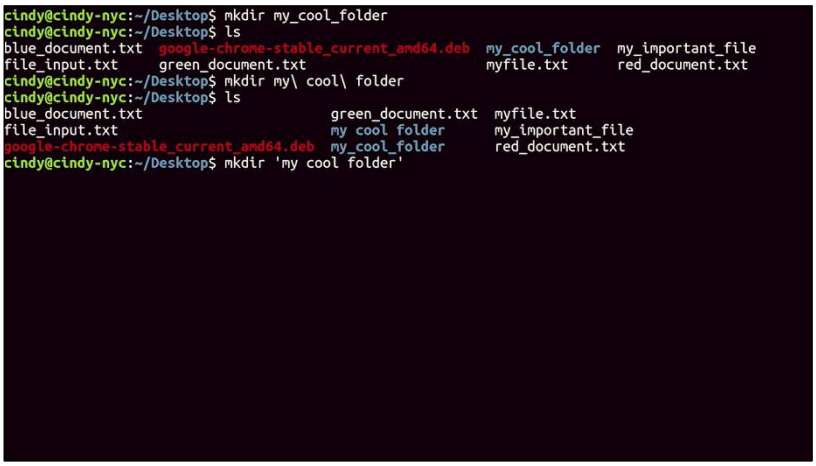
Instead of using back ticks like in windows to escape a character, in Bash, you can use a backslash. Similar to Windows, you can also use quotes to encompass an entire file name.



How do you think you would make a directory called my cool folder in Linux with spaces? mkdir my\ cool\ folder. There it is.



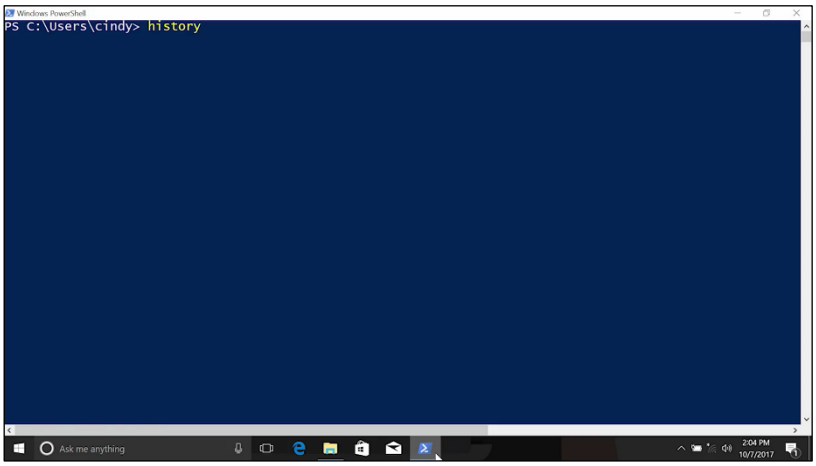
Or, mkdir ' my cool folder'. Works as well.

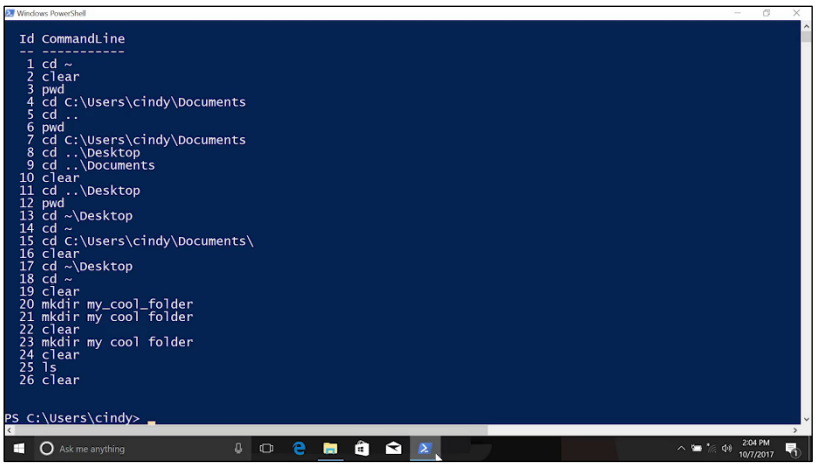


If you guessed this, you're right. If you guessed wrong, that's okay. Just re-watch this video so you can get a better grasp of how we came to this conclusion.

1.2.10 Windows command history

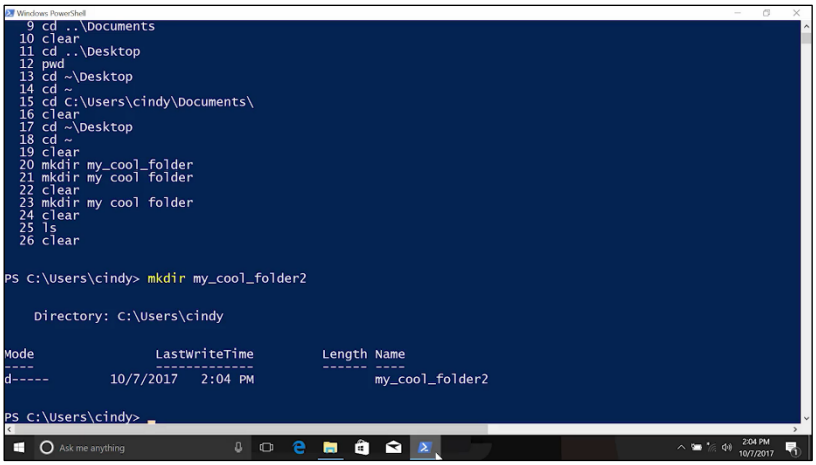
Picking right up from the last video, let's say we want to make a couple of directories, my_cool_folder2 and my_cool_folder3. We could just type mkdir my_cool_folder2, and then type again mkdir my_cool_folder3, but instead we're going to use another cool PowerShell feature called history.



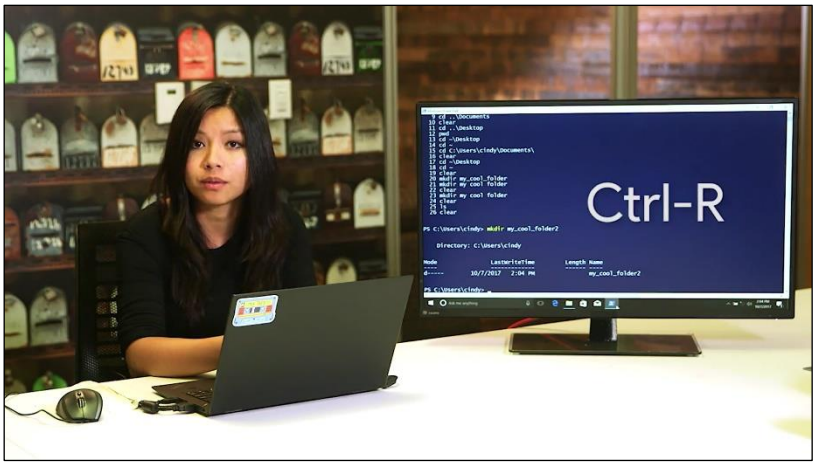


Each and every time you enter in a command, it gets saved into memory and added to a special file. You can go through the previous commands you used with the history command. I'm now showing a list of commands that I entered earlier. This information alone isn't very useful.

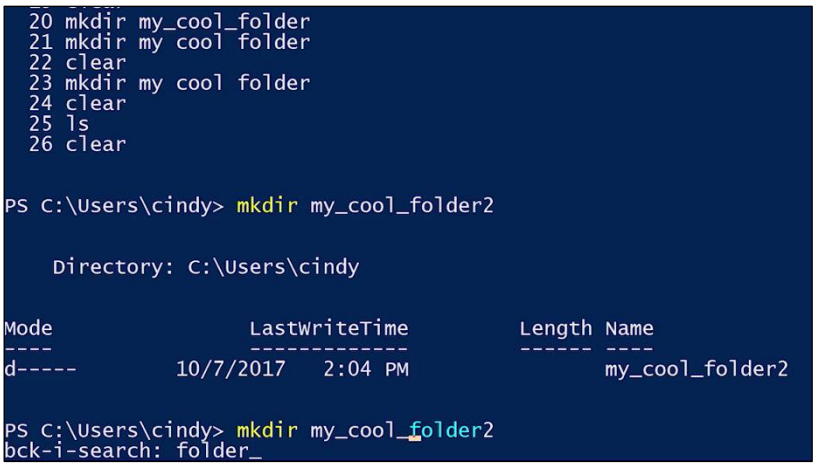
Instead, there's a better use of the history that lets us quickly scroll through these commands and use them again. We can scroll through these commands with the up or down keys on our keyboard. I'm going to go up to my previous command, and I should see that I have mkdir my_cool_folder. Instead of typing the whole thing to make a new folder, I'm just going to append the number 2 to my command. And boom, a new file is created without having to type everything over again. Cool, right?



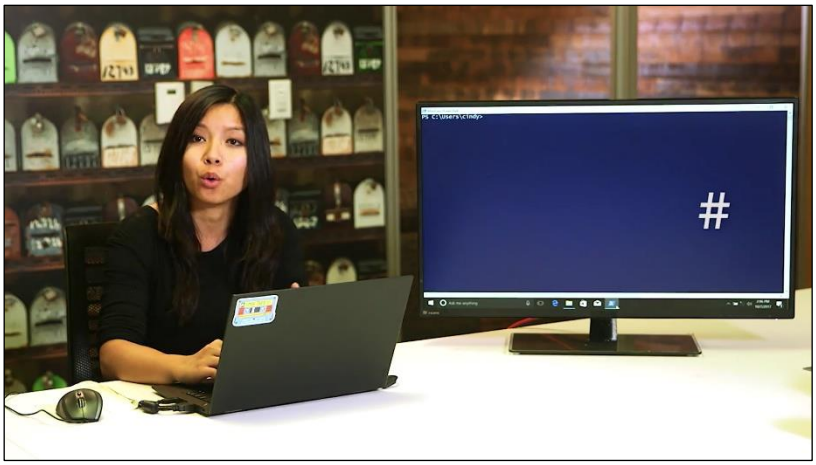
You can even search through your previously used commands using the history shortcut Ctrl+R.



From here you can start typing bits and pieces of the command you want to look for, and it'll show you matches. Let's search for the word folder. I should see the mkdir commands I was using before. Pretty neat.



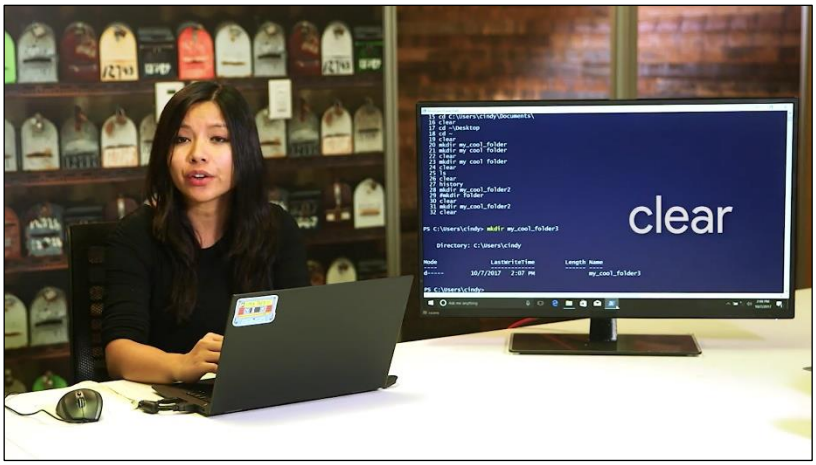
If you're using an older version of PowerShell, it may not have the Ctrl+R feature. If that's the case you can type the # symbol followed by some part of your old command, and then use Tab completion to cycle through the items in your history.




```
PS C:\Users\cindy> #mkdir
```

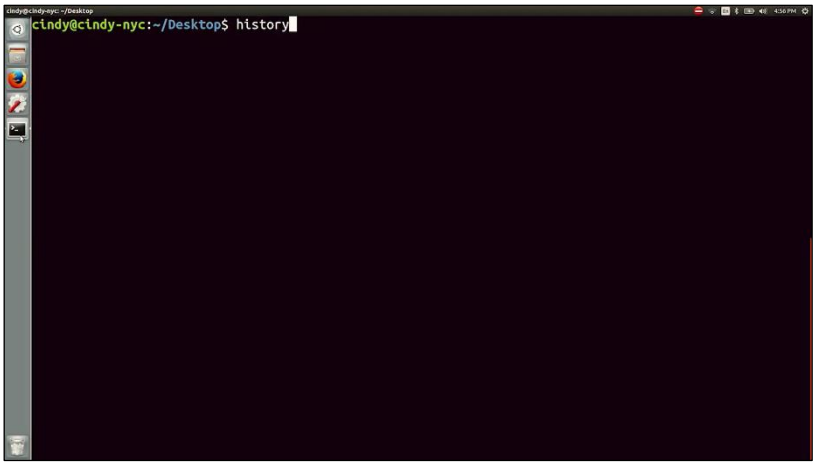
```
PS C:\Users\cindy> mkdir my cool folder
```

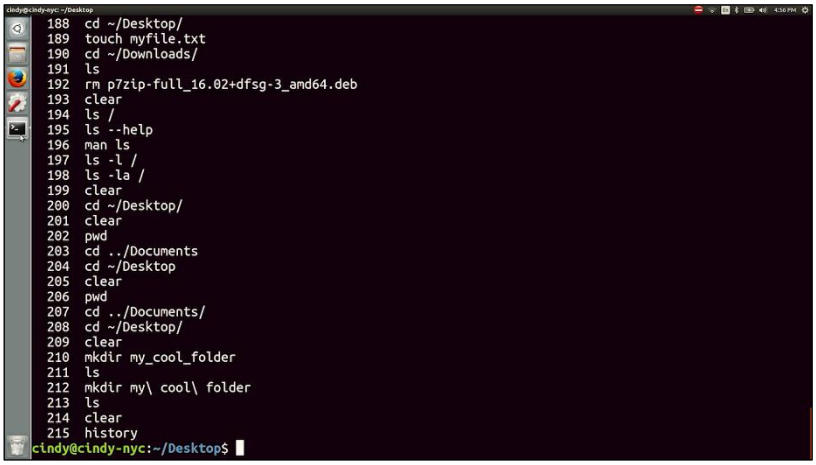
The history feature, along with Tab completion and get-help, will be your best friends while you work in PowerShell. Keep them close to you and get to know them super well. Hmm, our shell is looking a little cluttered. It's kind of hard to see where I'm at, so let's clean up our shell a little bit. We can do that with the clear command. This doesn't wipe your history, it just clears the output on your screen. It looks a little better.



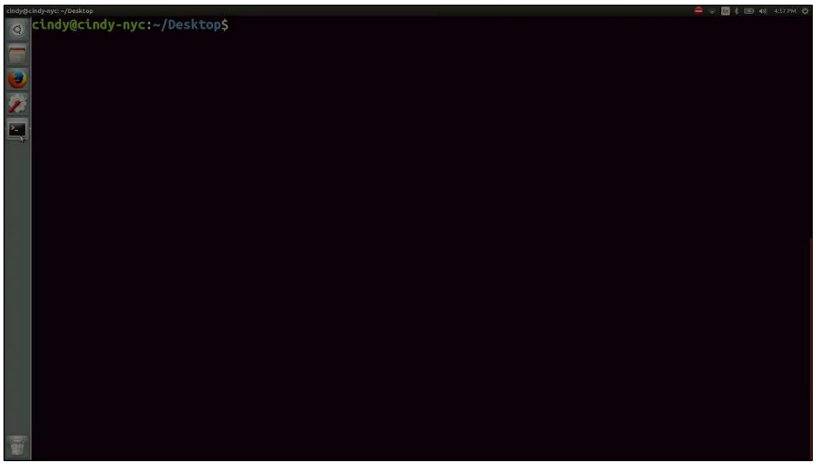
1.2.11 Linux command history

The exact same history command that's used in Windows is used in Linux. From here, we can use our up and down keys and even search through our history with Ctrl-R.



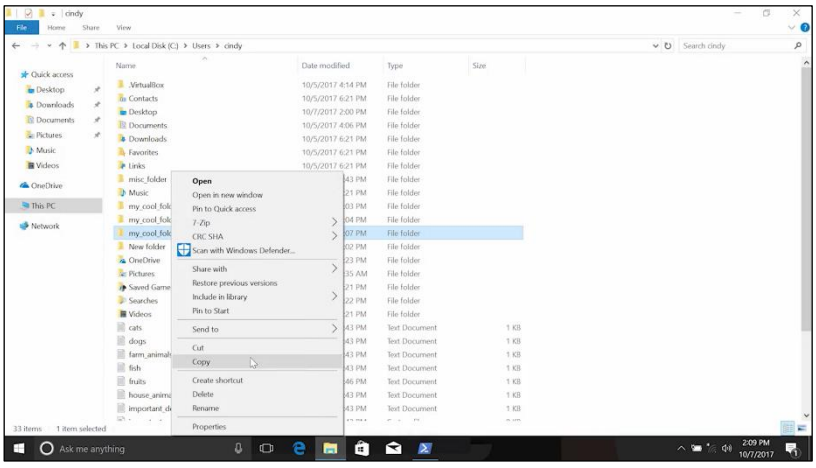


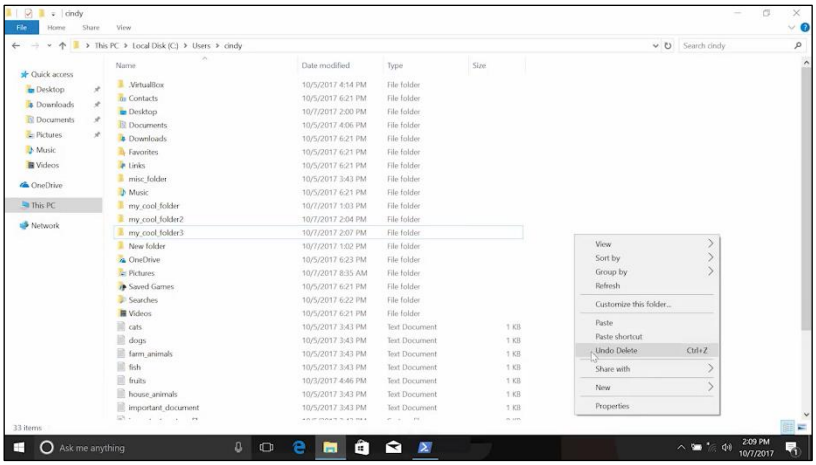
To clear your terminal app, what do you think you'll do? That's right. The clear command.



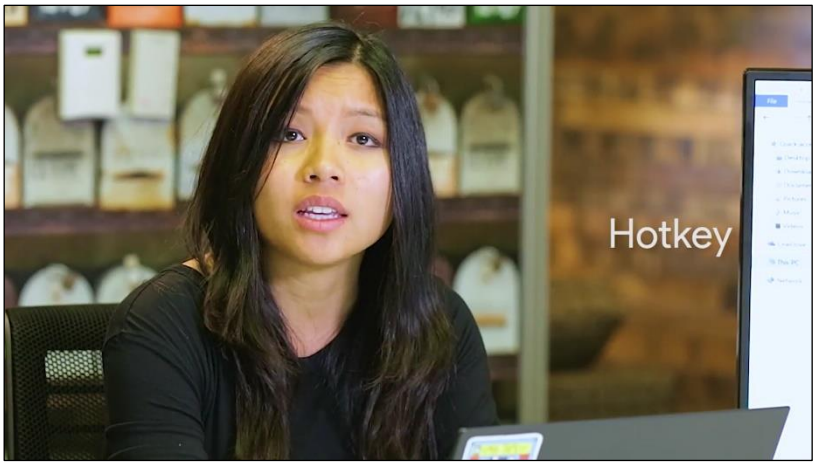
1.2.12 Windows copying files and directories

We've already created a few files and directories, but we need a couple more. We don't want to create them off from scratch. So let's make copies instead. In the Windows GUI, all you need to do is right-click, copy, then paste.

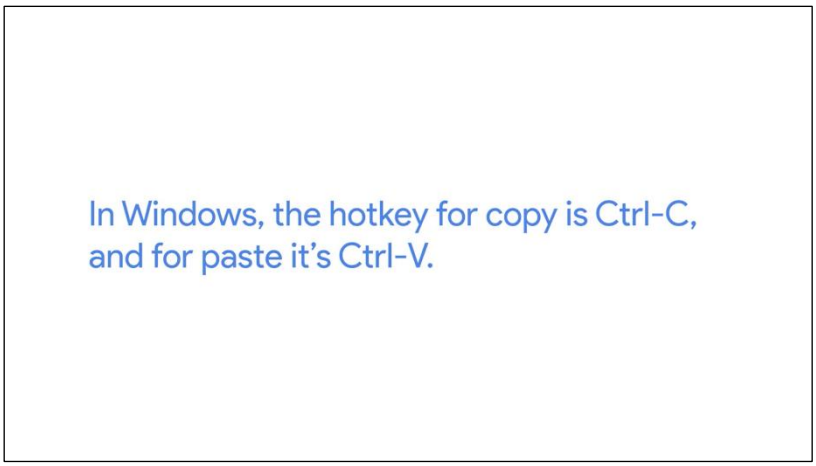




You can also use hotkeys if you want. A hotkey is a keyboard shortcut that does some sort of task.



In Windows, the hotkey for copy is Ctrl-C, and for paste, it's Ctrl-V.

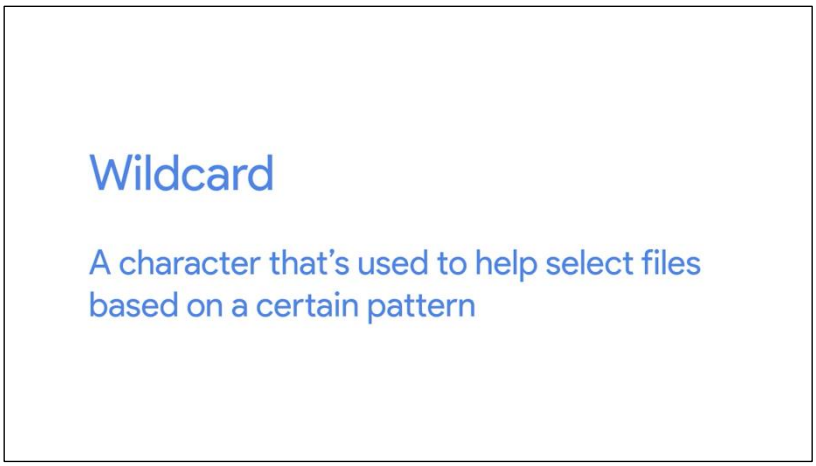


In PowerShell, the command used to copy something CP. We also need to add a file that we want to copy and the path of where we want to copy it too.

Let's copy mycoolfile.text to the desktop. There you can see mycoolfile.text was added to our desktop.

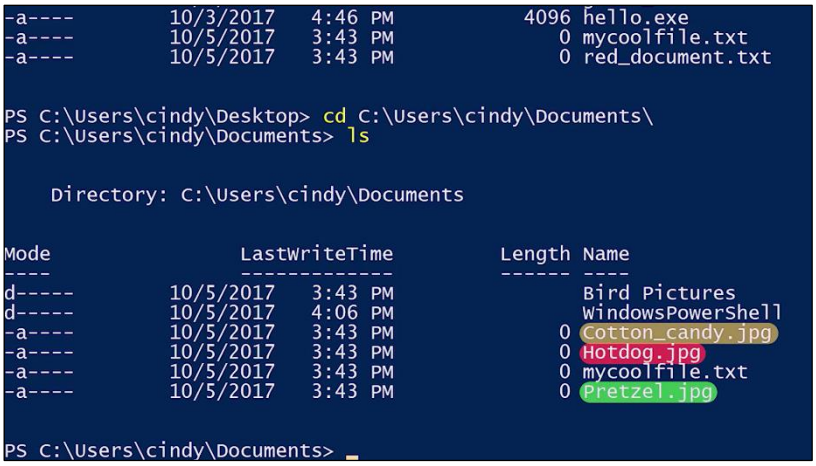
```
PS C:\Users\cindy\Documents> cp mycoolfile.txt C:\Users\cindy\Desktop\
PS C:\Users\cindy\Documents> cd ~\Desktop
PS C:\Users\cindy\Desktop> ls


    Directory: C:\Users\cindy\Desktop


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        10/5/2017   3:45 PM                CoolFiles
d-----        10/5/2017   3:45 PM                ShareMe
-a----        10/5/2017   3:45 PM              0 blue.txt
-a----        10/5/2017   3:43 PM              0 blue_document.txt
-a----        10/5/2017   2:00 PM            475 colors.zip
-a----        10/5/2017   3:43 PM              0 green_document.txt
-a----        10/3/2017   4:46 PM           4096 hello.exe
-a----        10/5/2017   3:43 PM              0 mycoolfile.txt
-a----        10/5/2017   3:43 PM              0 red_document.txt


PS C:\Users\cindy\Desktop>
```

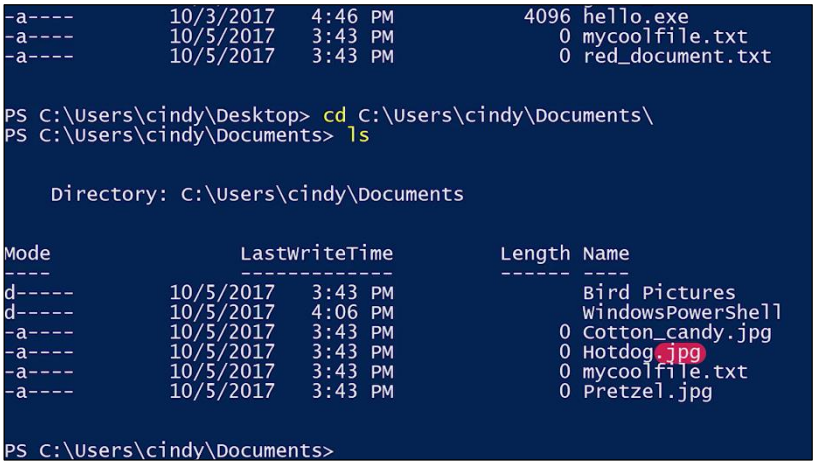
I have a few of these files I want to move over, but I'm feeling a little lazy and don't want to run this command over and over again. So, I'm going to use something called a wildcard to help me copy over multiple files at once. A wildcard is a character that's used to help select files based on a certain pattern.



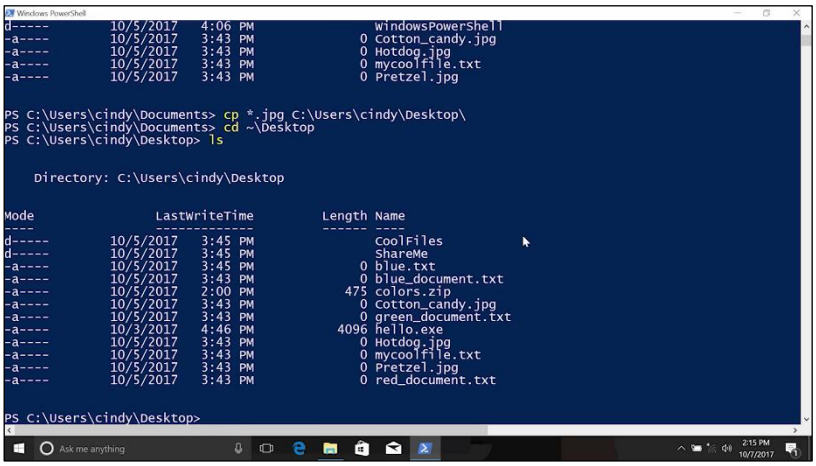
Let's say you want to get all the files that were JPG and copy them somewhere. Then I go on to my documents directory. I have files called hotdog.jpg, cotton-candy. jpg, and pretzel.jpg. I need to come up with a pattern to help me select all these files. What do they have in common besides being

named after delicious food? The.jpg extension. Literally, anything else can be in front of the.jpg file extension, and it won't matter.





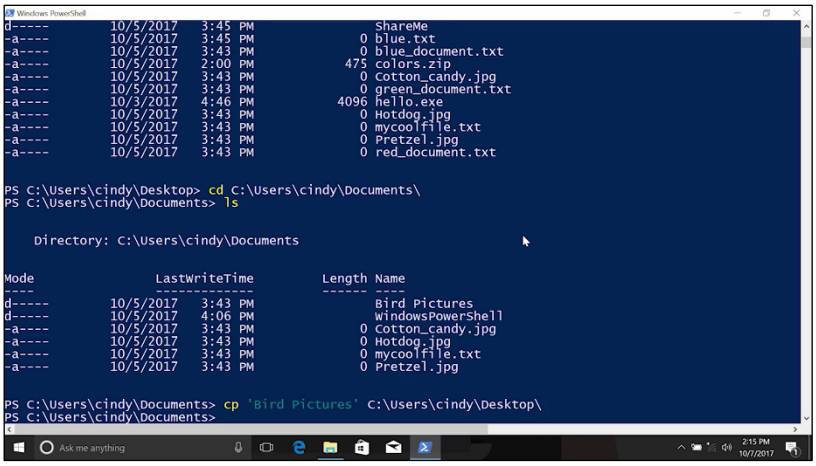
That's what the wildcard asterisk does. It's a pattern for anything. So I'm essentially saying, select all the files with the pattern anything.jpg. So, to copy over all the JPGs in the folder, I can use CP, asterisk symbol,.jpg, and the path I want to copy them to. Let's just verify. There it is.

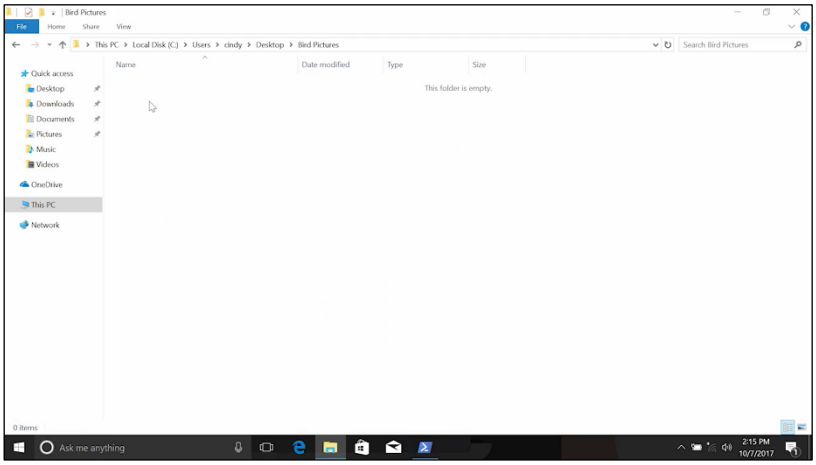


Now, instead of copying files one by one, we can use a single command to get all the files we want. For now, the only select you'll be using is the asterisk for all.

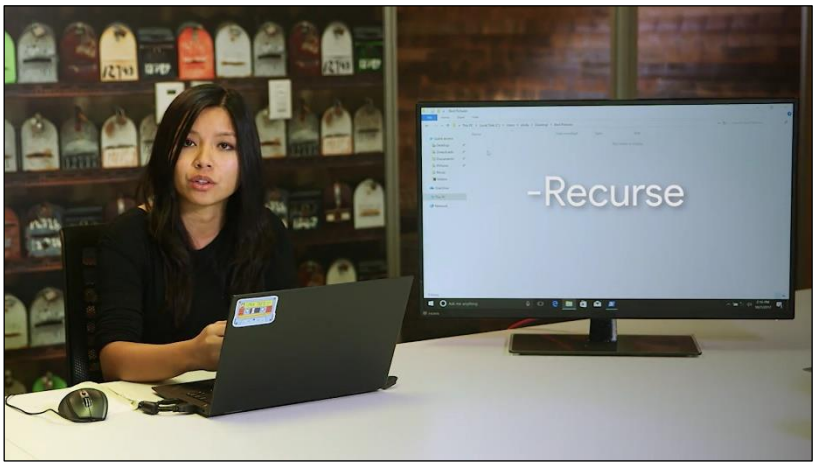
Next up, let's say I want to copy over a directory. I'm going to try to copy a folder called Bird Pictures to my desktop. Let's just go back into documents. That's Bird Pictures. Now copy Bird Pictures to

desktop. Now, this does exactly what we told you to do. It copies the directory. However, this directory is empty. What it doesn't do, is copy over the contents of the directory.

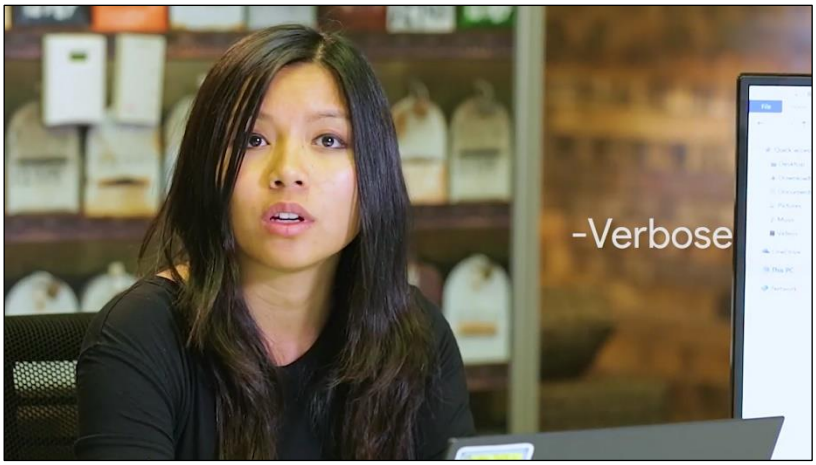




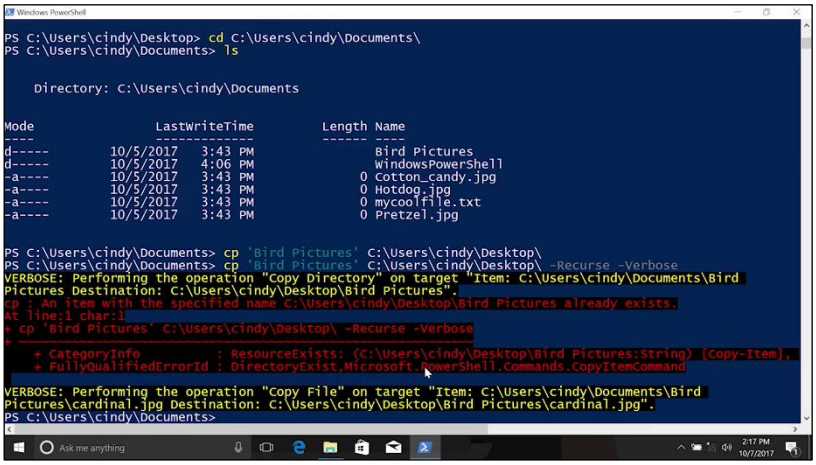
To copy of the contents of a directory, you need to use another command parameter, Recurse. The -Recurse parameter list the contents of the directory. Then if there are any sub-directories in that listing, it'll recurse or repeat the directory listing process for each of those sub-directories.

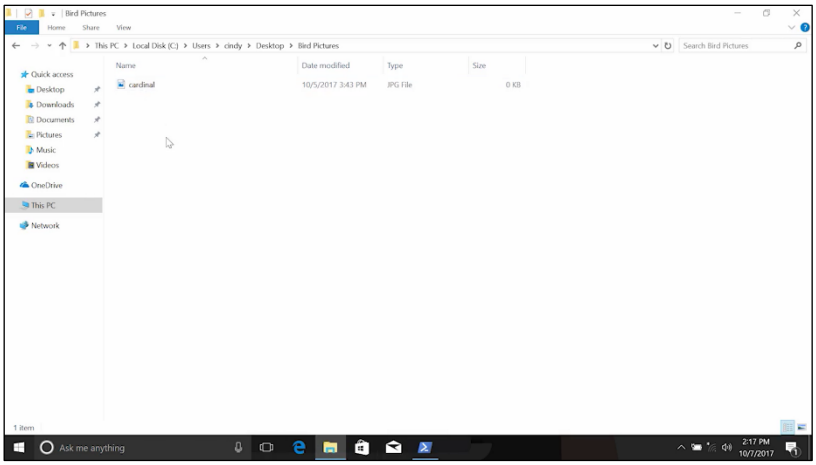


We need to use the -Recurse parameter with copy to copy the contents of the directory along with the directory itself. We're going to use a new parameter Verbose. Copy doesn't output anything to the CLI by default unless there are errors. When we use copy -Verbose, it will output one line for each file the directory being copied. Let's give it a try.



Copy Bird Pictures, and the Recurse, and Verbose file. This just messages us that we've already copied Bird Pictures, but what we didn't do, was copy over the file, which is now here.

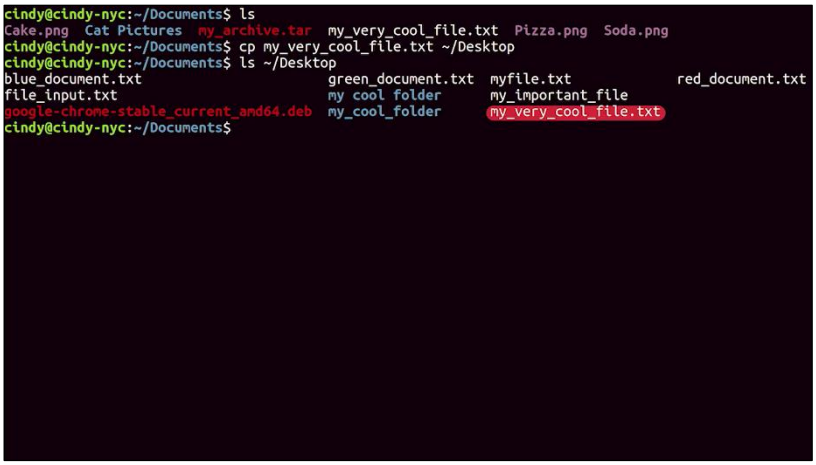




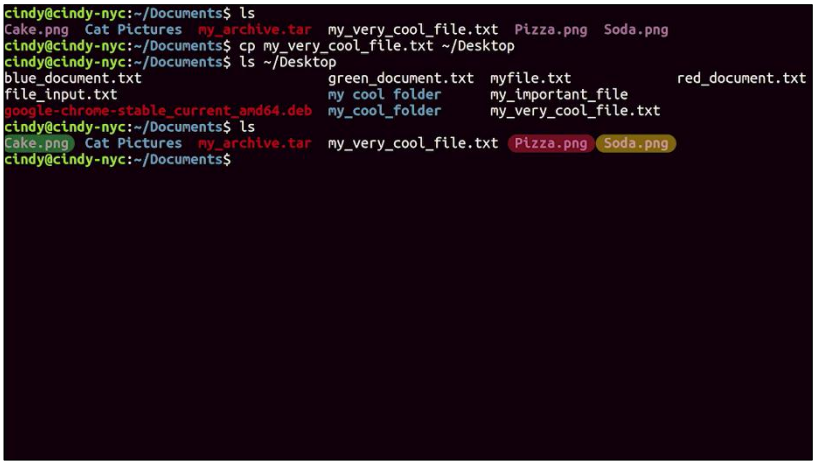
Excellent. Now the directory and all the contents are copied to my desktop.

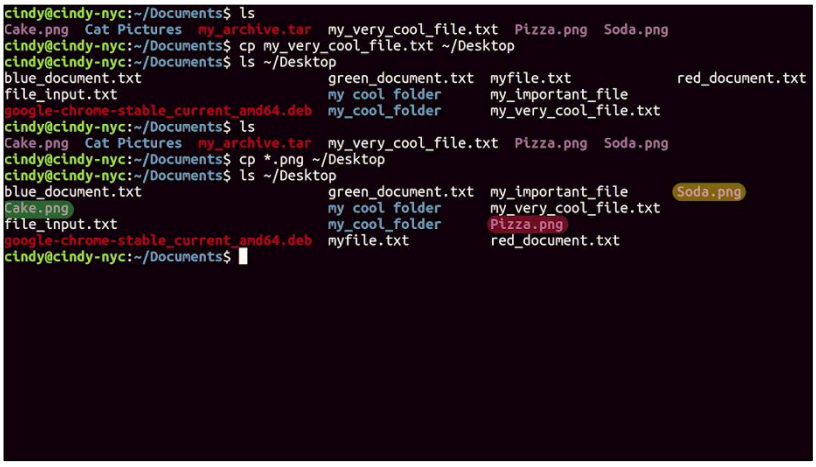
1.2.13 Linux copying files and directories

In Bash, the exact same Windows command can be used for copying files. Let's take a look at this directory. Let's copy my_very_cool_file.txt to my desktop. And there it is.

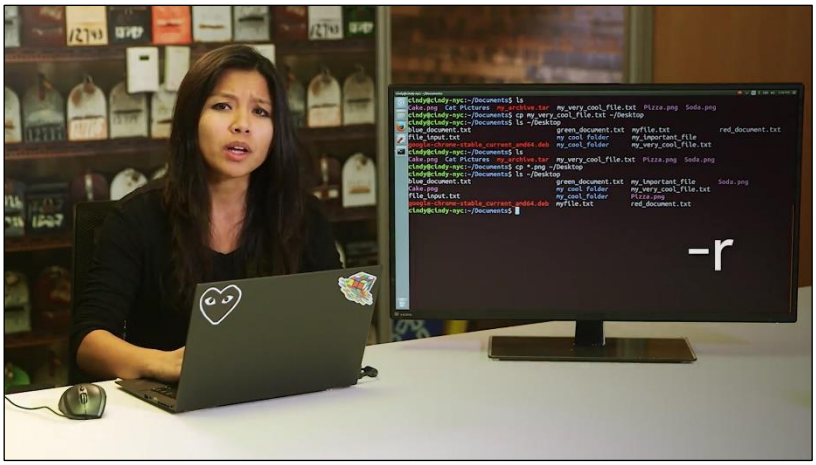


We can also use the same asterisk wildcard to select patterns. Since this is similar to our Windows copy command, what do you think we can use to copy over the .png files in this directory? I have files called Pizza.png, Soda.png, Cake.png. So I can use copy *.png, then the desktop directory. Now if I look at my desktop again, there they are.





The same copy rules apply in bash. If we want to copy over a directory, we have to recursively copy over the directory to get all the contents. The flag for recursive copy is dash r.

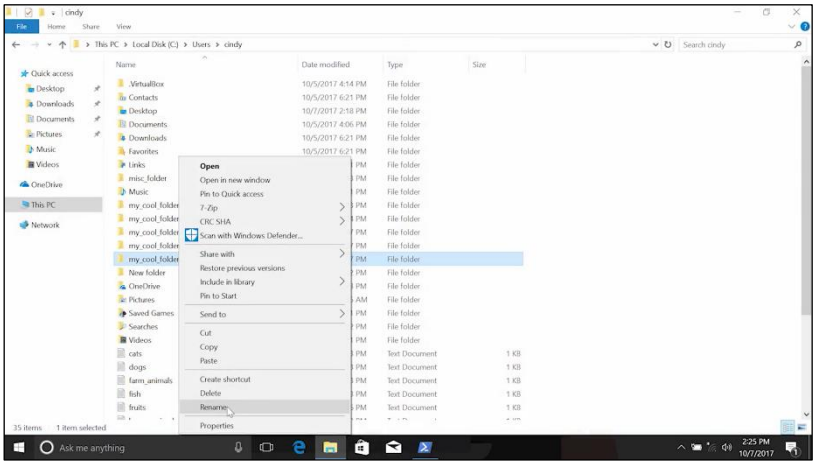


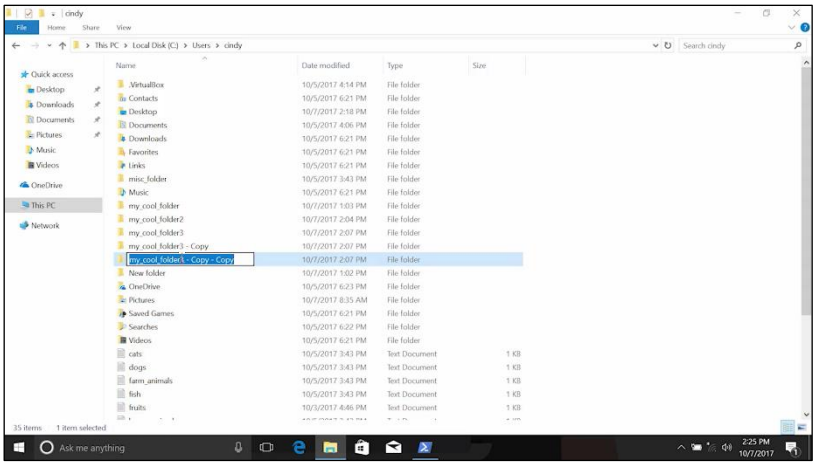
If I want to copy over my cat pictures folder to the desktop, I can do something like this. And there it is.

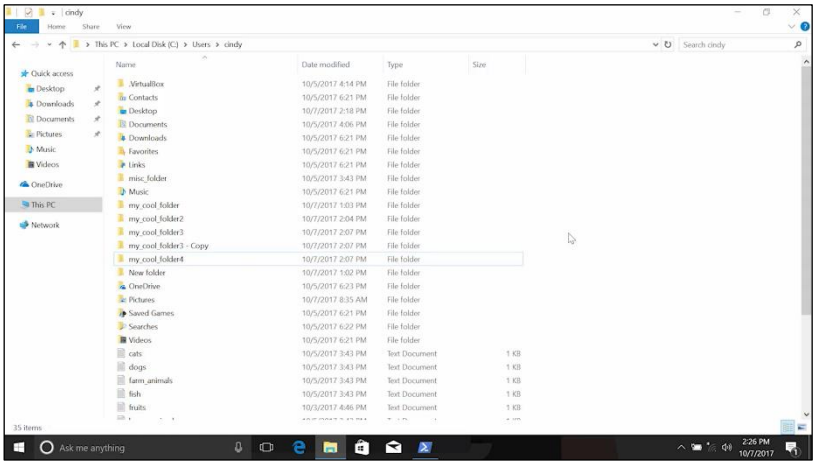
```
cindy@cindy-nyc:~/Documents$ ls
Cake.png  Cat Pictures  my_archive.tar  my_very_cool_file.txt  Pizza.png  Soda.png
cindy@cindy-nyc:~/Documents$ cp my_very_cool_file.txt ~/Desktop
cindy@cindy-nyc:~/Documents$ ls ~/Desktop
blue_document.txt                      green_document.txt  myfile.txt            red_document.txt
file_input.txt                         my cool folder      my_important_file
google-chrome-stable_current_amd64.deb  my_cool_folder      my_very_cool_file.txt
cindy@cindy-nyc:~/Documents$ ls
Cake.png  Cat Pictures  my_archive.tar  my_very_cool_file.txt  Pizza.png  Soda.png
cindy@cindy-nyc:~/Documents$ cp *.png ~/Desktop
cindy@cindy-nyc:~/Documents$ ls ~/Desktop
blue_document.txt                      green_document.txt  my_important_file     Soda.png
Cake.png                               my cool folder      my_very_cool_file.txt
file_input.txt                         my_cool_folder      Pizza.png
google-chrome-stable_current_amd64.deb  myfile.txt          red_document.txt
cindy@cindy-nyc:~/Documents$ ls
Cake.png  Cat Pictures  my_archive.tar  my_very_cool_file.txt  Pizza.png  Soda.png
cindy@cindy-nyc:~/Documents$ cp -r 'Cat Pictures' ~/Desktop
cindy@cindy-nyc:~/Documents$ ls ~/Desktop
blue_document.txt  google-chrome-stable_current_amd64.deb  myfile.txt            red_document.txt
Cake.png           green_document.txt                      my_important_file     Soda.png
Cat Pictures       my cool folder                          my_very_cool_file.txt
file_input.txt     my_cool_folder                          Pizza.png
cindy@cindy-nyc:~/Documents$
```

1.2.14 Windows moving and renaming files and directories

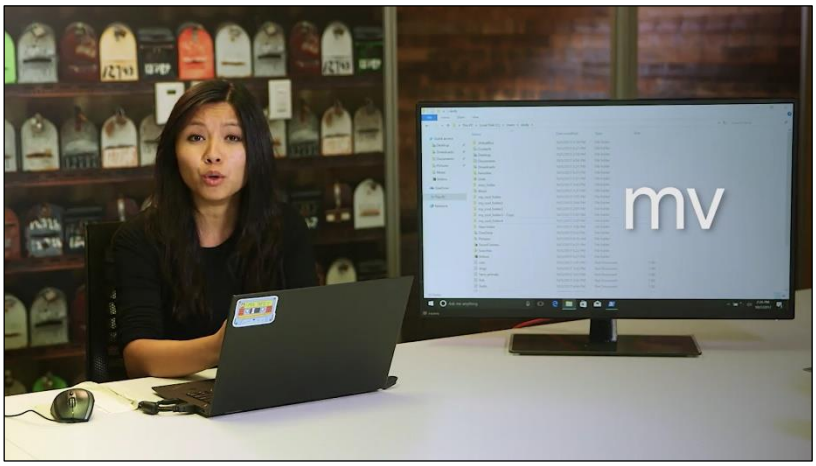
We talked about making and copying files and directories so far. But what if we wanted to rename something that we've created? Well, in the Windows GUI, if you are to rename a file, we just right-click and rename.



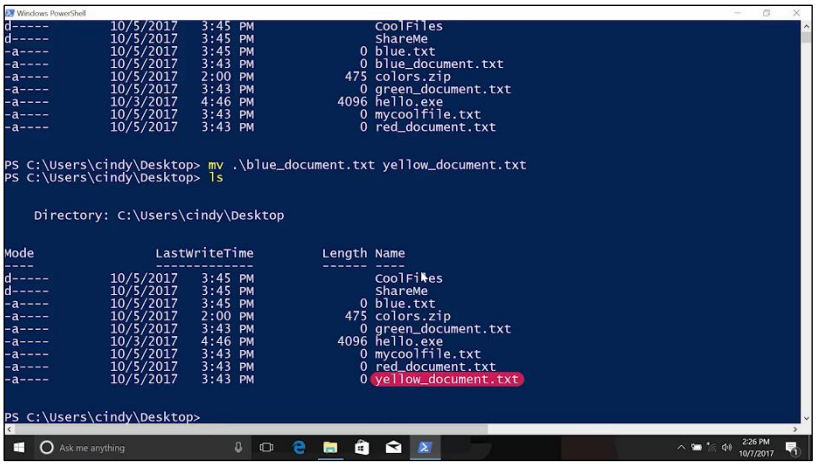




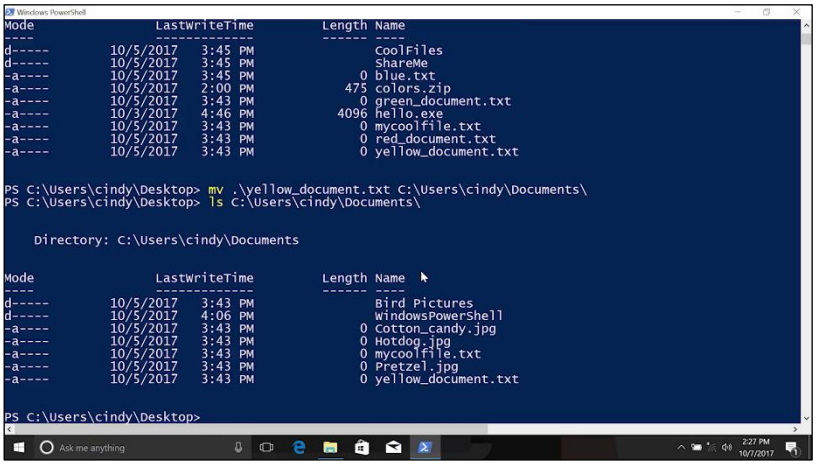
In the command line, if we wanted to rename a file, we can use the move or move item command.



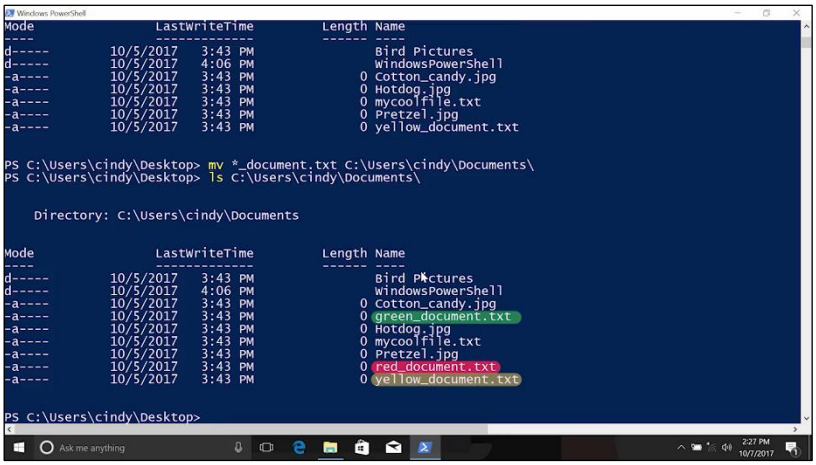
It lets us rename files. Lets move the file without changing the directory that it's stored in. On my desktop here, I have blue document and I'm going to move or rename it to yellow document. Now, you can see that I have a yellow document.



As you might guess, the move command also lets us move files from one directory to another. Let's move the yellow document into My Documents. I can verify that. There it is, cool.



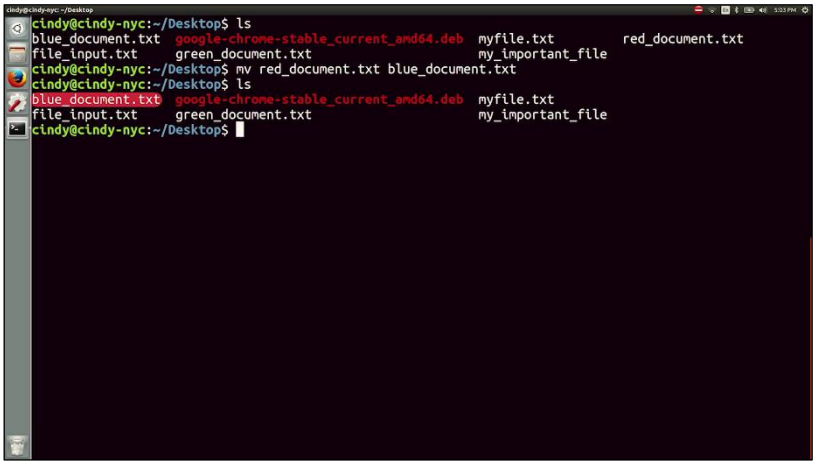
You can even move multiple files by using wildcards. And now you can see, the rest of my colored documents went into My Documents.



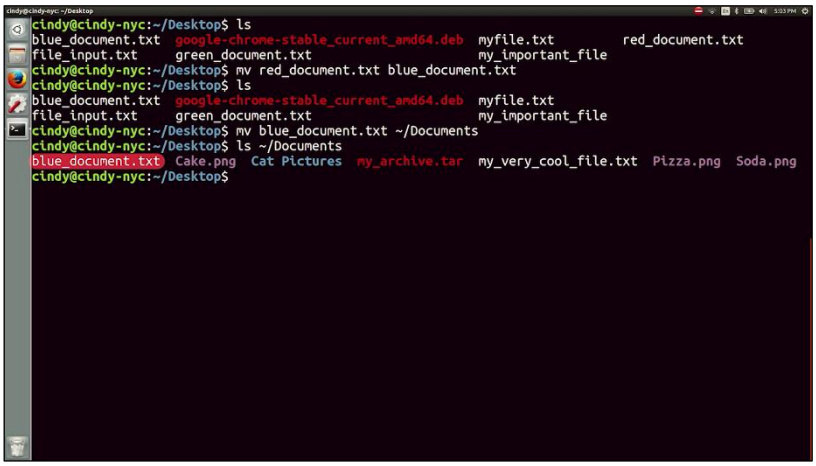
1.2.15 Linux moving and renaming files and directories

The exact same command can be used for Linux. Mv, or move, can rename and move files in directories. Same thing applies here.

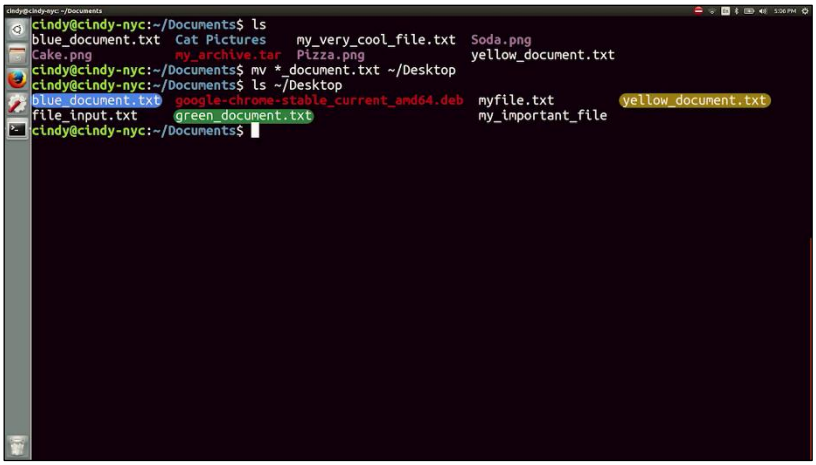
I'm going to move my red_document and rename it to blue_document. Now we can see it's been renamed to blue_document.



Then, I'm going to move the blue_document in to the documents folder. There it is.

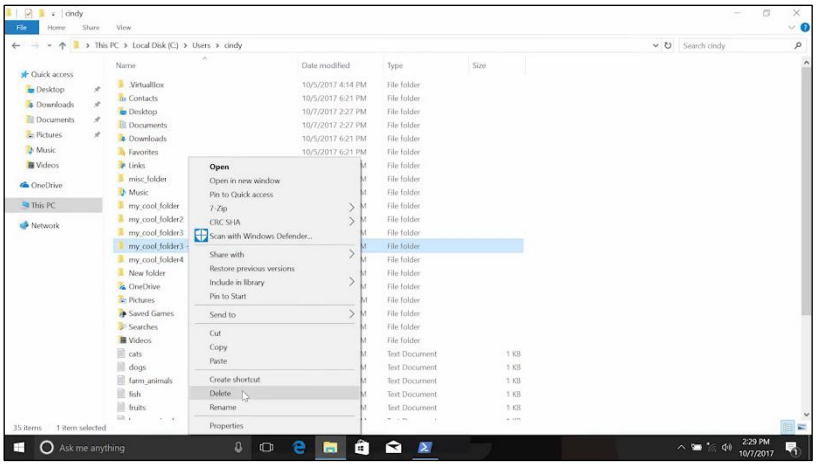


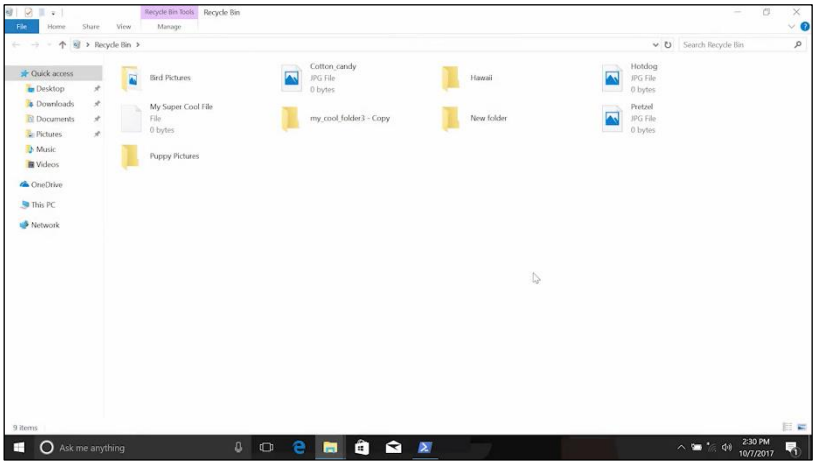
Using wildcards, we can move multiple files at once, just like Windows. Let's move all of the underscored document files here to our desktop. Now if we check the desktop, there they are.

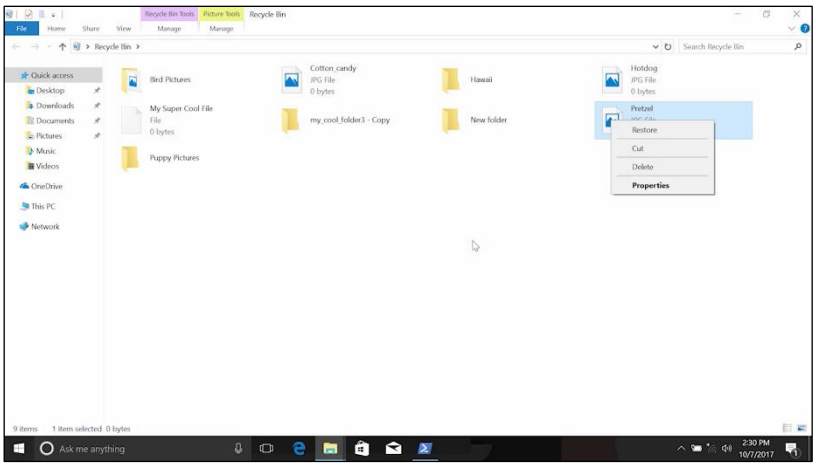


1.2.16 Windows removing files and directories

All righty, now that we've learned how to list, create, and move around files in directories, let's start removing them. In the Windows GUI, if you wanted to remove a file or folder, just right-click and delete.

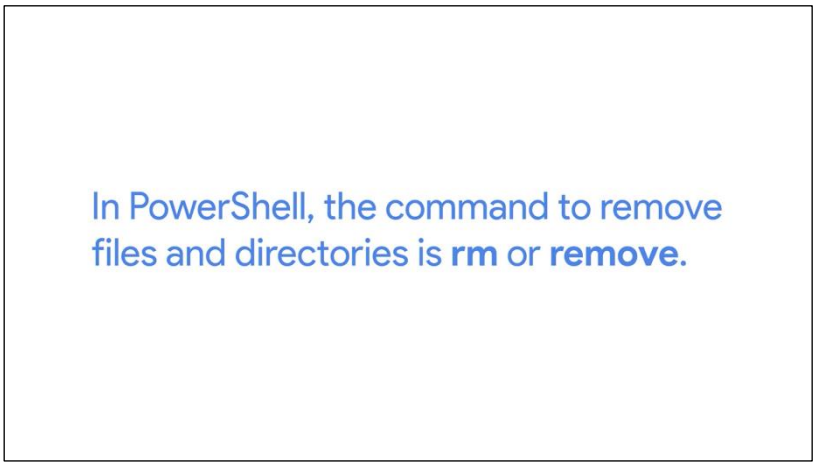






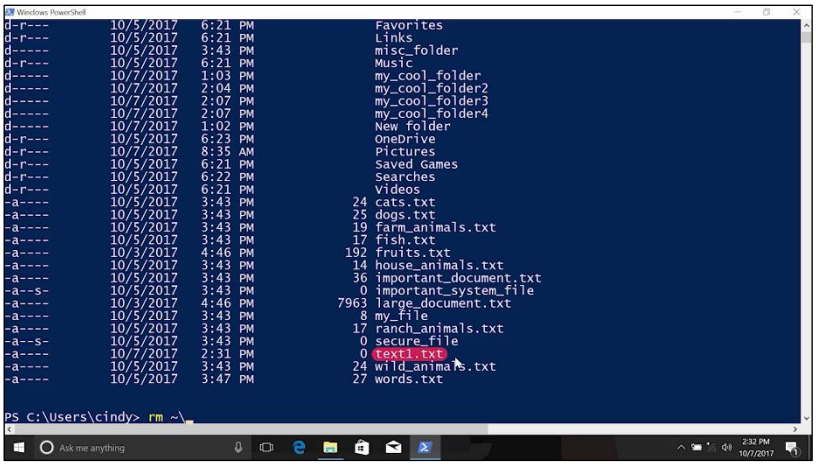
The file ends up in the recycle bin, which you can find on your desktop. If you wanted to restore a file here, you could just right-click and Restore. If you empty your bin for any reason you won't be able to retrieve those files.

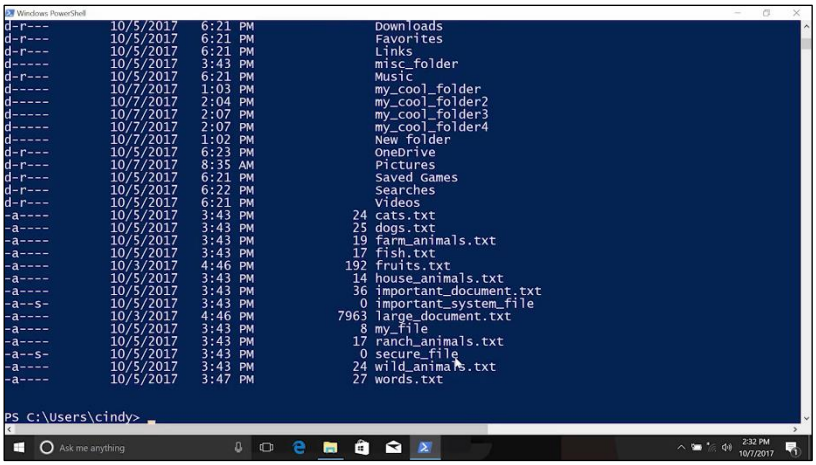
In PowerShell, the command to remove files and directories is rm or remove.



Take caution when using remove because it doesn't use the recycle bin. Once the files and directories are removed, they're gone for good.

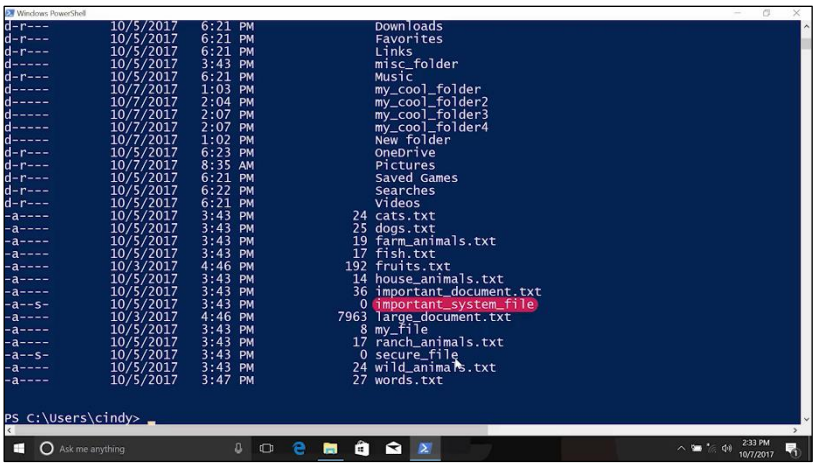
Let's remove a file called text1.txt in my home directory. We can see, There it is. I'm just going to remove it. And now it's gone.

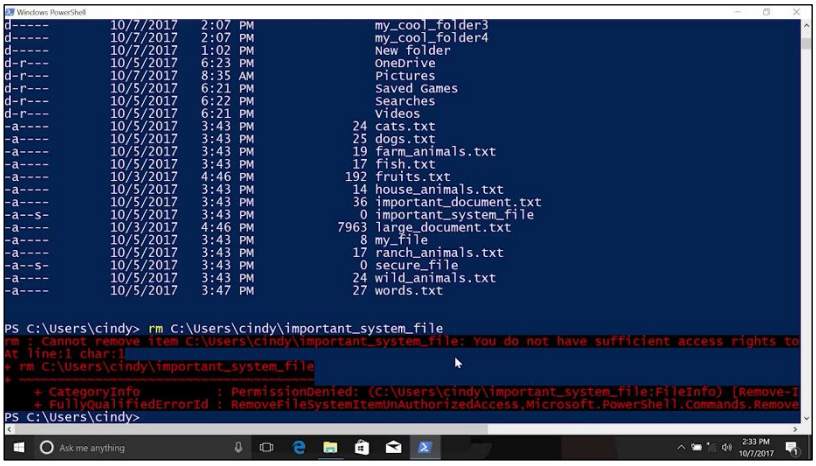




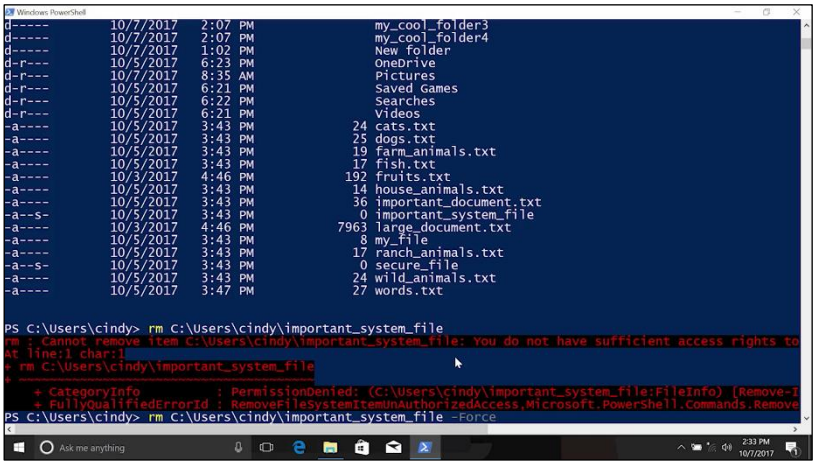
The remove command might seem like a dangerous weapon in the wrong hands. Fortunately, there are safety measures in place that only give this ability to users that are actually authorized to use it. We'll talk more about file permissions in a different lesson. But let's take a quick look at what I mean.

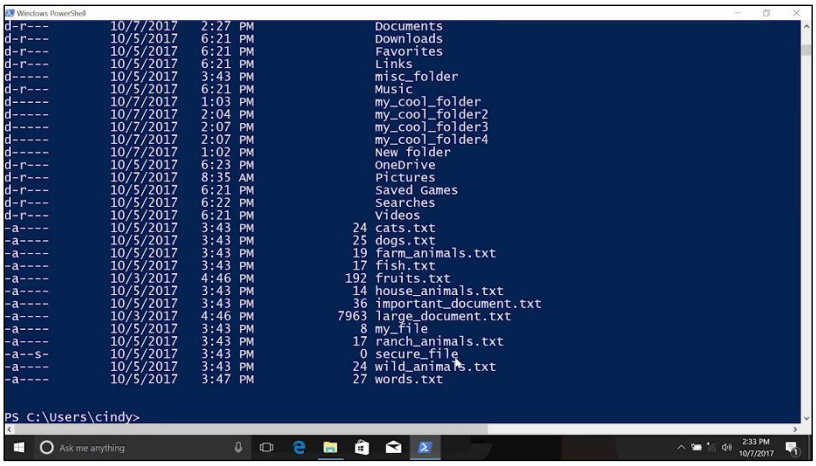
Let's remove a file called important_system_file. I get an error message saying that I don't have permission to delete this file.





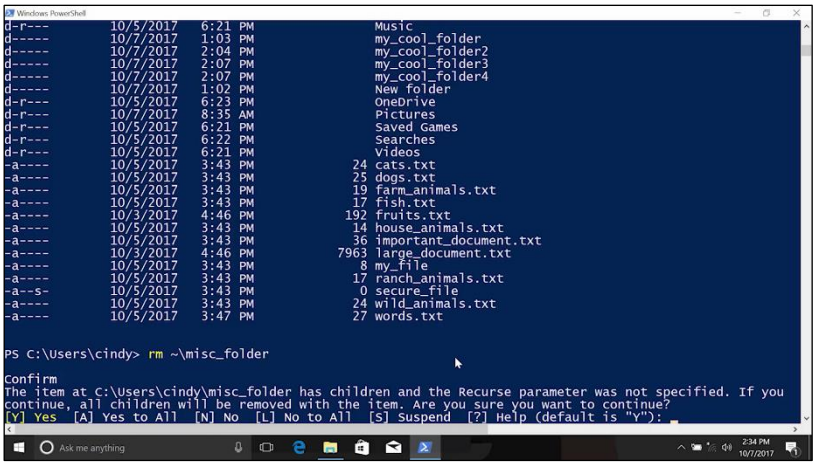
In some cases like this one, it's because it's been marked as a system file. In other cases, it might be because I don't have enough permissions in the file system to remove the file. I do have the right permissions this time, but since it is an important file, PowerShell wants to make sure that I meant to do this. If I repeat the command with the -Force parameter, remove will go ahead and remove the file. Let's take a look. -Force, And you can see the file's gone.





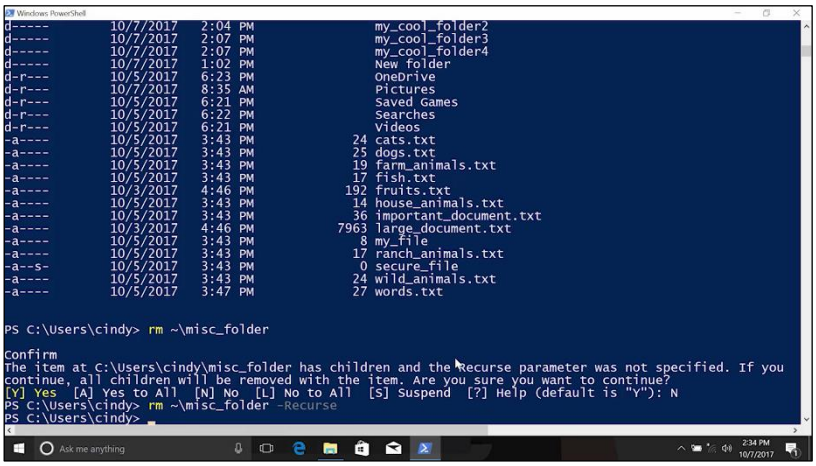
If the file belongs to someone else, or if I'm not an administrator, then I might not have the right permissions to remove the file. In that case I'll need to access an administrative account to remove the file.

Okay, let's try removing a directory with remove next. Here we go. Here's another place where PowerShell is going to ask us if we really meant to do this. Since this is in a directory, it contains other files. And we did not use the -Recurse parameter.



We see a prompt asking us to confirm if we really want to remove the directory and all its contents. We can say Yes or Yes to All to continue. We can also cancel this command and run it again with the -Recurse parameter. That way, PowerShell knows that we understand the consequences of what we're doing.

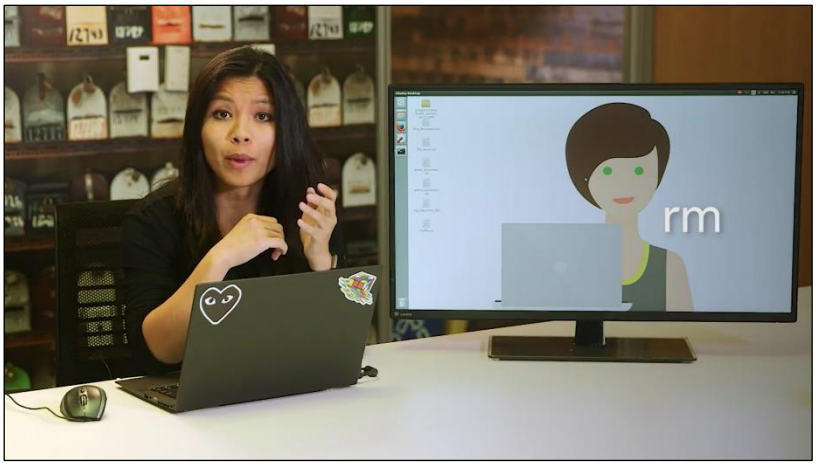
So let's go ahead and cancel this and try again. -Recurse. Yeah, now it's gone. And that's the remove command in a nutshell.



Again, because of the nature of this command, you'll want to be extra careful when removing files or directories.

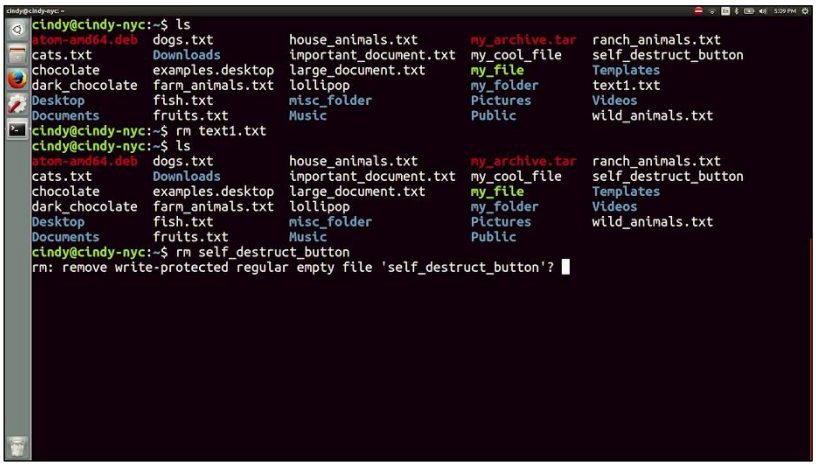
1.2.17 Linux removing files and directories

To remove files from Linux, just like in Windows, we can use the rm or remove command.



Let's remove this text1 file. And just like that, it's gone.

Similar to Windows, we get a message if we try to remove something that we shouldn't be able to. Let's remove this self_destruct_button. Awesome, everything is working as intended.



Next let's try removing a directory. If you thought to yourself that we need to also recursively remove this directory, you'd be right, excellent deduction skills. So rm -r, let's remove the misc_folder directory. And if we check the misc folder is now gone.

Remember, when using rm command, take extra precaution that you aren't removing something important by accident.

1.2.18 Cindy why OS is important

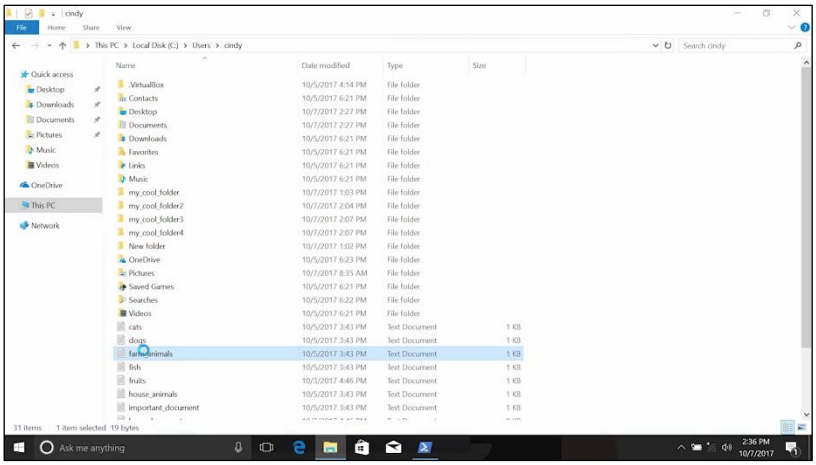
[MUSIC] I knew enough to be dangerous, and I think that's what got me into my systems administrator role in Linux. When I got in that role I was working with people who were insanely brilliant. They had Wikipedia pages written about them, about their contribution to Linux and all these open source contributions. They weren't using the operating system, they were engineering it, they were contributing code, fixing hardware issues and fixing software issues. That type of environment really leveled up my skills in terms of Linux. because I had to learn, I had to keep up somehow, so I would read their bug reports and what they did. I guess I'd say, about after a year, I was really comfortable on the command line. I was packaging my own tools and I was writing code and I was contributing to open source projects. It was definitely an eye opener considering how much I thought I knew about operating systems to what I know now. The feeling you get when you contribute code to something that thousands, if not millions of people might use, you kind of don't believe that you just did that. That's the feeling you get when you do something in the open source community. I'm passionate about operating systems because there's a lot of stuff that you can do with them. You can contribute code to an operating system like Ubuntu or Debian and you can make an actual impact. I mean I can't go out and build a new CPU or something and have people use it but I can contribute code, I can fix a bug. There's so much stuff you can do with the operating system, it's unbelievable.

1.3 File and text manipulation

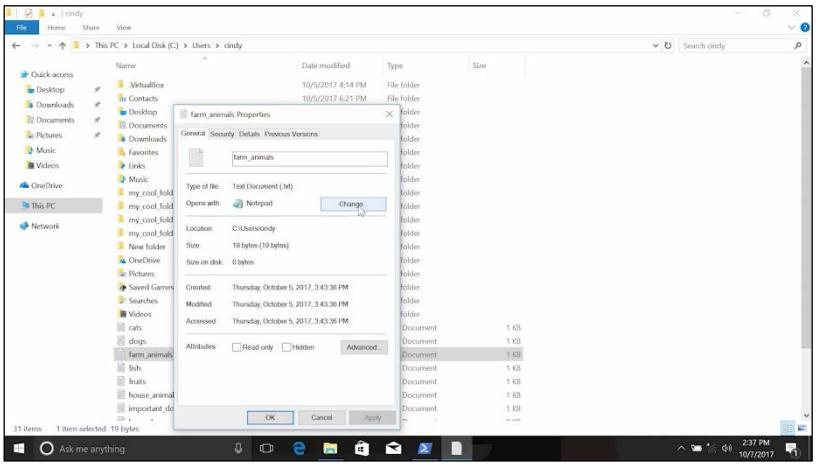
1.3.1 Windows Display file content

Now that we've learned the basics of file and directory navigation, let's learn how we can display and edit files, search for text within files and more.

In the Windows GUI, if we want to open a file and view its contents, we can just double click on the file. Depending on the file type, it will open on a default application. In Windows, text files default to open in an application called notepad.



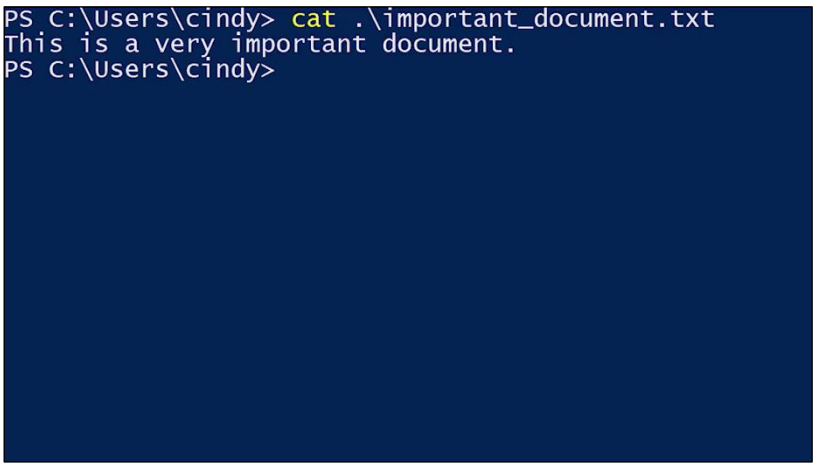
But we can change this if we want to. To change the default application that opens files, just right click and click Properties. Under 'Open with', we can change the application to another text editor, like Word Pad.



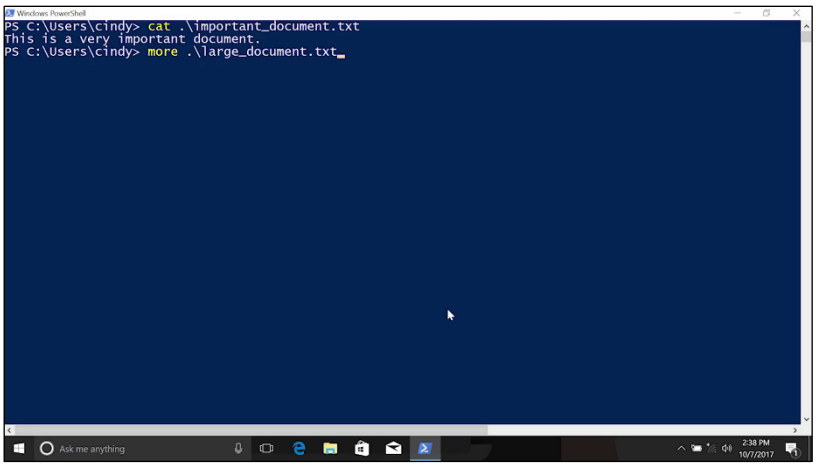
Most of the files that we'll be dealing with throughout this course, will be text and configuration files. So, let's just focus on those files instead of images, music files, etc.

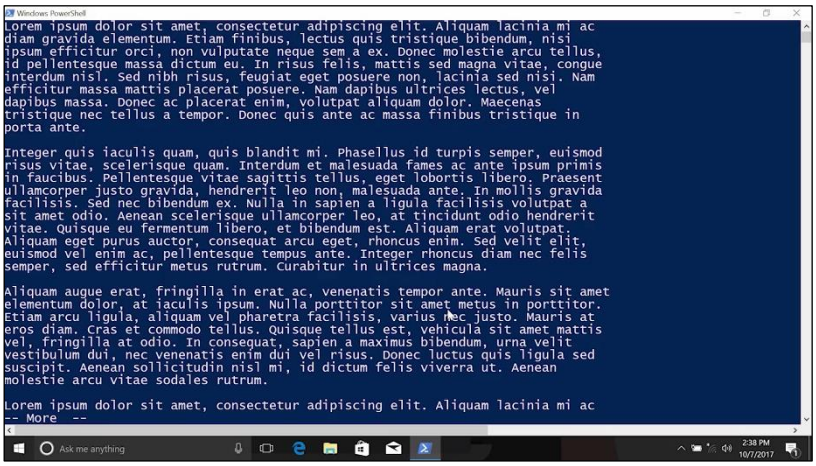
Viewing the contents of a file in PowerShell is simply using the 'cat' command, which stands for concatenate. Let's give it a try.

This will dump the contents of the file into our shell.

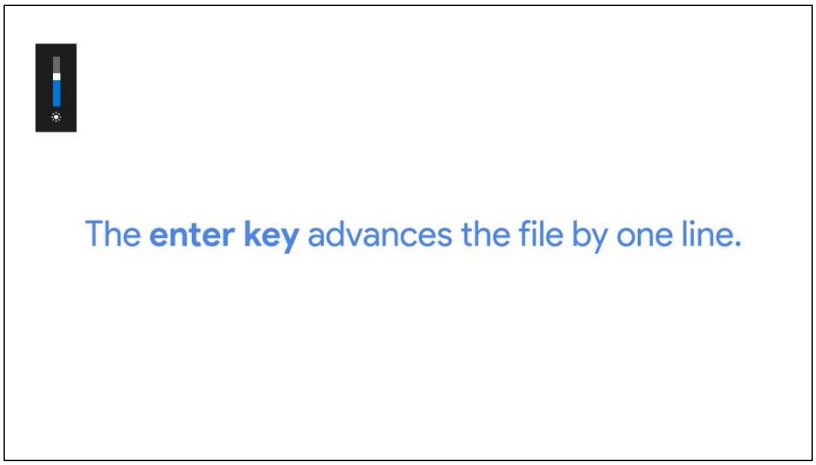


This isn't the best solution for a file since it just keeps writing the content until the whole file is displayed. If we want to view the contents of the file one page at a time, we can use the 'more' command, like this. The 'more' command will get the contents of the file but will pause once it fills the terminal window.

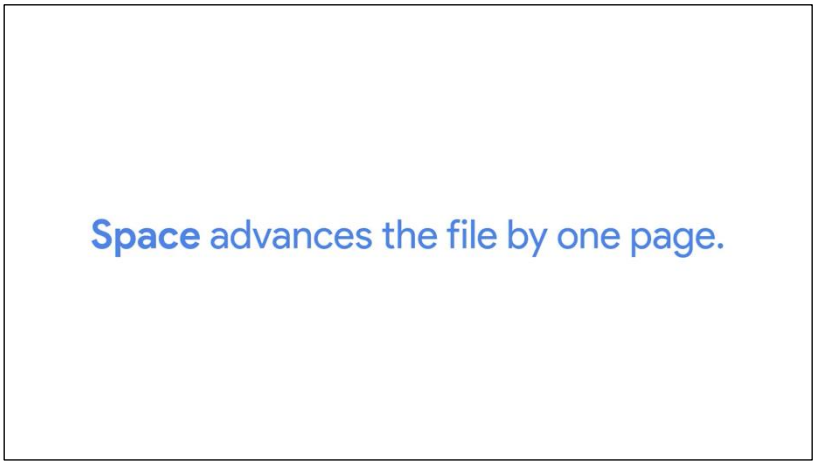




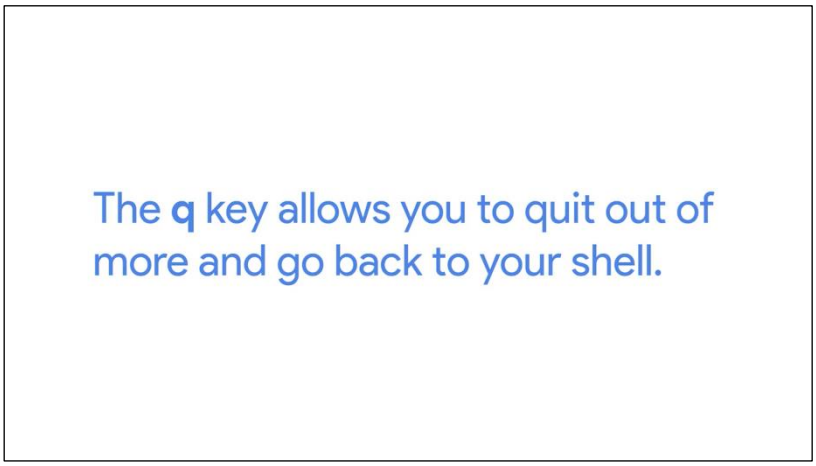
Now, we can advance the text at our own pace. When we run the 'more' command, were launched into a separate program from the shell. This means that we interact with the more program with different keys. The Enter key advances the file by one line.

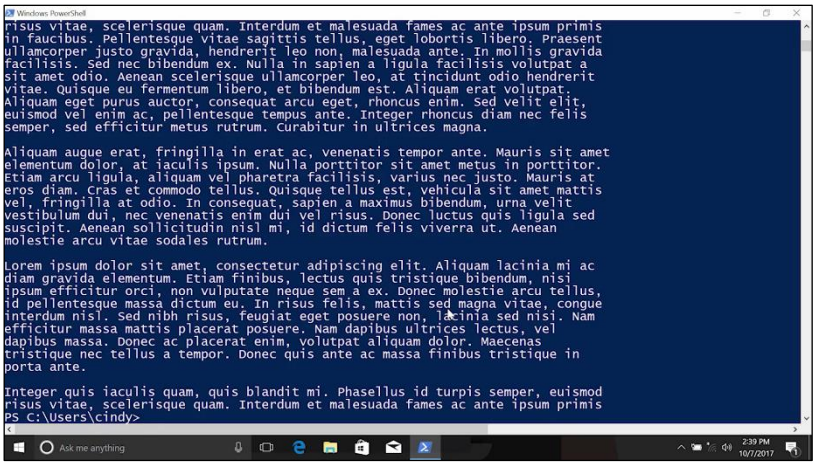


You can use this if you want to move slowly through the file. Space advances the file by one page. A page in this case depends on the size of your terminal window.

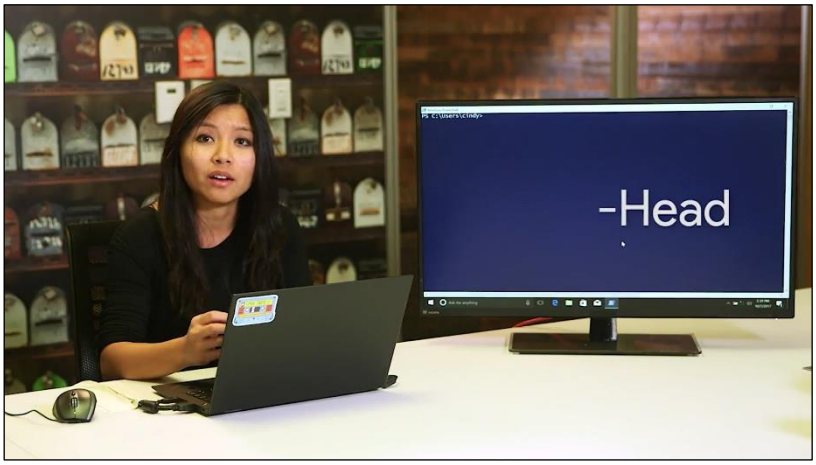


Basically, 'more' will output enough content to fill the terminal window. The q key allows you to quit out of 'more' and go back to your shell. If we want to leave the 'more' command and go back to our shell, we can just hit the q key. Here we are.

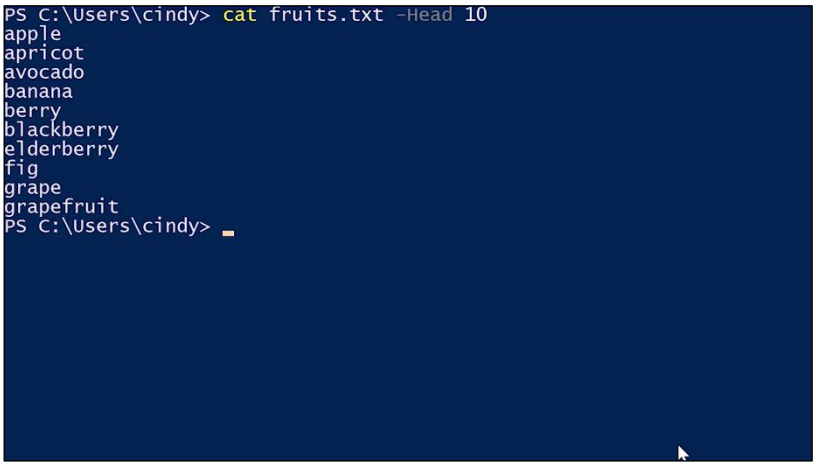




Now, what if we just wanted to view part of the file? Let's say we want to quickly see what the first few lines of the text file are. We don't really want to open up the whole file. Instead, we just want to get a glimpse of what the document is. This is called the head of the file. To do this, we can go back to 'cat' and add the -Head parameter.



This will show us the first 10 lines of the file.



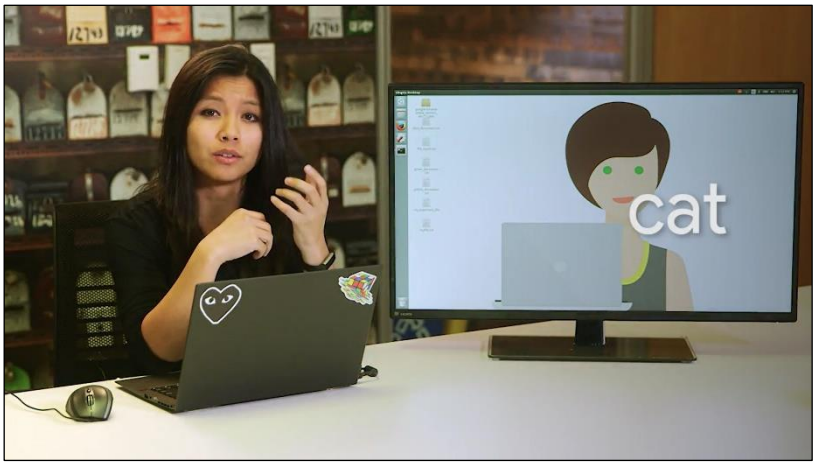
Now, what if we wanted to view the last few lines or the tail of the file? I bet you can guess what you are going to do. This will show us, by default, the last ten lines of the file.

```
PS C:\Users\cindy> cat fruits.txt -Head 10
apple
apricot
avocado
banana
berry
blackberry
elderberry
fig
grape
grapefruit
PS C:\Users\cindy> cat fruits.txt -Tail 10
papaya
passion fruit
peach
pear
persimmon
pineapple
raisin
raspberry
star fruit
strawberry
PS C:\Users\cindy>
```

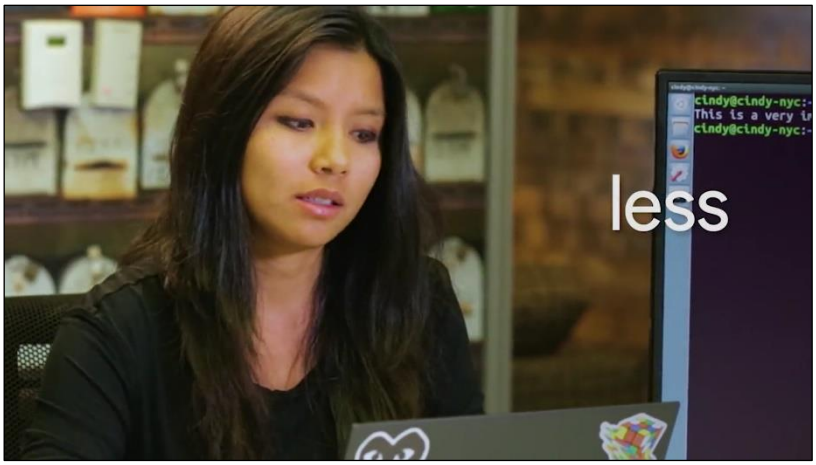
Again, these two commands don't seem like they have any immediate use to you yet. We'll see their benefits when we work with logs in an upcoming lesson. Now, let's take a look at how to do these same tasks in Linux.

1.3.2 Linux Display file content

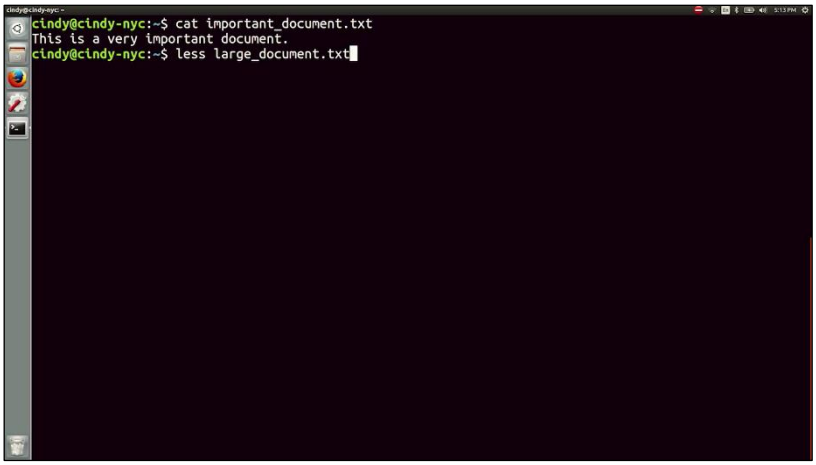
To read a simple file in Bash, we can also use the cat command to view a document. So let's look at important document.

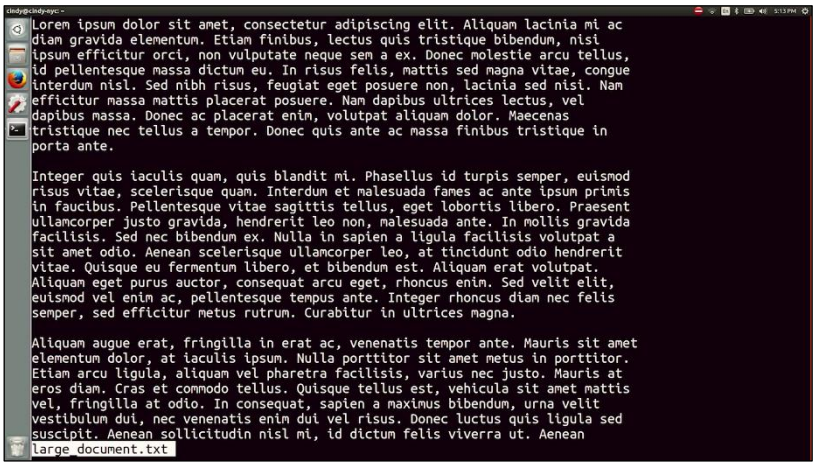


The cat command is similar to the Windows cat command, since it doesn't do a great job at viewing large files. Instead, we use another command, less.



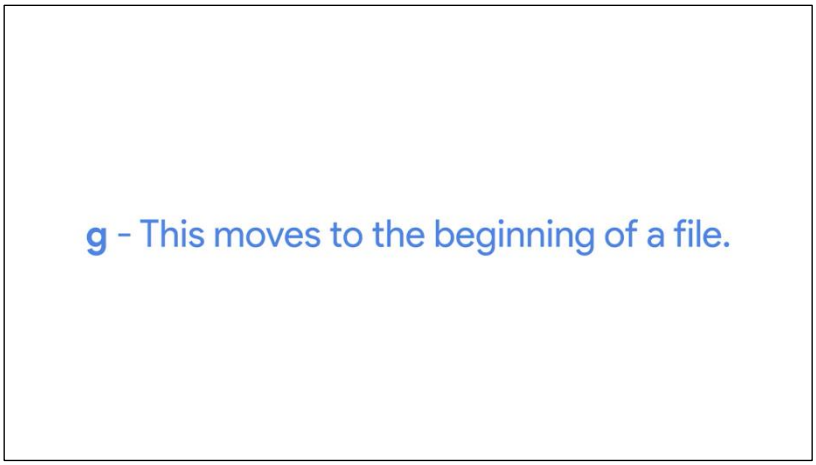
Less does a similar thing that more does for Windows, but it has more functionality. Fun fact, there's a Bash command called more, but it's been slowly dying out in favor of less. It's literally a case of less is more. Similar to more, when we use less we're launched into an interactive shell.



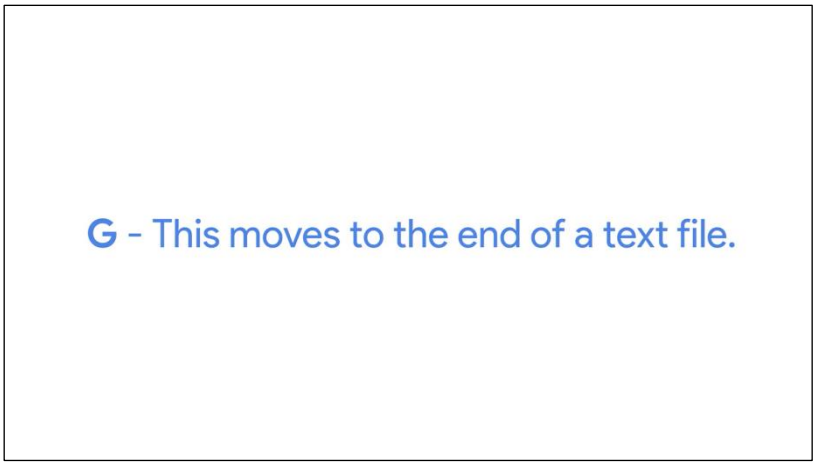


Some of the most common keys you'll use to navigate this tool are the up and down keys, page up and page down.

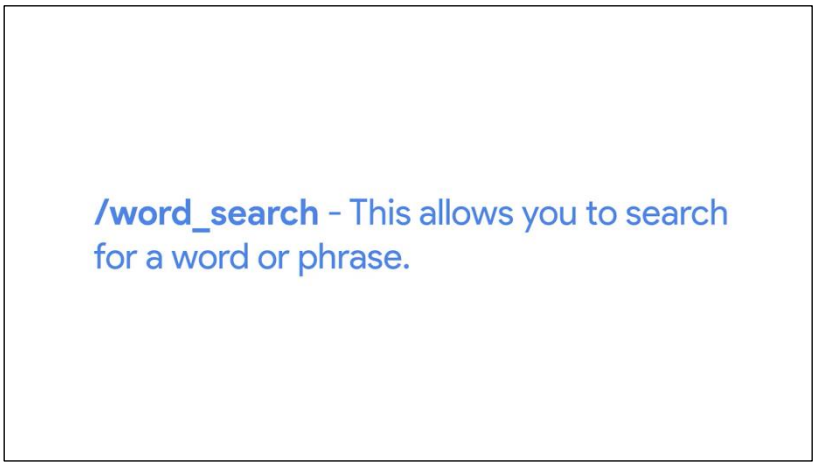
g, this moves to the beginning of a file. You can see now we're at the beginning.

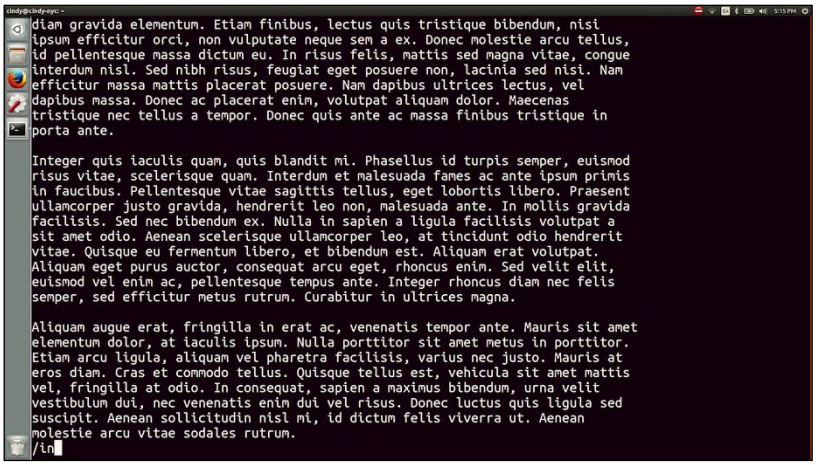


Capital G, this moves to the end of a text file. Now we're at the end.

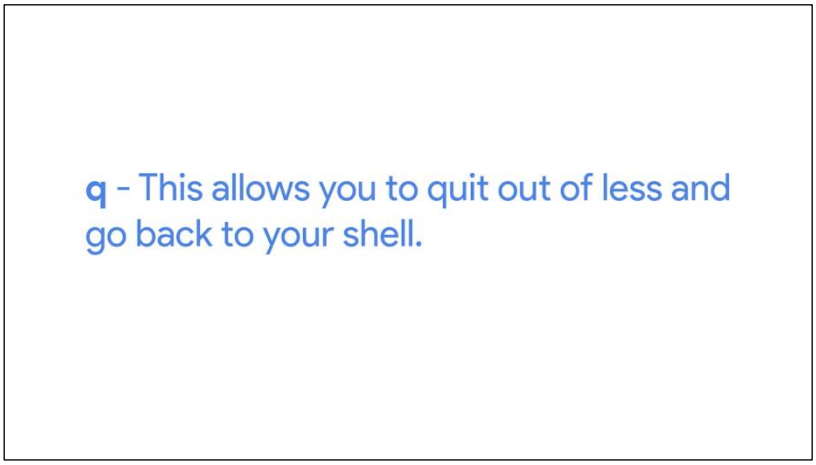


Slash and then a word_search. This allows you to search for a word or phrase. If I type in slash then type the word I want to search for, I can scan through the text file for words that match my search.



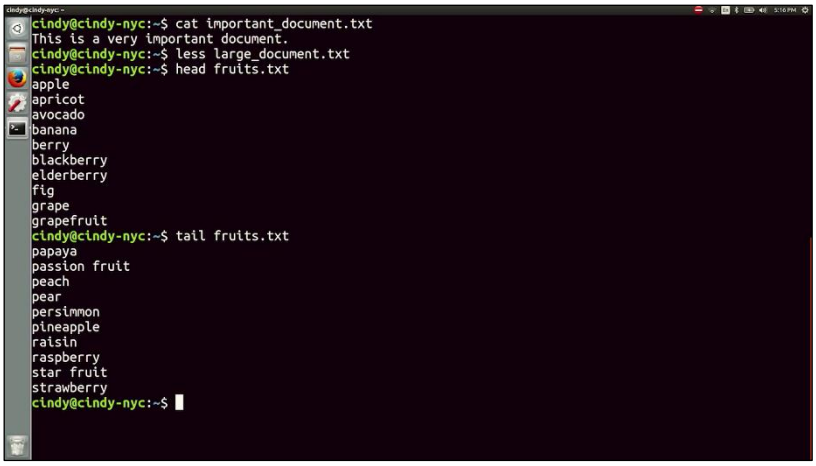


Q, this allows you to quit out of less and go back to your shell, similar to the q key in the Windows more command.



Do you see how less offers functionality like searching within a file? Less is a great tool to use to view files of any size. You'll no doubt end up using this command often as an IT support specialist.

Similar to the Windows cat and head parameter, we can do the same thing in Linux using a command called head. This will show you, by default, the first ten lines of a file. Now what if you wanted to view the last few lines of a file? You can use a command called tail. This will show you, by default, the last ten lines of a file.

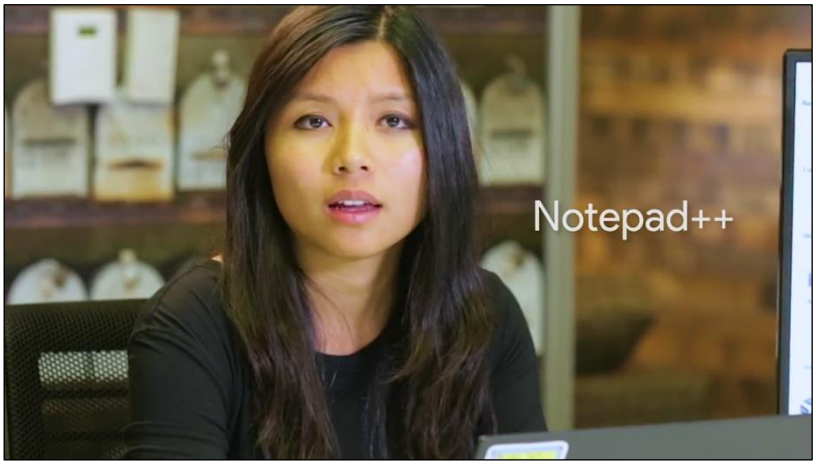


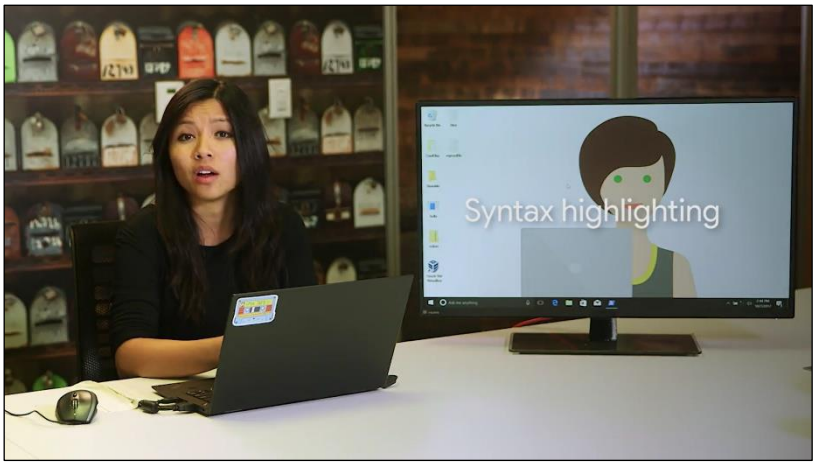
1.3.3 Windows Modifying text files

So far, we've discussed how to read and modify files. But we haven't covered how to edit file contents yet. Spoiler alert, you're about to learn. You can edit text based files in notepad, which we used earlier to view a text file.

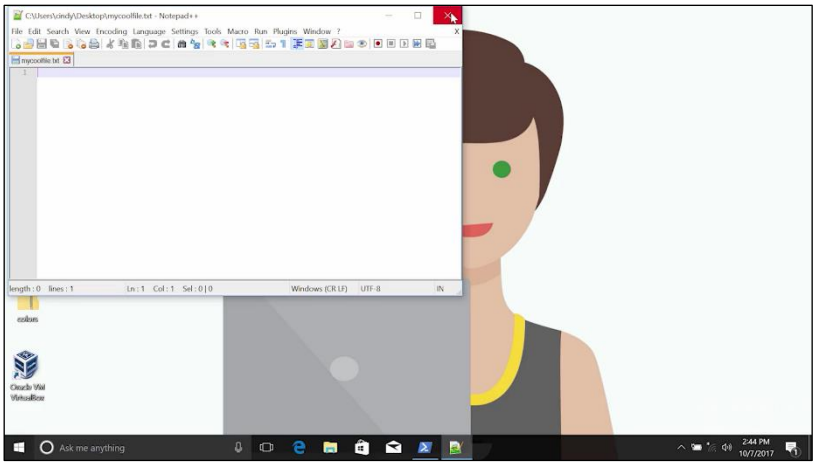
Notepad is great for basic editing. But when making changes to configuration files, scripts, or other complex text files, you might want something with more features. There are lots of good editors out there for the Windows GUI.

For this demonstration, we'll use one called Notepad++. Notepad++, which you can access from the next supplemental reading, is an excellent open source text editor, with support for lots of different file types. Notepad++ can open multiple files and tabs. It also does syntax highlighting for known file types, and has a whole bunch of advanced text editing features.



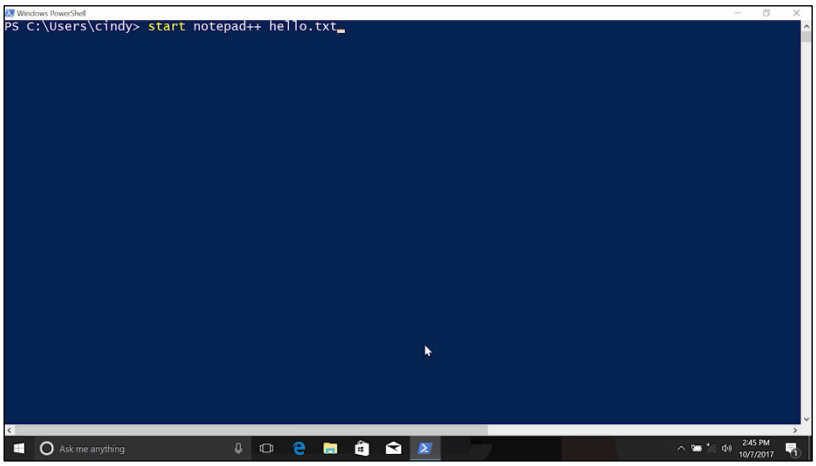


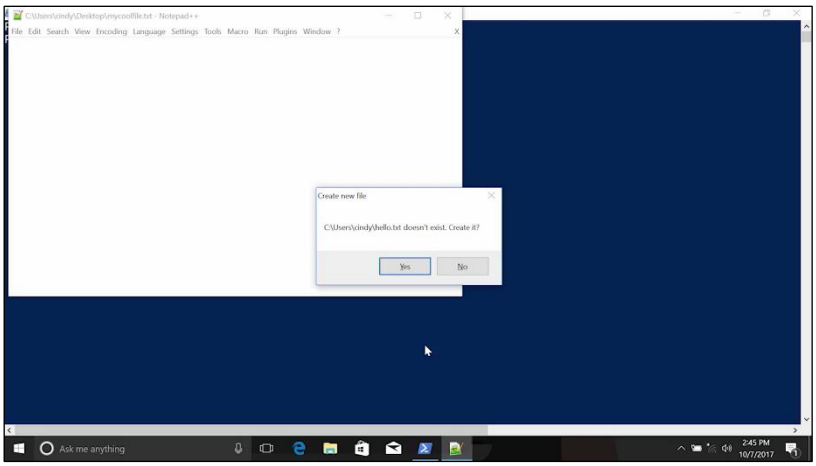
Syntax highlighting is a feature that a lot of text editors provide. It displays text in different colors and fonts to help you categorize things differently. We've already installed Notepad++ on our machine. So, you can check out their website and do the same. Now, you can edit any file using Notepad++ by right clicking it and selecting edit with Notepad++.



What if you wanted to edit a file from the CLI? Unfortunately, there's no good default editor in the Powershell terminal. But we can launch our Notepad++ text editor from the CLI and begin modifying text that way.

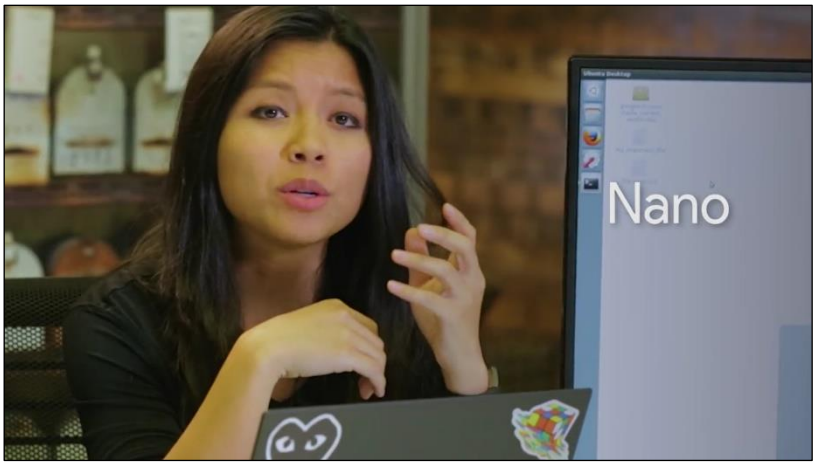
So start, Notepad++, and then just a filename. As you can see, it opened up Notepad++, and asked if I wanted to create this file. If you'd like to read about text editors that you can specifically use in the CLI, check out the supplemental reading on an advanced text editor called Vim.





1.3.4 Linux Modifying text files

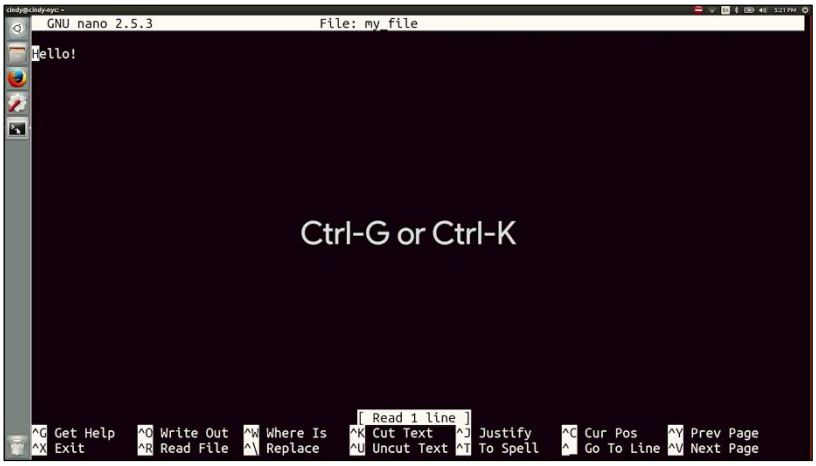
In Linux, there are many popular text editors that we can use to modify files. We won't have enough time to cover them all. So let's just focus on one editor that can be found on virtually any distribution, Nano.



Nano is an extremely lightweight but useful text editor. We've included it in the supplementary readings after this video, so go check it out. To edit a file in Nano, just type Nano then the file name.

Once we do that, we'll be launched into the Nano program. From here, we can start editing content as we normally would with any other text editor.

At the bottom of the screen, you'll notice a few options like caret G and caret K. The caret means to use Ctrl-G or Ctrl-K. We won't talk about all these options, but a few that might be useful are Ctrl-G, which helps open up a help page, and Ctrl-X which is used when you want to save your work or exit Nano.



Let's go ahead and edit this file, then save our changes. It's asking me if I want to save the file or exit and discard my changes. I'm just going to hit Y because I want to save them.

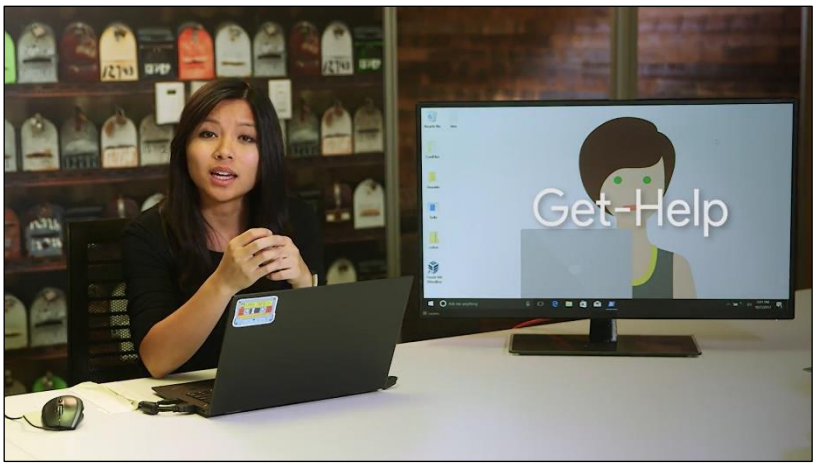
Once I do that, I'll be exited from Nano. Let's verify we actually changed that file. There it is.

Nano is a super useful tool if you need a quick text editor in Linux. But if you want to be a true OS power user, I recommend that you read the supplemental material I've included to learn more about the text editors that are used in the industry, like Vim or Emacs.

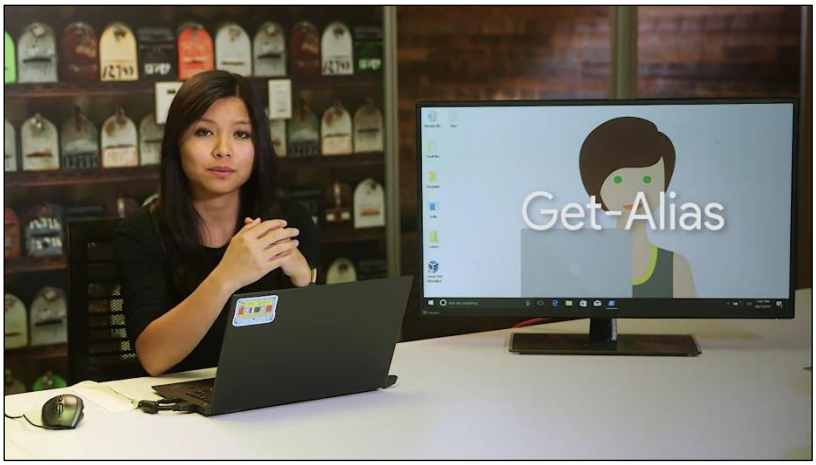
1.3.5 Windows Powershell

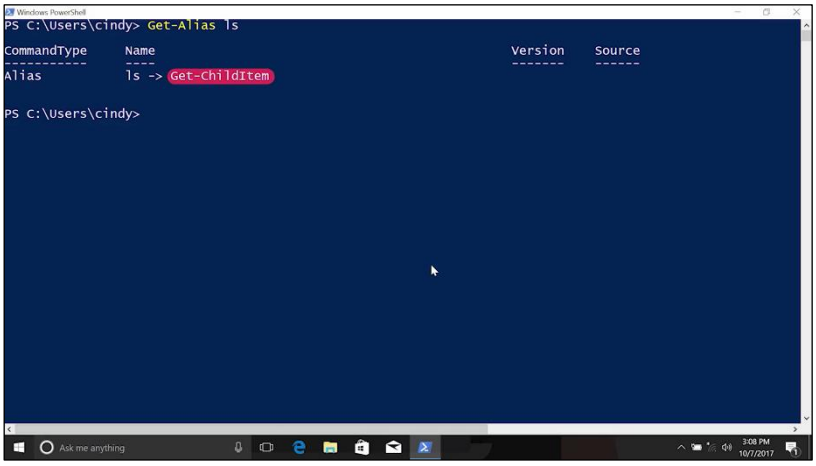
So far in this course we have been using command aliases in PowerShell. PowerShell is a complex and powerful command language, that's also super robust. We've been able to use common aliases, that are exactly the same as their Linux counterparts.

But from here on out, we'll need to deploy some advanced command line features, so we'll need to look at real PowerShell commands. You've already seen an example of a real PowerShell command, Get-Help, which is used to see more information about commands.



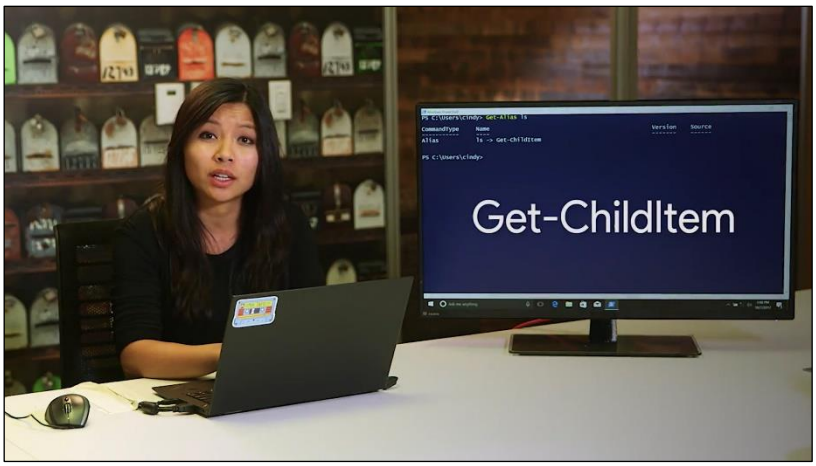
There's another PowerShell command that we can use to look at one of our aliases, that we've been using as our list directory. To see what the actually PowerShell command is that gets executed, we can use the PowerShell command, Get-Alias.

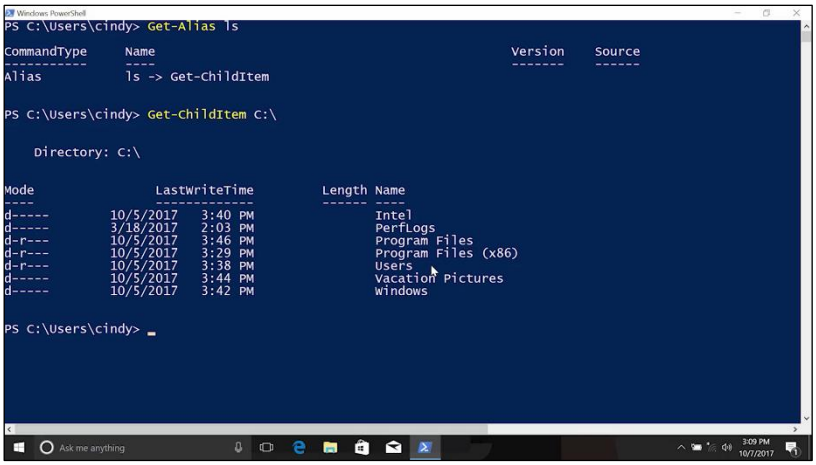


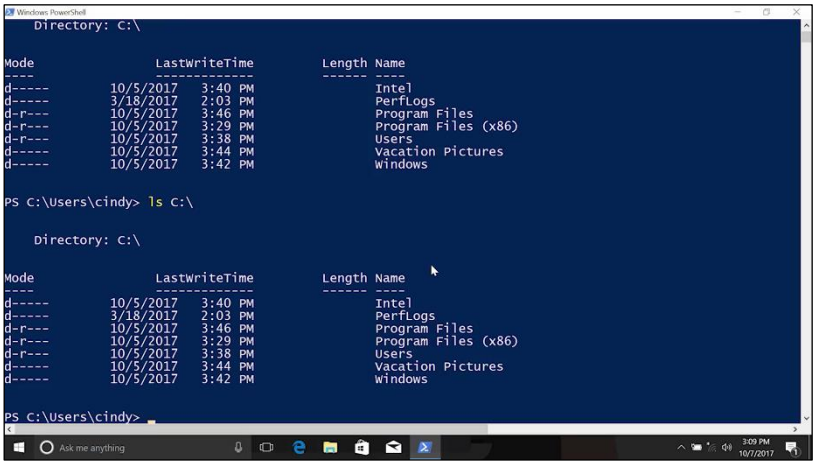


Interesting when we call LS, we are calling the PowerShell command Get-ChildItem, it gets or lists the children which are the files and sub directories of the given item.

Let us actually run this Get-ChildItem command with the item C:\. You'll see this is the same output as, ls C:\. Cool.







PowerShell commands are very long and descriptive, which makes them easier to understand. But it does mean a lot of extra typing, when you're working interactively at the CLI. Aliases for common commands are a great way to work more quickly in PowerShell. We've been using them up to this point to help us hit the ground running with the command line.

In Windows, you pretty much have three different ways you can execute commands. You can use real PowerShell commands, or the relatable alias names. Another method that we've mentioned, but haven't really talked about yet is cmd.exe commands.

Cmd.exe commands are commands from the old MS-DOS days of Windows. But they can still be run do to backwards compatibility.

Keep that in mind, that they aren't as powerful as PowerShell commands. An example of a cmd.exe command is dir. Which coincidentally points to the PowerShell command Get-ChildItem, which is also where, ls Alias gets pointed to.

PowerShell	Alias	cmd.exe
Get-ChildItem	ls	dir

Remember the PowerShell command Get-Help, well there's a command parameter that you can use to get help with command.ext commands, /?.

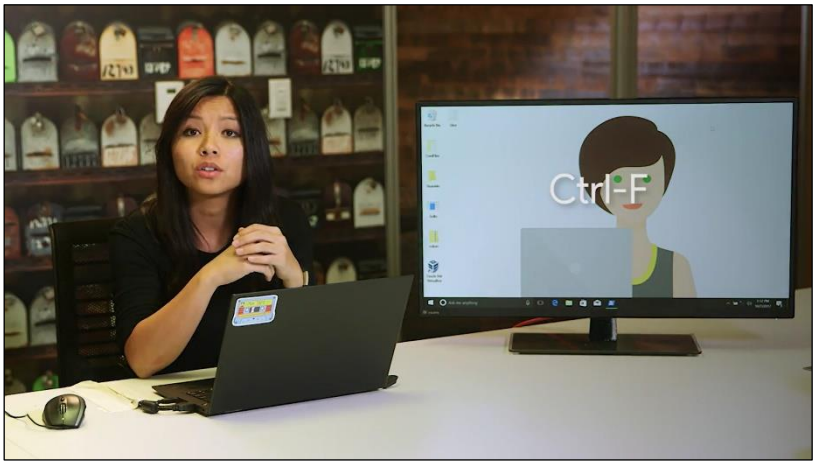
Get-Help is used for PowerShell commands like **Get-Help ls**, and **/?** is used for other commands like **dir** such as **dir /?**.

Keep the difference in mind, Get-Help is used for PowerShell commands like Get-Help ls, and /?, is used for other commands like dir/?. If I tried to use, ls/?, it will return nothing, because the PowerShell command that ls is an alias of, doesn't know how to handle to the parameter /?, and vice versa.

You're free to use whatever commands you feel comfortable with. But in this course we're going to use common aliases, and PowerShell commands.

1.3.6 Windows Searching within files

You've probably had to search for words in a text document before. Whether it was to find and replace words or for something else. Most text editors work the same way when it comes to finding words in the document. All you need to do is Ctrl+F to search for the word. Pretty simple right?

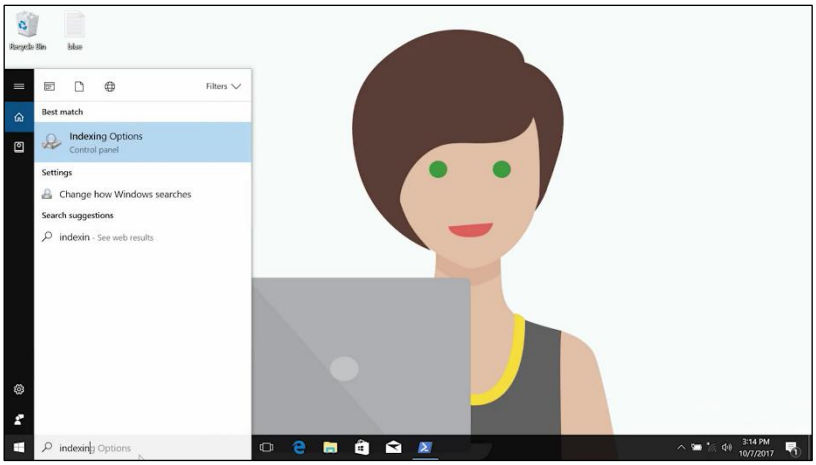


But what if you wanted to see if a word existed in multiple files? There are a few ways we can do this. Let's talk about the GUI options and then we'll turn to PowerShell and learn how to search for words from the CLI.

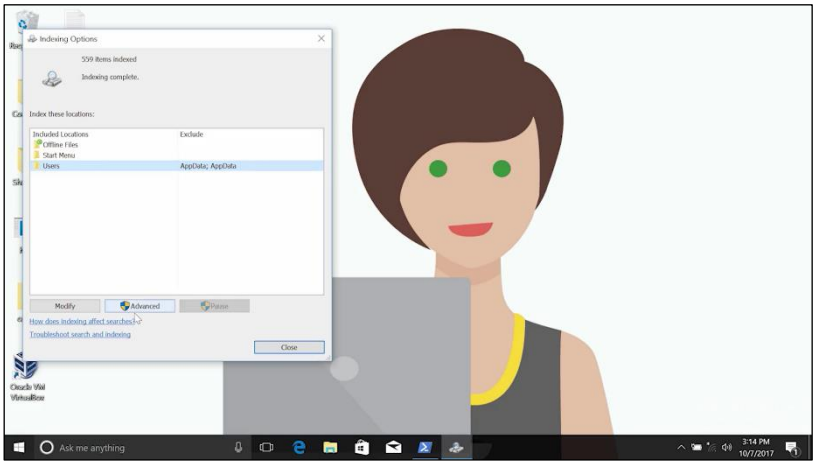
Windows has a service called the Windows Search Service. This service indexes files on your computer by looking through them on a schedule. It then compiles a list of names and properties of the file that it finds into a database. This is a time consuming and resource intensive process. So on many Windows Servers, those search service isn't installed or is disabled. On Windows 8 and Windows 10 desktop computers, It's often enabled for files in your home directory, but not for the entire hard drive. By default, the Windows Search Service will let you find files based on their name, path, the last time they were modified, their size, or other details, but by default you can't search for words inside the files.

The Windows Search Service can be configured to search file contents and their properties. This increases the amount of time that it takes for the indexer to do its work. It's sort of like the computer is doing all of the searches that you might want to do ahead of time and then you just have to look up the result. Let's configure the service to index file contents and see what it looks like.

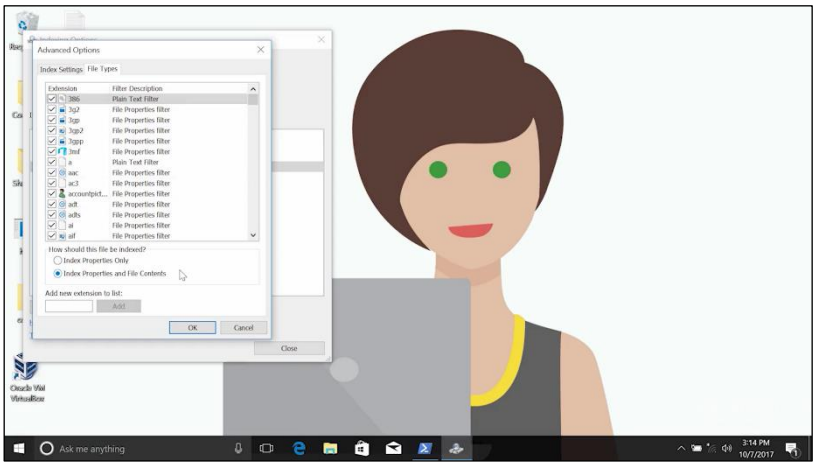
The settings we're looking for are in the Control Panel, but we can use the Start menu to find the settings we need faster. Open the Start menu and then type indexing. You'll see the Indexing Options in your results of the search, click on that.



Now you want to change the settings for the user folder which is where all the home directories are stored. Select Users and then click Advanced.



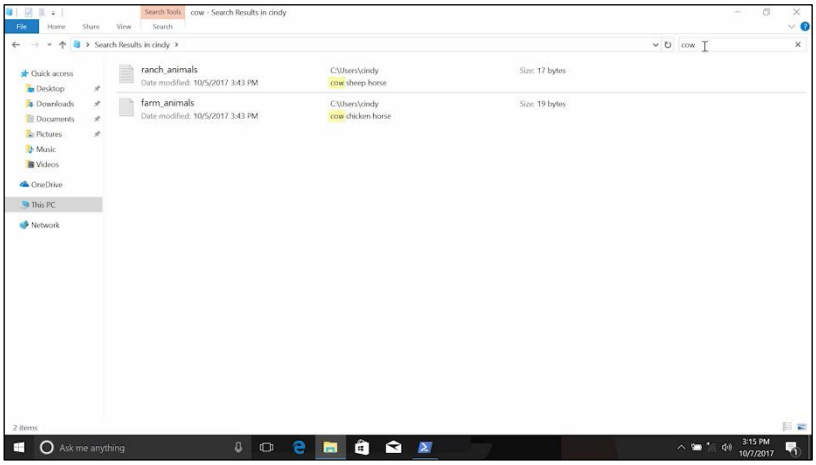
Now select the File Types tab, and select Index Properties and File Contents. Click Okay.



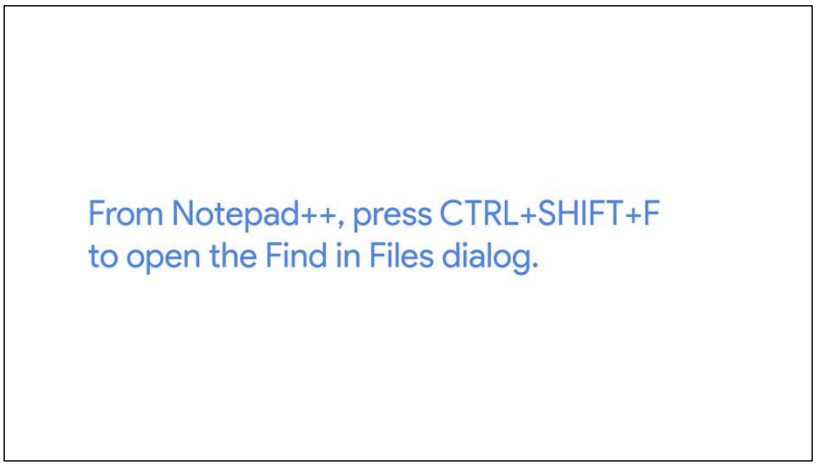
Now close out of the indexing options.

When you do this, the Windows Search Service will start to rebuild the index based on your new settings. This could be super fast or could take while. It all depends on how many files you have and how large they are.

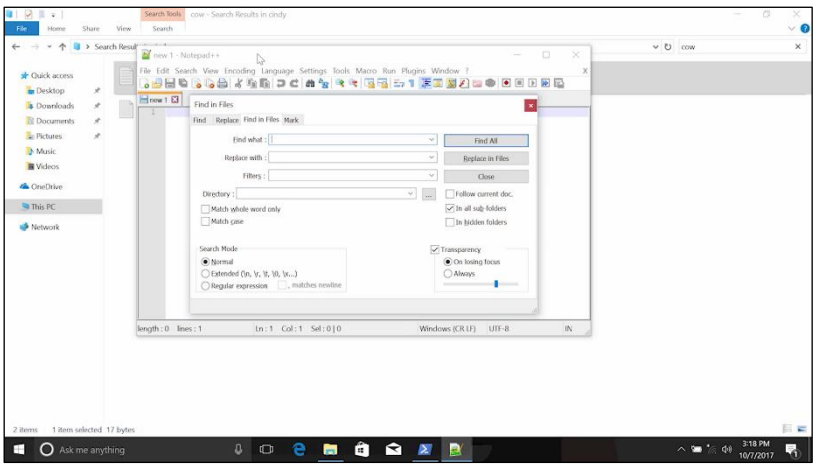
On this system, I've already let the re-indexing complete. Now I can use Windows Explorer, my home directory, to find files that have a specific word in them. Let's search for the word cow. The results turn up farm animals and ranch animals dot text.

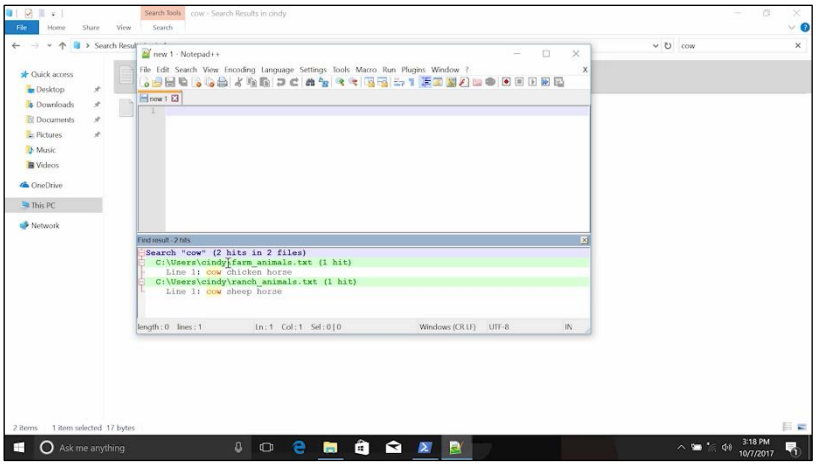


Awesome, we can see the word cow in this text file. If you don't want to use the Windows Search Service, we can also use Notepad++, the editor that we installed in an earlier lesson. From Notepad++, press Ctrl+Shift+F to open the Find in Files dialog.

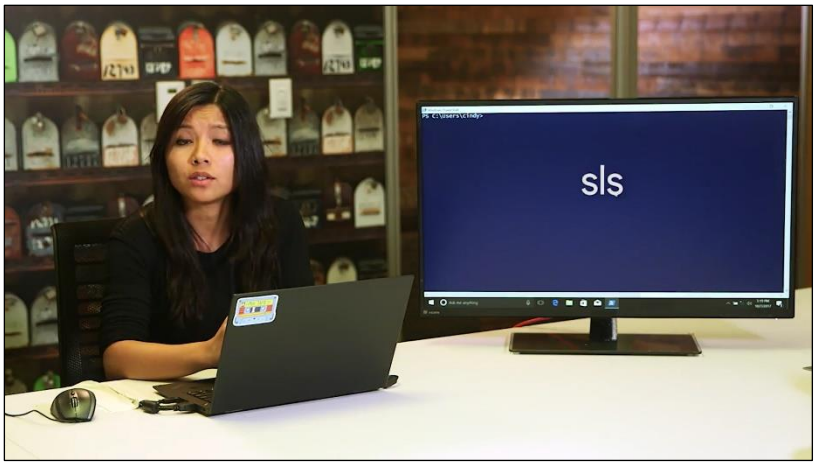


From here, we can specify what you want to find and what files you want to search. You can limit your search to a specific directory, to a specific set of file extensions and you can even actually replace the word with another one from here. So lets search for the word cow again and this time I'll search on my home directory. Find all, there we go. Now it returns farm animals and ranch animals.

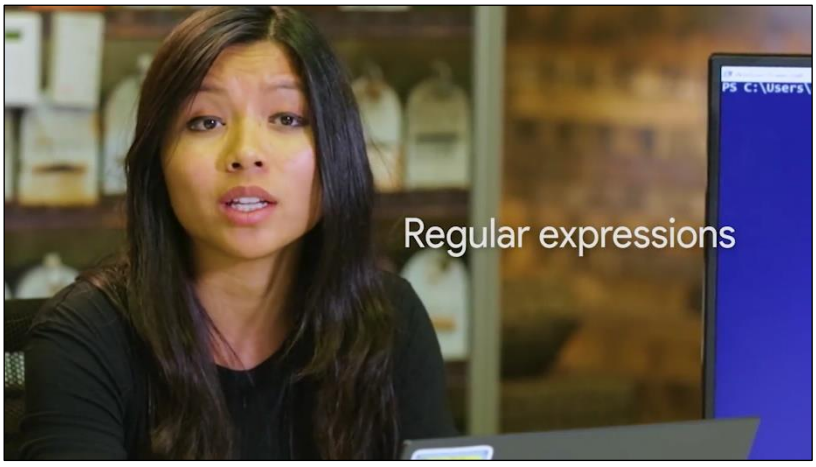




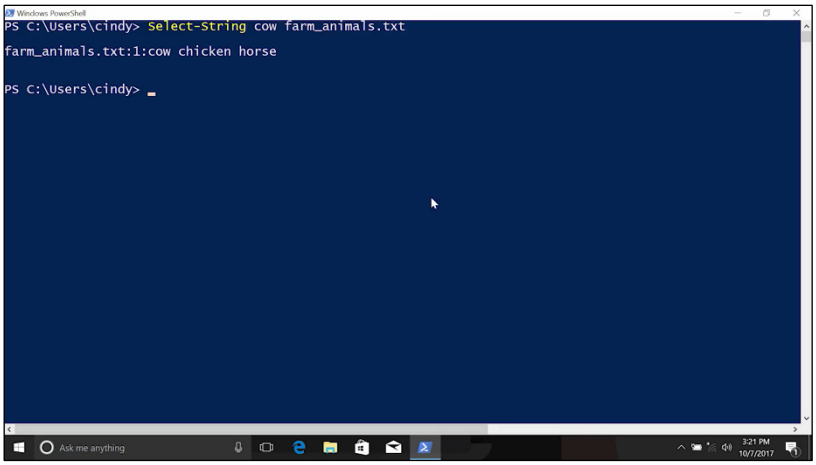
If we can't or don't want to use a GUI, we can search for words within files from the command line. In PowerShell, we're going to use the SLS or Select-String command to find words or other strings of characters and files. You can think of strings as a way for the computer represent text.



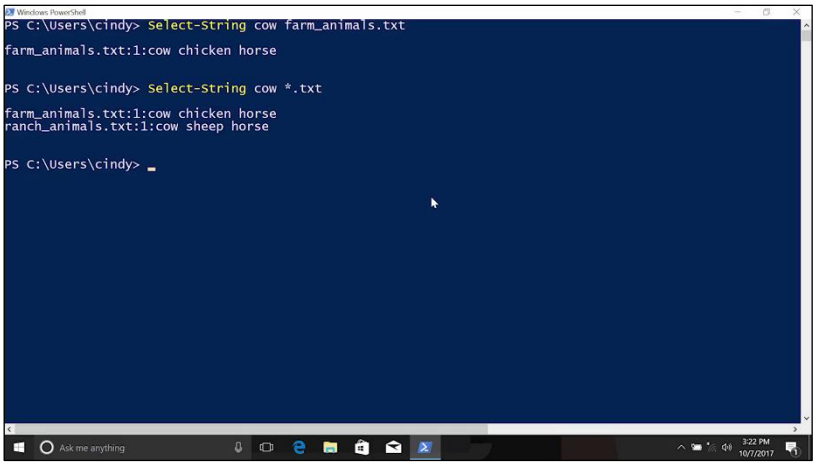
The Select-String command lets you search for text that matches a pattern you provide. This could be a word, part of a word, a phrase or more complicated patterns that are described using a pattern matching language called Regular Expressions. Keep in mind that this is a really powerful capability that we're just scratching the surface of. So here we're going to search for a word in a file in my home directory.



Let's search for the word cow again. You'll see that Select-String found cow and it tells you the file and the line number where it found it.



Excellent, if you wanted to search through several files in a directory, you can use pattern matching to select them. Remember the wildcard character asterisk for selecting all, we can use that here as well. Now we can see that it found farm animals and ranch animals.

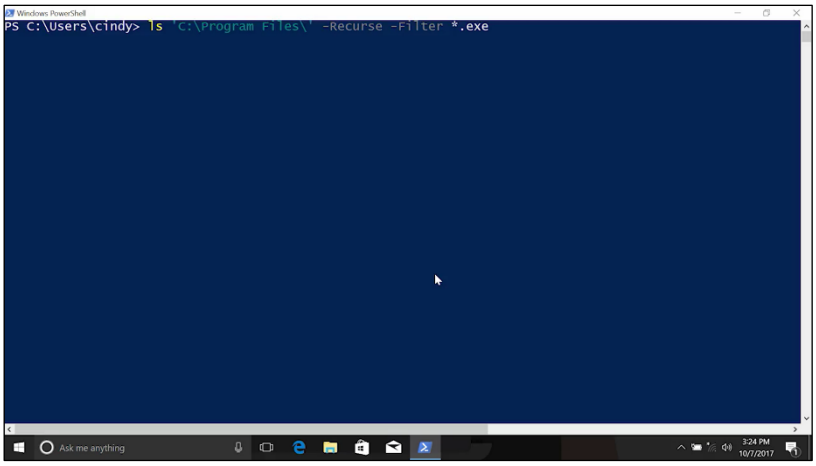


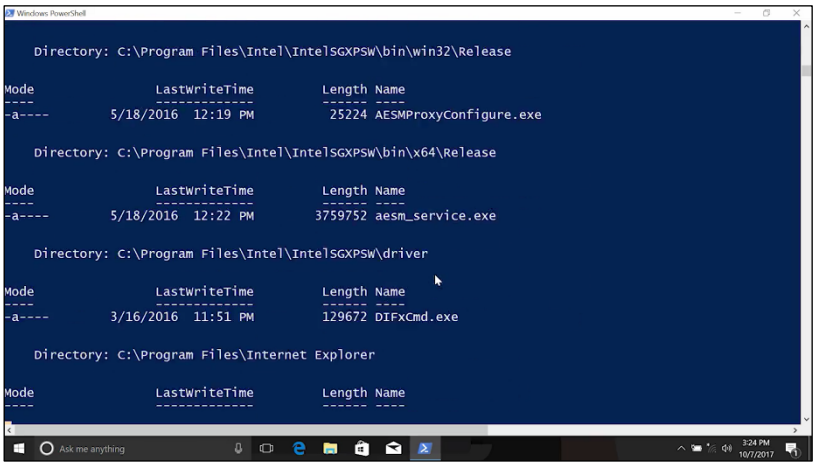
Select-String can do lots of other things too. We'll get a chance to see that in later lessons. Being able to find a string in a file or a set of files, it's going to be a critical skill for you on this course and in your IT support work. It's also an important tool that we're going to learn to combine with other tools to do really powerful things from the CLI.

1.3.7 Windows Searching within directories

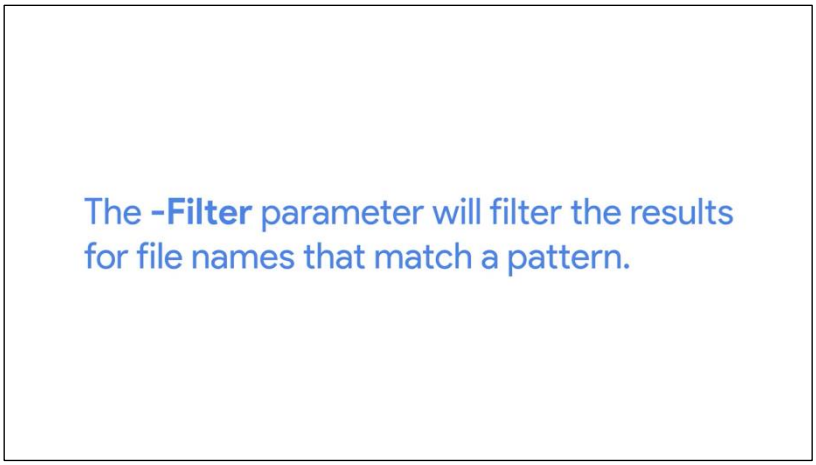
What if we wanted to search for something within a directory, like looking for just the executables in that same directory. This is where the command parameter -filter comes in.

I'm just going to LS my programs files here with the -recurse, -filter and look for exes.

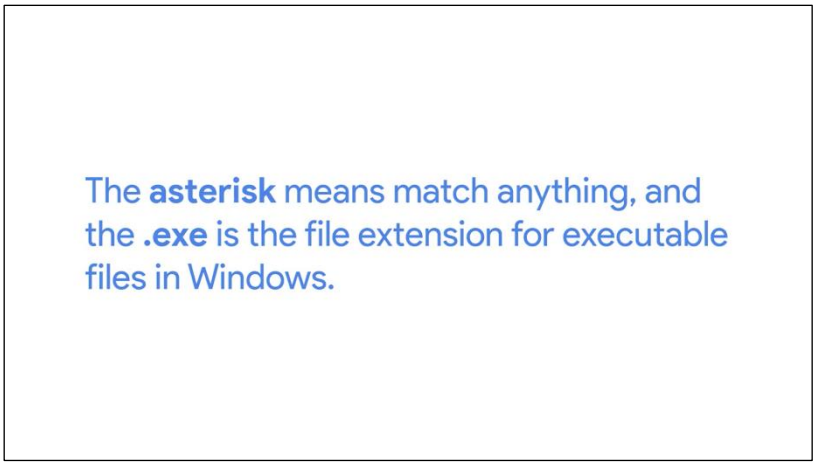




Well, that's lots of exes. The -filter parameter will filter the results for file names that match a pattern.



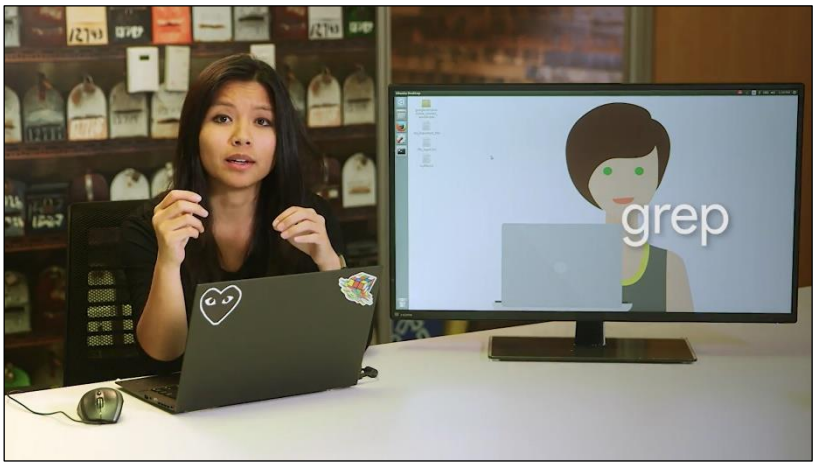
The Asterisk means match anything. And the.exe Is the file extension for executable files in windows.



So the only results we're going to get are the files that end in.exe. Cool.

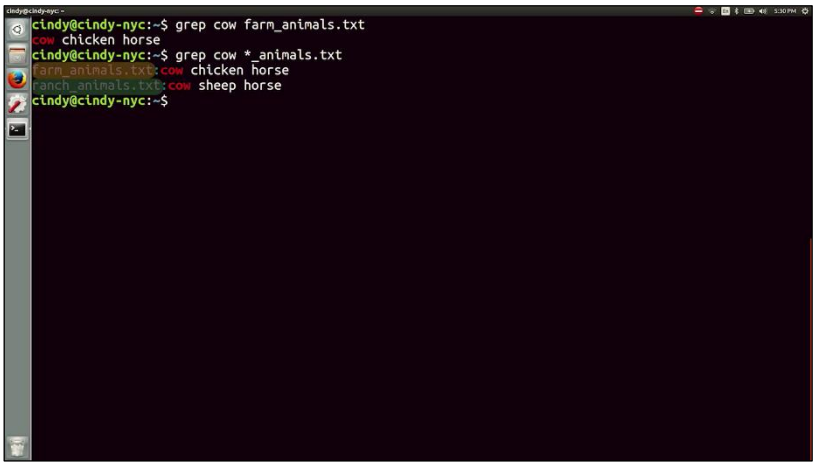
1.3.8 Linux Searching within files

In Bash, we can search for words within files that match a certain pattern using the grep command.



What if you wanted to know of a certain file existed in a directory or if a word was in a file? Similar to the PowerShell select-string command, we can use the grep command in Bash. Let's search for the word cow in farm animals. You'll see that grep found cow in the text file, farm animals.

You can also use grep to search through multiple files. Let's use the asterisk wildcard command here. And you can see that it found cow in farm animals and ranch animals. You'll be using grep a lot throughout this course and in later courses, so it's an important command to remember.

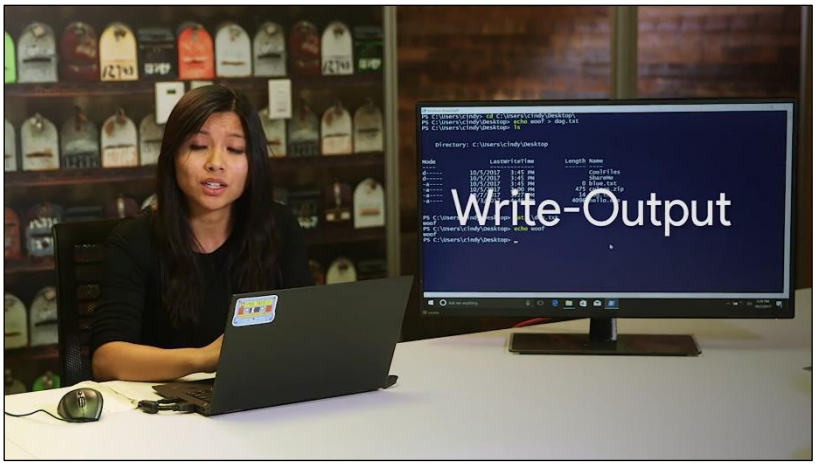


1.3.9 Windows Input output and the pipeline

All right, we've learned a bunch of individual, very powerful tools. These are the most important day-to-day commands that you'll need to work in PowerShell. Now, we're going to learn how to combine these tools to make them even more powerful. Let's run the following command in our desktop directory. Then we'll break it down piece by piece. Scan cd into my desktop directory. Okay, I go woof > dog.txt. Will do an LS to check our desktop, and we'll now see a file called dog.txt. Inside that file, we should see the word, woof. Oh, there it is. What's happening here? Let's take a closer look, echo woof.

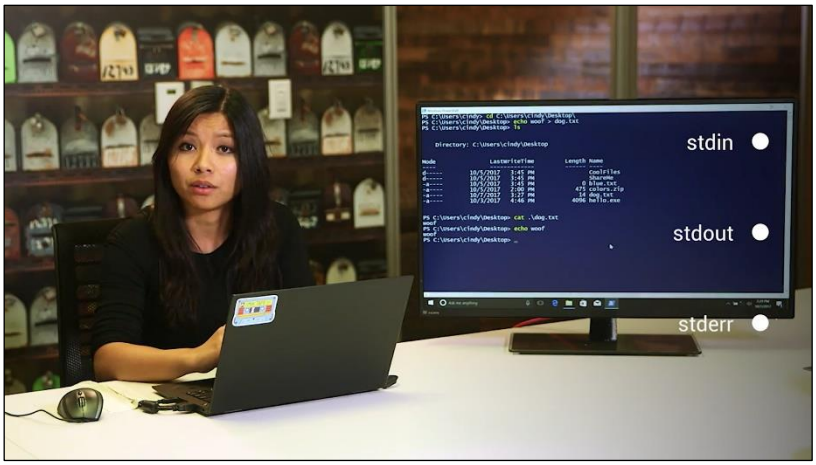
```
PS C:\Users\cindy> cd C:\Users\cindy\Desktop\
PS C:\Users\cindy\Desktop> echo woof > dog.txt
PS C:\Users\cindy\Desktop> ls


    Directory: C:\Users\cindy\Desktop


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        10/5/2017   3:45 PM                CoolFiles
d-----        10/5/2017   3:45 PM                ShareMe
-a----        10/5/2017   3:45 PM              0 blue.txt
-a----        10/5/2017   2:00 PM            475 colors.zip
-a----        10/7/2017   3:27 PM             14 dog.txt
-a----        10/3/2017   4:46 PM           4096 hello.exe


PS C:\Users\cindy\Desktop> cat .\dog.txt
woof
PS C:\Users\cindy\Desktop> echo woof
woof
PS C:\Users\cindy\Desktop>
```

In PowerShell, the echo is actually an alias for Write-Output.



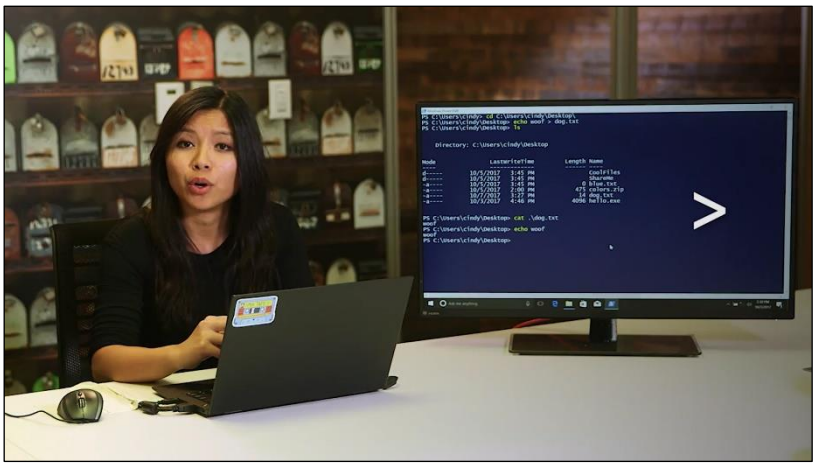
That gives us a clue to what's happening. We know the echo command prints out our keyboard input to the screen. But how does this work? Every Windows process and every PowerShell command can take input and can produce output. To do this, we use something known as I/O streams or input output streams.

Each process in Windows has three different streams: standard in, standard out, and standard error. It's helpful to think of these streams like actual water streams in a river. You provide input to a process by adding things to the standard in stream, which flows into the process. When the process creates output, it adds data to the standard out stream, which flows out of the process.

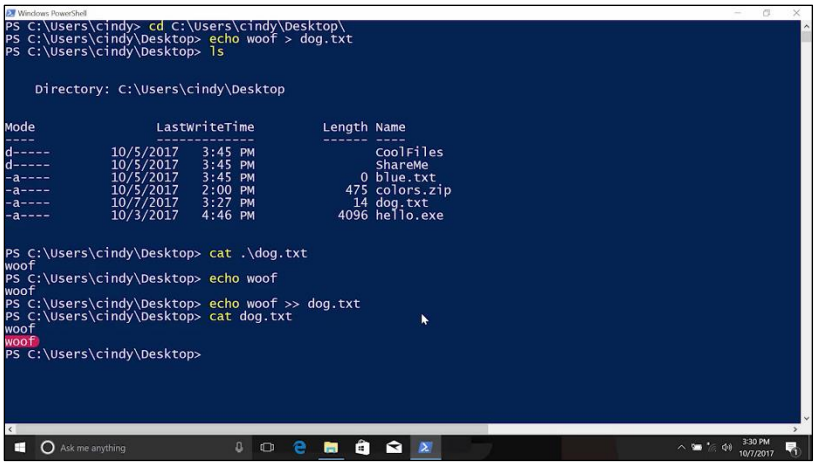


At the CLI, the input that you provide through the keyboard goes to the standard in stream of the process that you're interacting with. This happens whether that's PowerShell, a text editor, or anything else. The process then communicates back to you by putting data into the Standard out stream, which the CLI writes out on the screen that you're looking at.

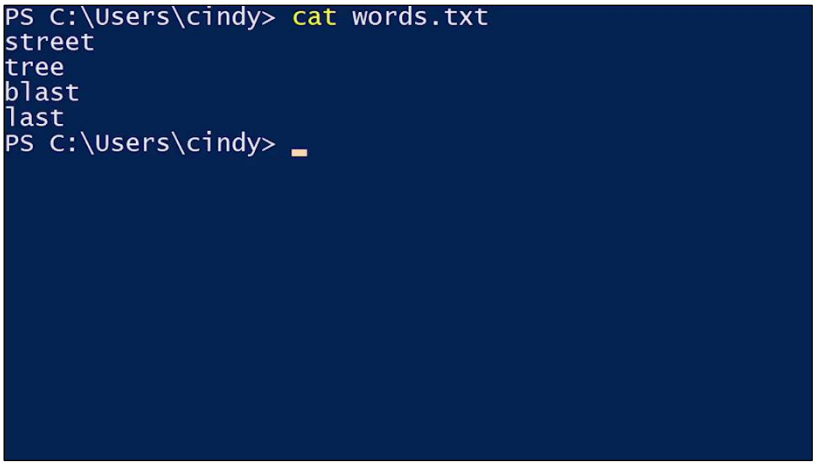
Now, what if instead of seeing the output of the command on the screen, we wanted to save it to a file? The greater than symbol is something we call a redirector operator that lets us change where we want our standard output to go.



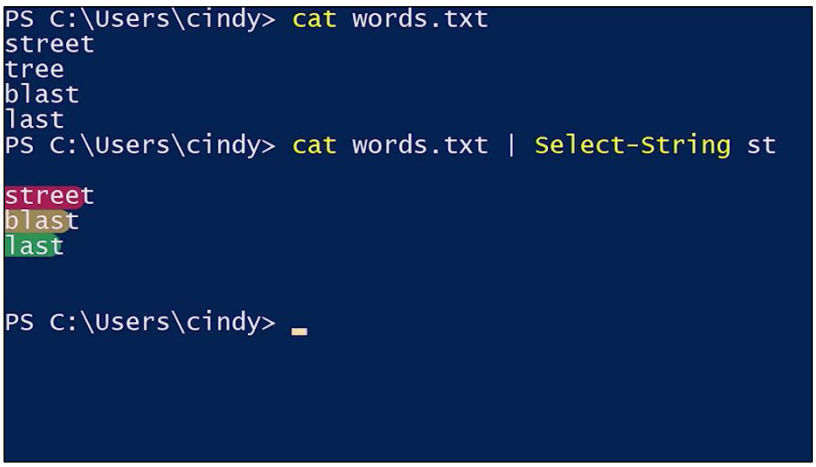
Instead of sending standard out to the screen, we can send a standard out to a file. If the file exists, it'll overwrite it for us. Otherwise, it'll make a new file. If we don't want to overwrite an existing file, there's another redirector operator we can use to append information, greater than, greater than. So let's see that in action, echo woof >> dog.txt. Now, if I look at my dog.txt file again, we can see that woof was added again.



But, what if we wanted to send the output of one command to the input of another command? For this, we're going to use the pipe operator. First, let's take a look at what's in this file. cat words.txt. Look at that, it's a list of words.



Now, what if we want to just list the words that contain the string st? We can do what we've done before and just use select-string or SLS on the file directly. This time, let's use the pipeline to pass the output of cat to the input of select-string. So cat words.txt | select-string st. And now, we can see a list of words with the string st.



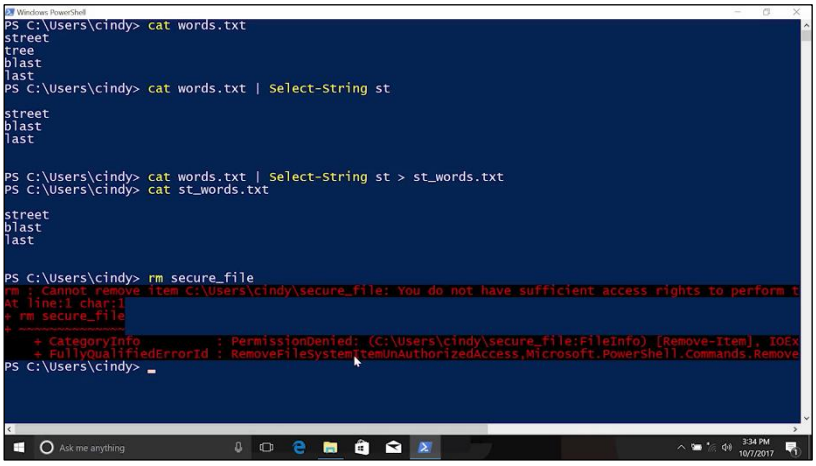
To tie things together, we can use output redirection to put our new list into a file. So now, greater than, and then a new file called st words.txt. Now, if I cat st words.txt, yup, there it is.

```
PS C:\Users\cindy> cat words.txt
street
tree
blast
last
PS C:\Users\cindy> cat words.txt | Select-String st

street
blast
last


PS C:\Users\cindy> cat words.txt | Select-String st > st_words.txt
PS C:\Users\cindy> cat st_words.txt

street
blast
last


PS C:\Users\cindy>
```

That's just a very basic example of how you can take several simple tools and combine them together to do complex tasks.

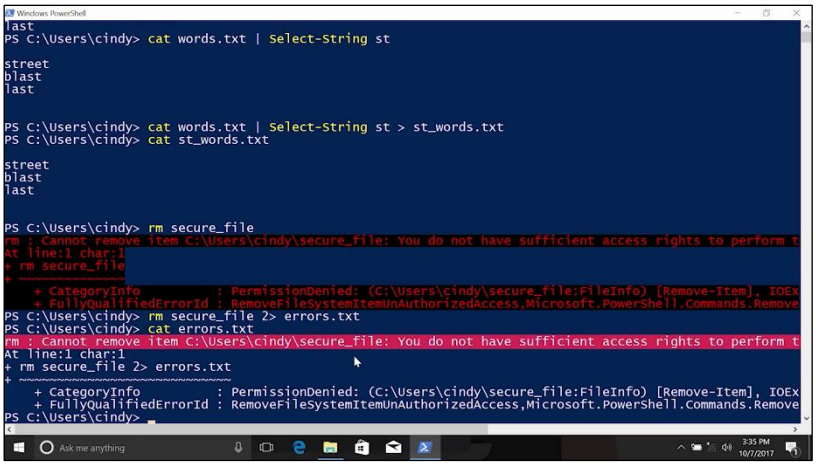
Okay, now we're going to learn about the last I/O redirector, standard error.

Remember when we tried to remove a restricted system file earlier and we got an error that said permission denied? Let's review that once more. This time, I'm going to remove another protected file, rm secure_file. We see errors like we're supposed to. But what if we didn't want to see these errors?

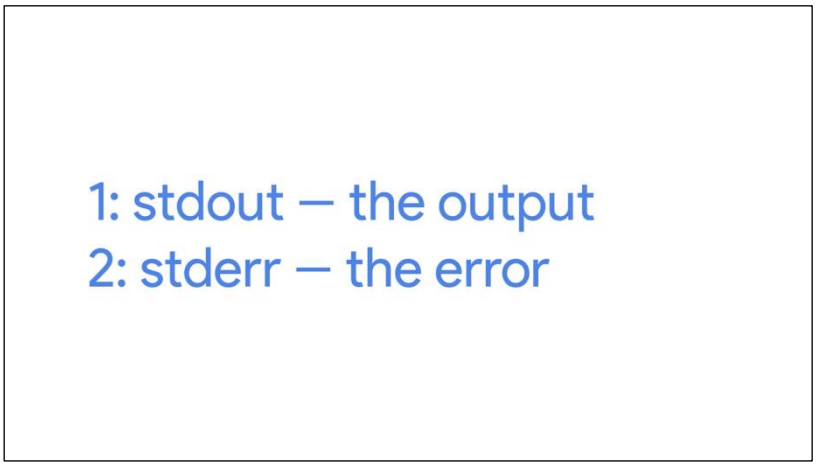


Turns out, we can just redirect the output of error messages in a different output stream called standard error. The redirection operator can be used to redirect any of the output streams, but we

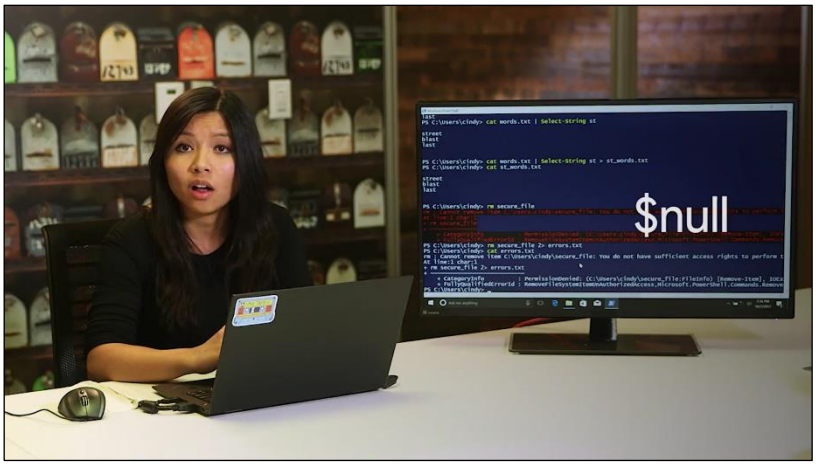
have to tell which stream to redirect. So, let's type, rm secure_file 2> errors.txt. If I look at errors.txt, I can see the error message that we just got.



So, what does the two mean? All of the output streams are numbered. One is for standard out, which is the output that you normally see, and two is for standard error or the error messages.

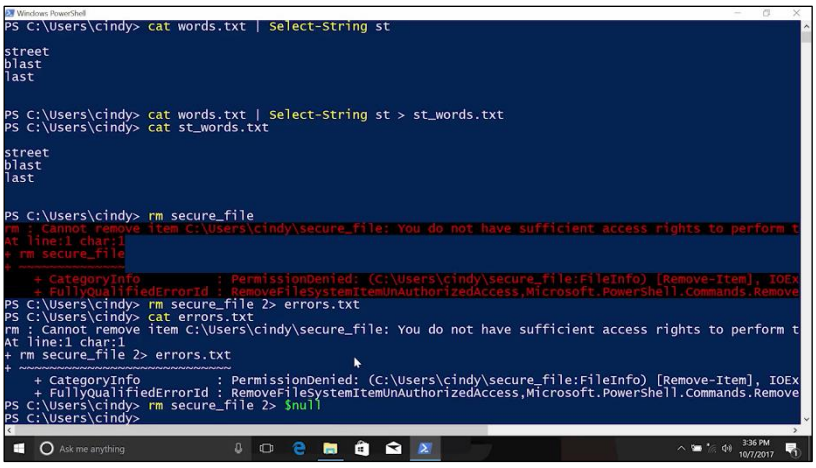


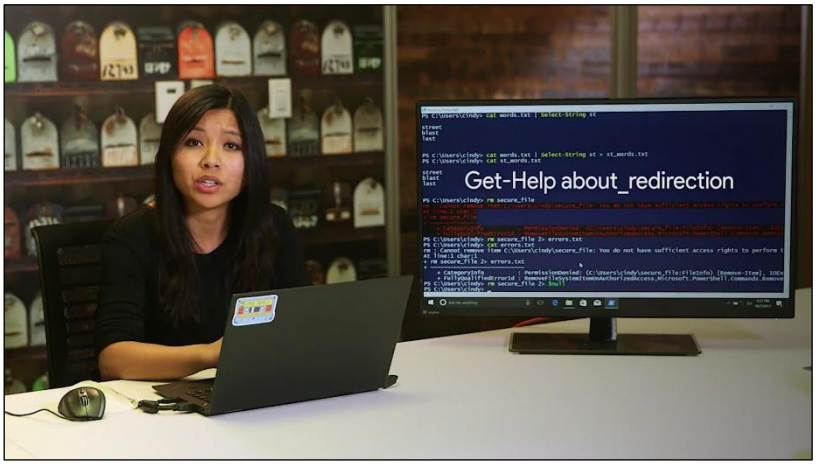
Heads up, PowerShell actually has a few more streams that we aren't going to use in this lesson. But they can be redirected in the same way. You can read more about them in the supplemental reading right after this video. So when we use two greater than, we're telling PowerShell to redirect the standard error stream to the file instead of standard out. What if we don't care about the error messages, but we don't want to put them in a file? Using our newly learned redirector operators, we can actually filter out these error messages. In PowerShell, we can do this by redirecting standard error to $null.



What's $null? Well, it's nothing. No, really. It's a special variable that contains the definition of nothing. You can think of it as a black hole for the purposes of redirection.

So let's redirect the error messages this time to $null, rm secure_file 2> $null. Now, our output is filtered from error messages. There's still much more to learn if you're interested. Try Get-Help about_redirection in PowerShell to see more detail.

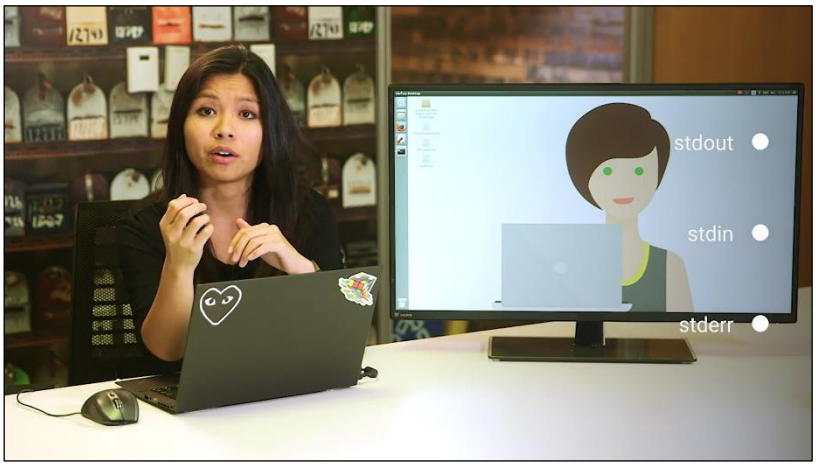




It may take a little time to get the hang of using redirector operators. Don't worry, that's totally normal. Once you do start to get used to them, you'll notice your command full skills level up and your job becomes a little easier. Now, let's take a look at output redirection in Linux.

1.3.10 Linux Input output and the pipeline

Similar to Windows, we have three different I/O or input-output streams: standard out, standard in and standard err.



Remember the standard out example in the last lesson? Well, the same concept applies in Linux. We echo the text woof here, but instead of sending it to our screen by default, we're going to redirect the output to a file using the standard out redirector operator. Let's verify and there it is. This overrides any file named dog.text with the content woof.

```
cindy@cindy-nyc:~/Desktop$ echo woof > dog.txt
cindy@cindy-nyc:~/Desktop$ cat dog.txt
woof
cindy@cindy-nyc:~/Desktop$
```

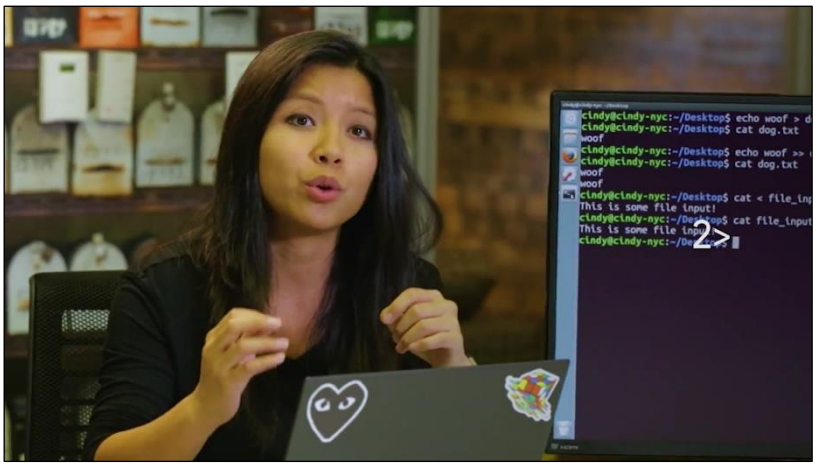
If we don't want to overwrite an existing file, we can use the append operator or greater than greater than. So, echo woof, dog.text. We could verify that. There it is.

```
cindy@cindy-nyc:~/Desktop$ echo woof > dog.txt
cindy@cindy-nyc:~/Desktop$ cat dog.txt
woof
cindy@cindy-nyc:~/Desktop$ echo woof >> dog.txt
cindy@cindy-nyc:~/Desktop$ cat dog.txt
woof
woof
cindy@cindy-nyc:~/Desktop$
```

One redirector operator that we talked about in the Windows lesson, but didn't show an example of, was the standard in redirector operator. The standard in redirector is denoted by a less than sign. Instead of getting input from the keyboard, we can get input from files like this. This command is exactly the same as cat file_input. The difference here is that we aren't using our keyboard input more, we're using the file as standard in.

```
cindy@cindy-nyc:~/Desktop$ echo woof > dog.txt
cindy@cindy-nyc:~/Desktop$ cat dog.txt
woof
cindy@cindy-nyc:~/Desktop$ echo woof >> dog.txt
cindy@cindy-nyc:~/Desktop$ cat dog.txt
woof
woof
cindy@cindy-nyc:~/Desktop$ cat < file_input.txt
This is some file input!
cindy@cindy-nyc:~/Desktop$ cat file_input.txt
This is some file input!
cindy@cindy-nyc:~/Desktop$
```

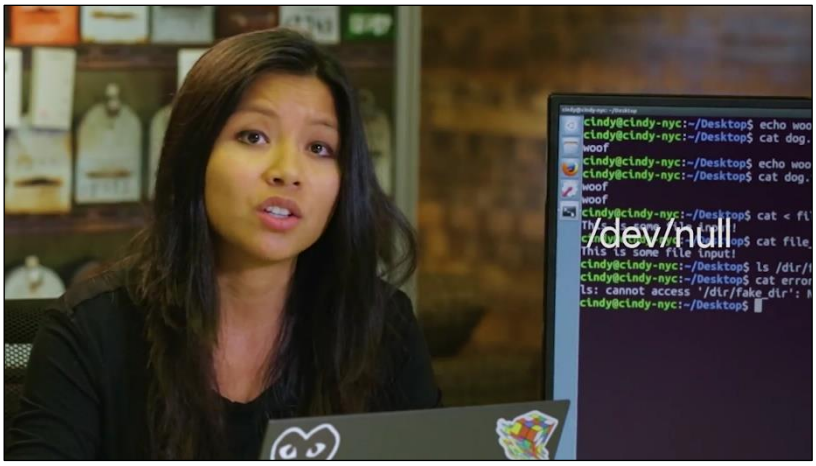
Finally, similar to Windows, the last redirector operator we'll talk about is standard err. Standard err displays error messages which you can get by using the two greater than, redirector operator. Just like Windows, the two is used to denote standard err. So, to redirect just the error messages of some output, you can use something like this, ls/ dir/ fake_dir 2> error_output.text.

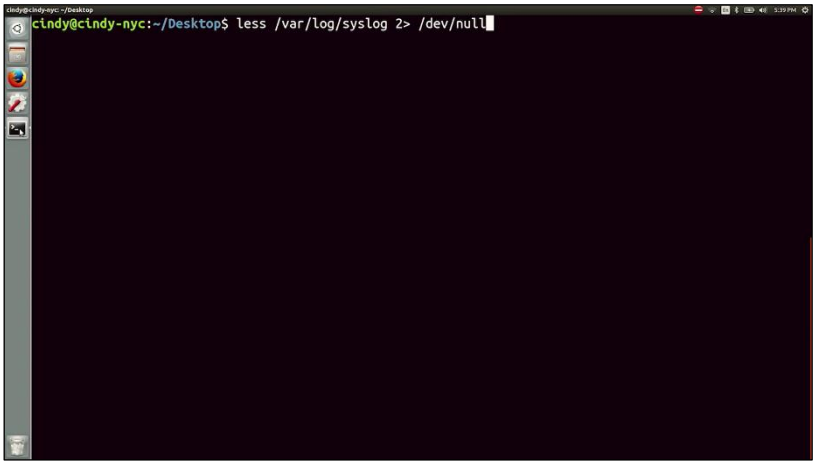


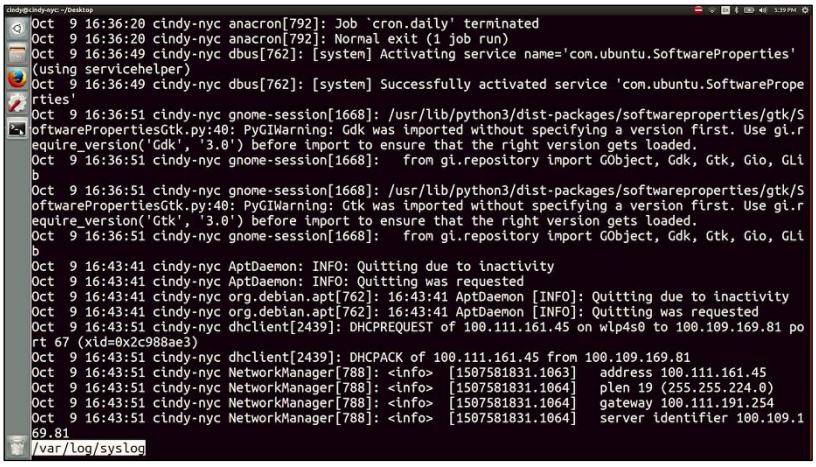
Now, if I view that, new document. Now, we can see the error message in error output.text.

```
cindy@cindy-nyc:~/Desktop$ echo woof > dog.txt
cindy@cindy-nyc:~/Desktop$ cat dog.txt
woof
cindy@cindy-nyc:~/Desktop$ echo woof >> dog.txt
cindy@cindy-nyc:~/Desktop$ cat dog.txt
woof
woof
cindy@cindy-nyc:~/Desktop$ cat < file_input.txt
This is some file input!
cindy@cindy-nyc:~/Desktop$ cat file_input.txt
This is some file input!
cindy@cindy-nyc:~/Desktop$ ls /dir/fake_dir 2> error_output.txt
cindy@cindy-nyc:~/Desktop$ cat error_output.txt
ls: cannot access '/dir/fake_dir': No such file or directory
cindy@cindy-nyc:~/Desktop$
```

Remember the dollar sign null variable that we used in Windows to toss unwanted output into a metaphorical black hole? We have something like that in Linux too. There's a special file in Linux called the /dev/null file. Let's say we want to filter out the error messages in a file and just want to see standard out messages. We could do something like this.



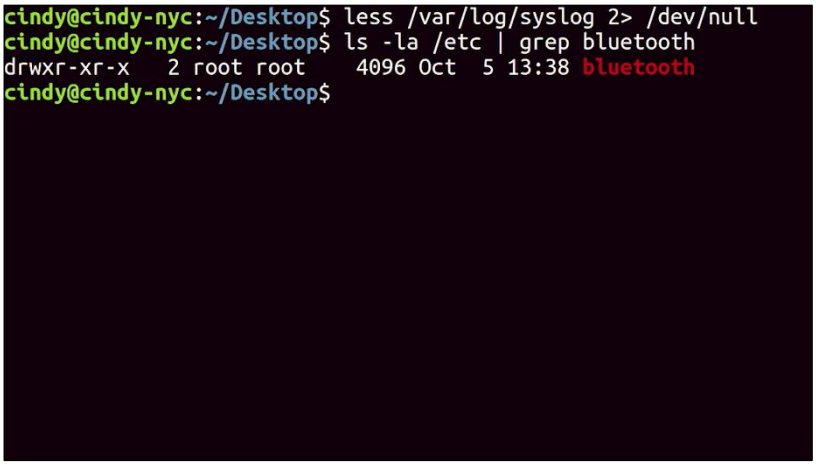




Now, our output is filtered from error messages.

Remember how we talked about taking the output of one command and using it as the input of another command, with the Windows pipeline? Well, the same thing exists in Linux. The pipe command allows us to do this.

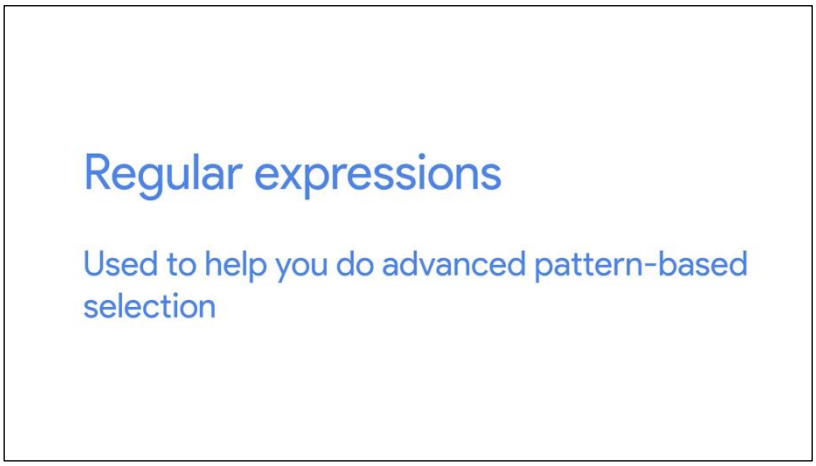
Let's say we want to see which sub-directories in the slash etc directory contain the word Bluetooth. We can do something like this. We're using the pipe redirector to take the output of ls-la/etc and pipe or send it to the grep command. Now, without even looking through the directory, we're able to quickly see if the Bluetooth directory is in here. There it is.



You've gotten a glimpse of the power of redirectors and as you dive deeper into the world of Linux, you'll be using them at regular basis. They're super valuable tools to have and now, they're part of your toolkit.

1.3.11 Windows and Linux Advanced navigation

You've learned a lot of commands and tools to help lay a strong foundation for IT support work. There are many other commands that you haven't seen yet. Don't worry, we'll get to them as they come up. As you advance in your career, you might even discover that the tools and commands you're using aren't powerful or efficient enough anymore. Maybe you'll want to search through files using more complex patterns.



To do that, you'll need to know about tools like regular expressions. Regular expressions are used to help you do advance pattern based selection. There's also so much more to power shell. There are excellent videos and articles that can guide you from the first steps you've learned here to being a Windows CLI master. If this sounds interesting to you, we really encourage you to check out the supplementary reading right after this video.

And no, we won't grade you on your knowledge of this material in these courses, but it could be really useful to you in the IT support field. You've done some seriously awesome work. We've covered

a lot of information in this lesson. Maybe this was the first time you've been exposed to Linux or Windows. If so, you've already passed a huge milestone in your learning journey. It's super important that you're able to use the commands you learned here by memory. I hope you wrote them down in your notes while watching the videos in this course.

Next up, we'll be testing you on some of the new commands you learned in Bash and Windows CLI. Make sure to re-watch the videos and practice the exercises, if you want a refresher before you start. When you're ready, we'll see you in the next lesson.

1.3.12 Ben first tech job

I have to say, I think, that looking back, I've been really lucky. I had started off even before I was a teenager teaching myself without computers and without even access to them. And somehow, kind of in the end of that era when I was an early teen, I managed to convince a bookstore owner to let me. He needed to buy a computer. And I would program it for him to automate his textbook business. [MUSIC] So I had never actually done that before, right. I had never laid my hands on that kind of computer before. I had never written a program like that before,. And the guy believed me and he did it. And he hired me. That was a part-time job that stuck with me, actually, until I was in my mid-20s. So for like 11 years, I had this part-time job automating the textbook business of a neighborhood bookstore. And the thing that's so amazing, and that I'm so grateful for, is that this guy had this faith in me and put this trust in me. Then I'm even more amazed that it actually kind of worked out, right. And I spent all those years doing it, and it helped their business and all that. But it was a great experience. And it was a great way to kick off my actual, early career as a professional programmer.