Getting Started: Looking Around the Linux/UNIX System

Now that you have an idea of what Linux and UNIX are, it’s time to take the first steps to becoming a user. In this chapter the very first thing you’ll learn is how to connect to a Linux/UNIX system so that you can login. You’ll then learn how to log in and out, and change your password to secure your account. After this you’ll learn the commands used to perform basic user functions like moving around the file system, by changing directories, commands for viewing the files and folders in a directory, and finding out other information about the system and what is happening on the system. You will also learn about the general layout of the Linux and UNIX file system and some of the standard Linux/UNIX directories, those that are similar in function to Users, Program Files, and the Windows or WINNT folders on a Microsoft Windows system.

At the end of this chapter you will be able to:

1. Use ssh to connect to a Linux server, login, change your password and logout.
2. Describe the general layout the Linux/UNIX file system and identify key directories.
3. Explain the concept of a home directory and identify your own home directory.
4. Move around the file system using both relative and absolute paths.
5. Identify your current location in the file system at any time.
6. List the content of a directory.
7. Display basic information about your user account.
8. Identify who is logged on the system and what they are doing.

Connecting and Logging In

The very first step to using Linux or UNIX is to log in to your account. Logging into a Linux or UNIX computer is much like logging into a Windows computer, especially if you've installed and booted Linux on your own computer. If you have the skills to install your own Linux server then I'll assume you're also able to login. But there's also a good possibility that you're going to use an account a Linux or UNIX server someone else is administering, like at work or in a class. If this is the case, and you're using an account on remote server you'll need to open a network connection with the server before logging in.

In this section you'll learn a few of the more common ways to make the network connection, plus a few details about the login process.

Background - telnet and ssh

To login to a Linux/UNIX server you'll first to initiate a network connection with the server. This is done using a protocol called ssh. Here's some quick background on ssh and its predecessor telnet.

In the old days connecting to a remote server to type commands was done using a protocol called telnet. Telnet emulated the old computer terminals which displayed each character you typed so you could see what you were typing, then passed each character you typed to the server so it could process your commands, and finally displayed any output from the server back on the terminal. The old computer terminals were plugged directly into the computer so there was no need to open a network connection. This was fairly secure as the only way an attacker could see the data being transmitted is if they had physical access to the terminal, or the server, or the cable connecting them.

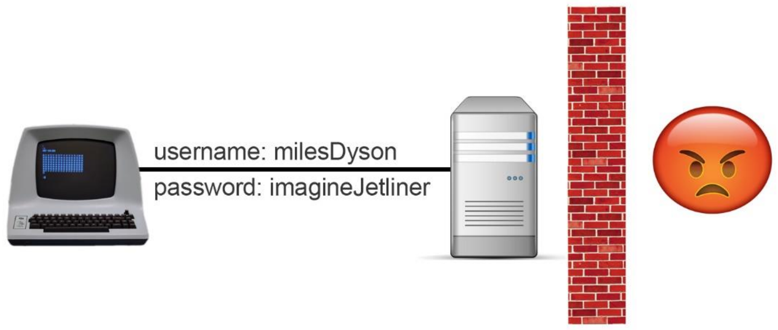


Figure 2.1 Old terminals were directly connected to the server, which protected data from attackers.

As the Internet and TCP/IP became prevalent telnet was developed to perform the same communication between the terminal and the server, except now the connection was across the Internet instead of being sent across a serial cable that was plugged directly into the server.

Telnet functioned fine except it had one issue. That is, telnet did a great job passing commands to the remote server, and receiving and displaying the output but the problem with telnet is that it provided no security. Security wasn't an issue with the old computer terminals as they were plugged directly into the server, so there was no way for anyone to easily intercept the data transmissions. But telnet connections, like any Internet connection, were passed through network routers, each of which could view the data as it passed through the router. This meant that when you entered things like your user name and password, it was easily visible, which was a giant security vulnerability.

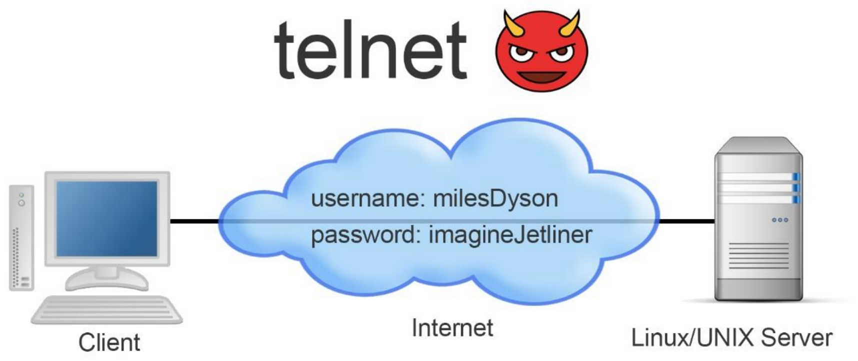


Figure 2.2 Telnet sends data across the Internet in clear text, exposing it to attackers.

To mitigate the security issue, a new protocol called the **s**ecure **s**ocket **s**hell or ssh was developed. Ssh provides the same general function as telnet, passing commands and data between a client terminal and a server, but ssh encrypts all the data to protect it as it passes through the network routers.

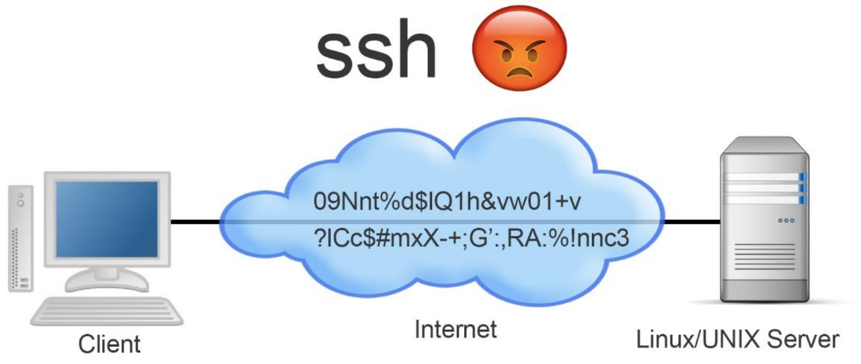


Figure 2.2 ssh encrypts data before transmission, protecting it from attackers.

For those of you who want to know and understand the technical network details, telnet uses port 23 by default, while ssh uses port 22.

Opening an ssh connection to a Linux/UNIX server

Before opening an ssh connection with a server you’ll need to know three things:

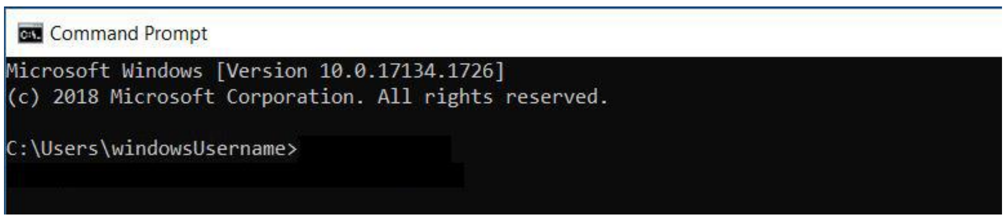
1. The server’s DNS name or IP address
2. Your user name for your account on the server
3. Your password for your account on the server

Once you’re armed with this information you have a few different ways to open an ssh connection to a Linux or UNIX server. Here are the two main ways to open the ssh connection, running ssh in a Command window or using an application.

Running ssh in a Command Window

If you’re running any of the major Operating Systems such as Microsoft Windows, Linux, or the MacOS you can open a Command or Terminal window and use the ssh command that comes with your system. Here’s the process for opening a Command window in Microsoft Windows. (If you need help opening a Terminal Window in Linux or MacOS you can instructions at the book’s web site, or do an Internet search.)

1. Go to the Windows search or type <WIN-S> <⊞-S> and type **cmd**, then select the Command Prompt application.



```
Microsoft Windows [Version 10.0.17134.1726]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\windowsUsername>[redacted]
```

Figure 2.6 The Command Prompt window.

2. This will open a Command Prompt Window. Click inside the window and type:

   ```
   ssh user@serverDNS
   ```

   For example, if you user name is **milesDyson** and the DNS name of the Linux server is **skynet.mil** the command would be:

   ```
   ssh milesDyson@skynet.mil
   ```

   When you do this keep in mind that Linux and UNIX are case sensitive, which means it makes a distinction between upper-case and lower-case characters. This means if your username is **benDover** you must enter it with lower case **ben**, upper case **D** and lower case **over**. If you enter the characters in a different case, for example **BENdover**, the login process will see it as a different user name and not recognize it as valid.

Also note that you probably want to specify the user name. It’s technically possible to leave it off and use the following form of the `ssh` command:

```
ssh serverDNS
```

The problem with omitting the user name is that the Windows 10 version of `ssh` will automatically add one using the name of the currently logged in Windows user. Plus, there’s no way to change the user name after the ssh connection is opened. This means using this form of the command will only work if your Windows user name and your Linux/UNIX user names are identical. But if the user names are different, or if your Windows computer is part of an Active Directory network you won’t be able to login with ssh using this form of the command. And if you’re on an Active Directory network the default user name that will be used with ssh will be userName@directoryName. For example, if you’re logged in to the CyberDyne Active Directory domain as the user **robertBrewster** and try to open an ssh connection to the **skynet.mil** Linux server the user name will be set to **robertBrewster@CyberDyne**, and the full ssh command will end up being:

```
ssh robertBrewster@CyberDyne@skynet.mil
```

3. Issuing the `ssh` command will open a connection to the server. If this is the first time you’ve connected to this server, the server will send an encryption key that will be used to encrypt all the data from this point forward. You’ll be shown some information about the encryption key and asked if you want to save it as shown in the following figure. Make sure and answer **yes**. This saves the encryption key on your client computer. The key will be used to encrypt all data sent to the Linux/UNIX server from this point forward, protecting the data from any attacker that has access to a router between you and the server.

```
C:\>ssh  milesDyson@skynet.mil

The authenticity of host 'milesDyson@skynet.mil (23.38.226.35)'
can't be established.
ECDSA key fingerprint is
SHA256:LDDCB2d1GpzPCI9lBbzhrLIeaUXa9jO3O2VHRcEagE4.

Are you sure you want to continue connecting (yes/no)?
```

Figure 2.7 Example ssh dialog, asking if you want to save and use the encryption key sent from the Linux/UNIX server.

If you answer **no** when asked if you want to continue connecting, the ssh session will end as there’s no way to establish a secure connection without exchanging an encryption key.

The encryption key exchange will only happen the first time you open a connection with the server. When a key is accepted it’s stored on the client computer and reused anytime a new ssh connection is made to this server. However, if you open ssh connections to different servers you'll get a new, different encryption key for each server.

4. After the connection is opened the server will start a login process that will ask you for your password. The actual prompt will look similar to the following:

   ```
   milesDyson@skynet.mil's password:
   ```

   When you enter your password the characters will not appear as you type them. This is done as a security measure to prevent "shoulder surfing" which is when someone looks over your shoulder at your password as you enter it. Most password systems, like the one used by Microsoft Windows, hide the actual characters but display a substitute character like `*` instead, to provide some feedback and assurance that you're typing something. But the built-in version of ssh provided with Windows doesn't display anything, because even knowing how many characters you type can help an attacker, and really decrease the amount of time required to crack your password. So, don't be concerned when you type your password and nothing is displayed. Just hit the `<Enter>` key when you're finished and you'll be logged in, assuming you entered the password correctly.

5. If you login successfully the system may display a message reporting the last time you logged in or if you're lucky a cartoon cow saying something funny. The system administrator controls the information displayed when you initially login, so what you see will vary from system to system. The thing you really want to see is called the command prompt and it will be displayed after any login messages.

   ```
   Last login: Fri Aug 29 1997 02:14 AM
   %
   ```

   $\xleftarrow{\hspace{3em}}$ **The command prompt**

   Figure 2.2 The display after successfully logging in.

   The command prompt gets its name because it's the indicator that the system is waiting for you to enter commands. The command prompt is typically a `%` or `$` for normal user accounts, and a `#` for the administrator account which is commonly called root. However, these are just de facto standards and the command prompt can be easily customized, so it could be almost anything.

   At this point you can begin typing commands.

6. If there's a problem with user name or password you entered you'll see the message **Permission denied, please try again.** After a brief pause the **password:** prompt will be displayed again, allowing you to try again. The Windows 10 version of ssh allows 3 failed password attempts before it exits. If this happens you'll see a message similar to the following. It's not the best error message as it doesn't provide an explanation of the problem or how to correct it. But in this case it means you entered an incorrect password more than the allowed number of times.

```
Permission denied (publickey,gssapi-keyex,gssapi-with-mic,password)
```

7. To logout of the Linux/UNIX server use the `exit` command. When the server receives this command it will know that you are done, so it can log you out cleanly and close the network connection.

   It is possible to end your session by simply closing the Command Prompt window. The problem with this type of ungraceful exit is that it may keep using resources on the server. That is, the server may hold the network connection open and may still be waiting for you to type commands. You can think of this as being similar to ending a phone call. It’s better if you say goodbye or give some indication that you’re ending the call. If you just hang up mid conversation the other person may stay on line and maintain their end of the connection, and it may take the other person a while to figure out you’re gone.

8. Note that the ssh encryption keys will be stored in the file `C:\Users\<`*`username`*`>\.ssh\known_hosts` where *`username`* is your Windows user name. Each key will be stored as a single long line that contains the DNS name of the server, if it’s known, the IP address of the server, some information about the key, and the key itself. There’s typically no reason to edit or delete an encryption key, but every once in a while you’ll run into a situation where you need to delete the encryption key. For example, if the server is upgraded or changed it may not recognize the old key. If you ever need to delete an encryption open the known_hosts file and delete the appropriate line.

Running ssh in an Application - PuTTY

A second option for opening an ssh connection to a Linux/UNIX server is to use a windows based application. There are several available but one of the most popular is called PuTTY. (The tty in the name is the abbreviation used for the old terminal ports on a server that allowed a terminal to be directly connected to the server.) PuTTY is a free, open source program originally developed by Simon Tatham for Microsoft Windows computers.

Here’s the process for downloading and using PuTTY:

1. **Download PuTTY**. There are many sites that provide putty downloads, but many of them are sketchy from a security viewpoint. I suggest downloading from the original developer’s site at: https://www.chiark.greenend.org.uk/ or from putty.org. Although, if you go to putty.org the download link will take you back to www.chiark.greenend.org.uk.

   NOTE - You *may* need to add a rule to your firewall to allow **PuTTY** access to the network. (See the sidebar below).

2. **Install PuTTY**. No matter where you download it from, it’s strongly suggested that you practice safe computing and perform a security scan of the download before starting the

installation. Unless you've used PuTTY and ssh before, and have specific requirements, accept the default options and settings during the install.

3. **Start PuTTY**. If you didn't select to start PuTTY at the end of the install you can start it by clicking the PuTTY icon on the Windows desktop or by selecting Putty from the Windows Start menu. It will be in either the PuTTY(64-bit) or PuTTY(32-bit) folder, depending on the version you installed. When you install PuTTY you get a few different applications such as Pageant and PSFTP, so make sure and select PuTTY.

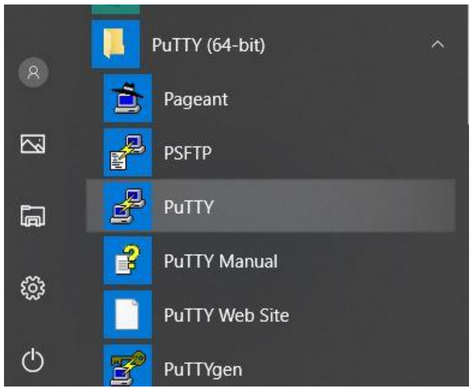


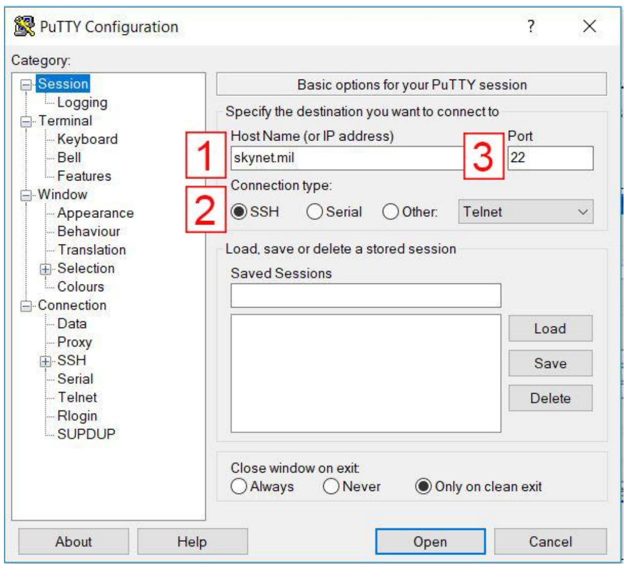
Figure 2.3 Starting PuTTY from the Windows Start menu.

4. **Configure the Connection**. The **PuTTY** Configuration window will appear. The first thing to note is that there are hundreds of settings that can be configured. If you click on the different menu items in the Category section on the left side of the PuTTY window you can view all of the possible settings.

   Luckily there are only a few items that need to be set configured to open an ssh session with a remote server, and they're all on the **Session** Category which is the one that's displayed by default when PuTTY starts. Here are the minimum settings that need to be configured:

   Make sure that **Session** is selected in the **Category** pane as shown in Figure 2.4. Note that the PuTTY interface may vary slightly from that shown as it has changed over the years.

   - In the **Host Name (or IP address)** box (1) enter the DNS name or IP address of the server. In the example the DNS name is skynet.mil.
   - In the **Connection type:** section (2) choose SSH. Note this section was labelled Protocol in older versions. This will automatically set the **Port** number (3) to 22.



Specify the destination you want to connect to

Host Name (or IP address): skynet.mil

Port: 22

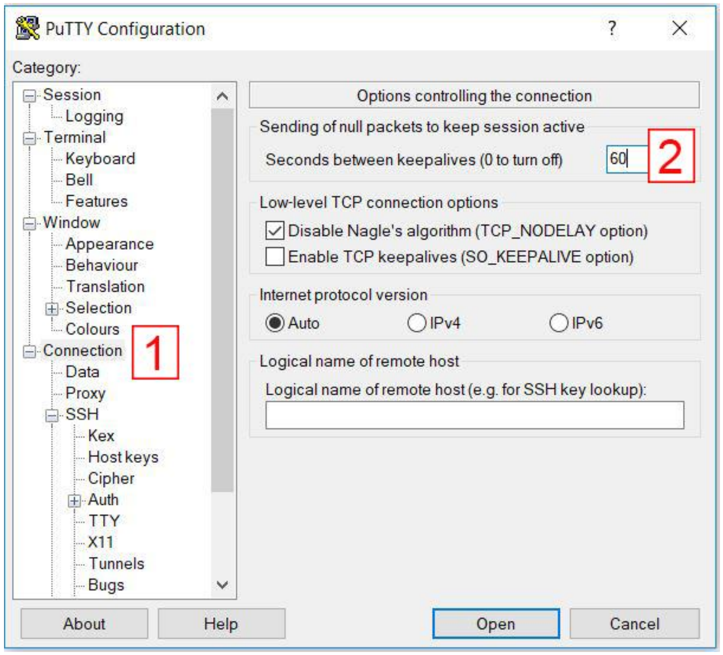
Connection type: [✓ SSH] Serial Other: Telnet

Close window on exit: Always Never [✓ Only on clean exit]

Figure 2.4 The main PuTTY Configuration screen.

- You will also want to check an option under the **Connection** section. This is an option that holds the network connection open even if there's no activity. If you don't set this PuTTY will drop the connection if you don't type anything for just a minute or two, which can be very annoying.

  To prevent timing out prematurely ensure that you're in the **Connection** Category (1), and then set the number in the **Seconds between keepalives** box (2) to anything besides 0.



Options controlling the connection

Sending of null packets to keep session active

Seconds between keepalives (0 to turn off) 60

Low-level TCP connection options

[✓] Disable Nagle's algorithm (TCP_NODELAY option)

[ ] Enable TCP keepalives (SO_KEEPALIVE option)

Internet protocol version

[● Auto] IPv4 IPv6

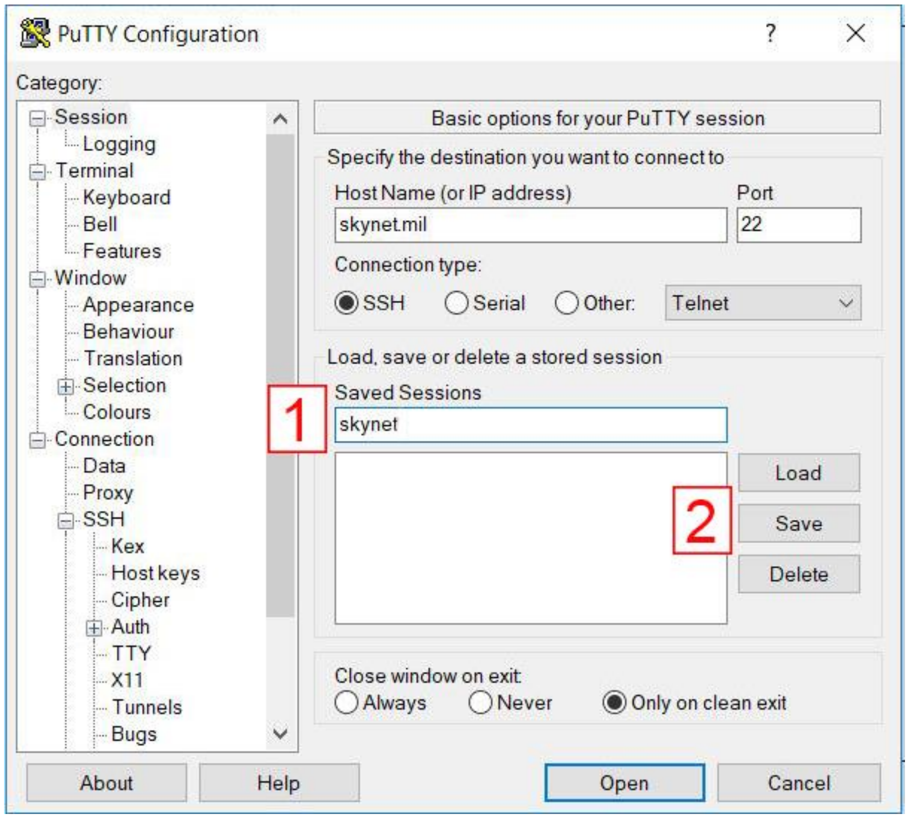
Logical name of remote host

Logical name of remote host (e.g. for SSH key lookup):

Figure 2.5 Configuring the keepalive setting to prevent PuTTY from dropping your connection.

- Once you have PuTTY configured you should save the settings as shown in Figure 2.6. This is done by returning the **Session** Category, supplying a name in the **Saved Sessions** (1) box, and clicking the **Save** button (2). In the example the configuration settings are saved under the name **skynet**. The next time you use PuTTY all your saved configurations will appear in the **Saved Settings** list. You can reload any of the saved configurations by clicking on the saved session name and selecting the **Load** button or by double-clicking the saved session name.

  You don’t actually have to save the settings. It’s just strongly suggested as it will make opening an ssh connection to a server much quicker if you need to login more than once.



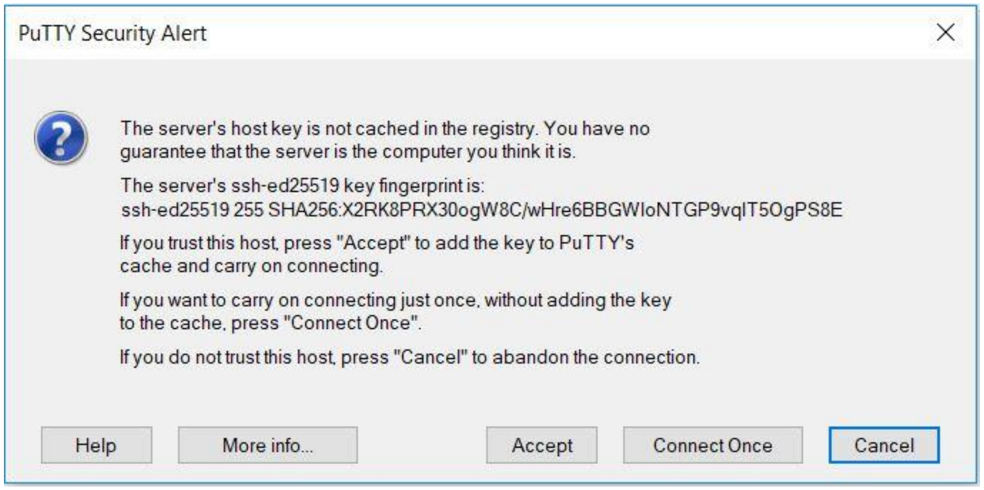
Host Name (or IP address): skynet.mil
Port: 22
Connection type: [✓ SSH] Serial Other: Telnet
Saved Sessions: skynet
Close window on exit: Always Never [✓ Only on clean exit]

Figure 2.6 Saving the PuTTY configuration.

5. **Connect to the Linux/UNIX Server**. Once PuTTY is configured hit the **Open** button at the bottom of the PuTTY window to open the connection to the server. This will open a new window and open an ssh network connection to the specified server.

   If this is the first time you've connected to this server, the server will send an encryption key that will be used to encrypt all the data from this point forward. You'll be shown some information about the encryption key and asked if you want to save it as shown in the following figure. Make sure and answer **Accept**. This saves the encryption key on your client computer. The key will be used to encrypt all data sent to the Linux/UNIX server from this point forward, protecting the data from any attacker that has access to a router between you and the server.

   If you click **Cancel** the PuTTY session will end as there's no way to establish a secure connection without exchanging an encryption key.


PuTTY Security Alert

The server's host key is not cached in the registry. You have no guarantee that the server is the computer you think it is.

The server's ssh-ed25519 key fingerprint is:
ssh-ed25519 255 SHA256:X2RK8PRX30ogW8C/wHre6BBGWIoNTGP9vqlT5OgPS8E

If you trust this host, press "Accept" to add the key to PuTTY's cache and carry on connecting.

If you want to carry on connecting just once, without adding the key to the cache, press "Connect Once".

If you do not trust this host, press "Cancel" to abandon the connection.

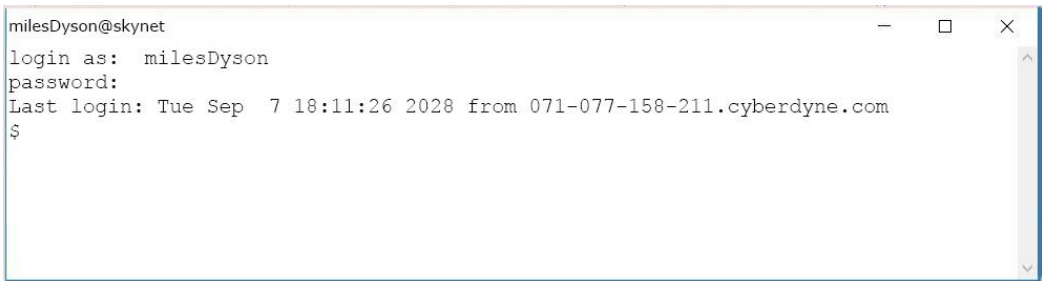
Figure 2.7 Message about storing and using the server's encryption key. You must click **Accept** or **Connect Once** to proceed.

The encryption key exchange will only happen the first time you open a connection with the server. When a key is accepted it's stored on the client computer and reused anytime a new ssh connection is made to this server. However, if you use PuTTY to open ssh connections to different servers you'll get a new, different encryption key for each server.

6. PuTTY will now be communicating with the login process on the Linux/UNIX server which will prompt you to enter your user name and password as shown in the following figure. Note that the messages and prompts displayed will vary between servers and Linux distributions. The messages and prompts will be similar to what's displayed in the figure, but may not be an exact match.

   When you enter your username and password there are two things to keep in mind. The first is that Linux and UNIX are case sensitive, which means it makes a distinction between upper-case and lower-case characters. This means if your username is **benDover** you must enter it with lower case **ben**, upper case **D** and lower case **over**. If you enter the characters in a different case, for example **BENdover**, the login process will see it as a different username and not recognize it as valid.

   The second is that your password may not appear when you enter it. This is done as a security measure to prevent "shoulder surfing" which is when someone looks over your shoulder at your password as you enter it. Most password systems, like the one used by Microsoft Windows, hide the actual characters but display a substitute character like `*` instead, to provide some feedback and assurance that you're typing something. But many Linux and UNIX systems don't display anything, because even knowing how many characters you type can help an attacker, and really decrease the amount of time required to crack your password. So, don't be concerned if you type your password and nothing is displayed. Just hit the `<Enter>` key when you're finished and you'll be logged in, assuming you entered the password correctly.



```
login as:  milesDyson
password:
Last login: Tue Sep  7 18:11:26 2028 from 071-077-158-211.cyberdyne.com
$
```

Figure 2.8 A typical login display. Note that for security purposes many systems don’t display anything as you enter your password.

7. If you login successfully the system may display a message reporting the last time you logged in or if you’re lucky a cartoon cow saying something funny. The system administrator controls the information displayed when you initially login, so what you see will vary from system to system. The thing you really want to see is called the command prompt and it will be displayed after any login messages.

```
Last login: Fri Aug 29 1997 02:14 AM
%
```

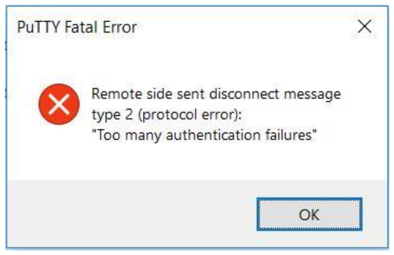
The command prompt

Figure 2.9 The display after successfully logging in.

   The command prompt gets its name because it’s the indicator that the system is waiting for you to enter commands. The command prompt is typically a `%` or `$` for normal user accounts, and a `#` for the administrator account which is commonly called root. However, these are just de facto standards and the command prompt can be easily customized, so it could be almost anything.

   At this point you can begin typing commands.

8. If there’s a problem with user name or password you entered you’ll see a message similar to **Access Denied** or **Permission denied, please try again.** After a brief pause the **password:** prompt will be displayed again, allowing you to try again. You’ll be allowed to try several passwords but each system is configured to drop the connection after some maximum number of failed attempts. If this happens you’ll see a message similar to the following.


PuTTY Fatal Error

Remote side sent disconnect message type 2 (protocol error): "Too many authentication failures"

Figure 2.9 The display after successfully logging in.

9. To logout of the Linux/UNIX server and close the network connection use the command: `exit`

   It is possible to end your session by simply closing the PuTTY window. The problem with this type of ungraceful exit is that it may keep using resources on the server. That is, the server may hold the network connection open and may still be waiting for you to type commands. You can think of this as being similar to ending a phone call. It's better if you say goodbye or give some indication that you're ending the call. If you just hang up mid conversation the other person may stay on line and maintain their end of the connection, and it may take the other person a while to figure out you're gone.

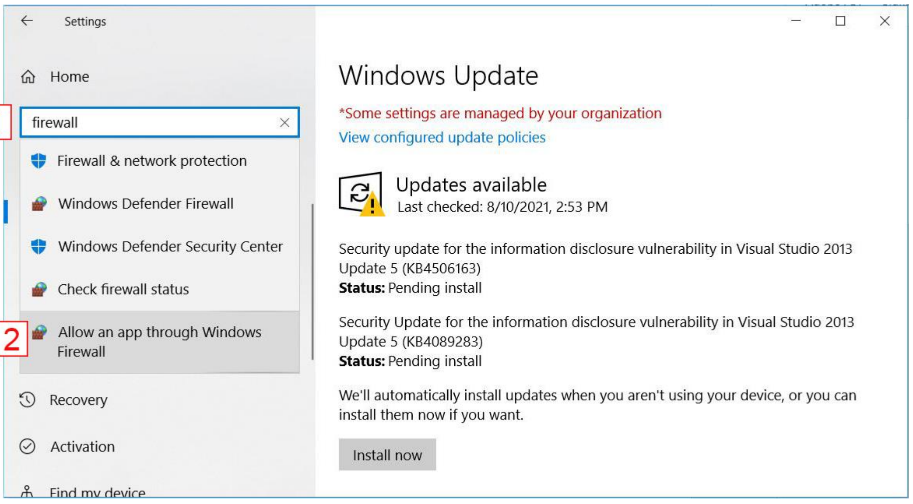
10. Note that the ssh encryption keys for PuTTY will be stored in Windows Registry in `HKEY_CURRENT_USER\Software\SimonTatham\PuTTY\SshHostKeys`. Each encryption key will be stored as a single registry key that contains the DNS name of the server, if it's known, or the IP address of the server if the DNS name isn't known. There's typically no reason to edit or delete an encryption key, but every once in a while you'll run into a situation where you need to delete the encryption key. For example, if the server is upgraded or changed it may not recognize the old key. If this happens there's no way to remove the key from PuTTY, you'll have to remove the key from Windows Registry.

Troubleshooting – Configuring the Windows Firewall to Allow PuTTY

If you can't connect to a remote Linux/UNIX server it may be because the firewall on your computer is set to block the ports used by ssh. In case you can't connect, here's a brief description of the process for setting a rule to allow PuTTY, or to be more technically correct to allow ssh to make a network connection and pass through the Windows 10 firewall. Note that if you need more detailed instructions for configuring the Windows firewall or help with another OS or another versions of Windows you can do an Internet search.

1. Ensure that you know where the PuTTY program has been installed. This is typically in the folder **C:\Program Files\PuTTY**.

2. Open Windows **Settings** and select **Update & Security**. Type **firewall** in the search box on the left (1). This will display a set of options below the search box. Select **Allow an app through Windows Firewall** (2) from the list of options.


Settings
Home
1
firewall
Firewall & network protection
Windows Defender Firewall
Windows Defender Security Center
Check firewall status
2
Allow an app through Windows Firewall
Recovery
Activation
Find my device
Install now

Windows Update

*Some settings are managed by your organization

View configured update policies

Updates available

Last checked: 8/10/2021, 2:53 PM

Security update for the information disclosure vulnerability in Visual Studio 2013 Update 5 (KB4506163)

Status: Pending install

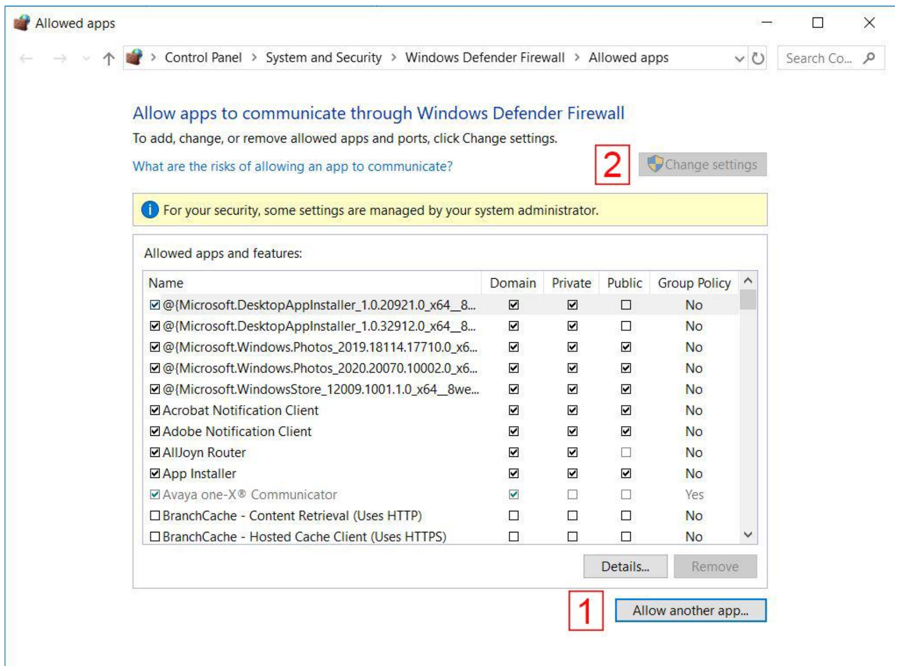
Security Update for the information disclosure vulnerability in Visual Studio 2013 Update 5 (KB4089283)

Status: Pending install

We'll automatically install updates when you aren't using your device, or you can install them now if you want.

Figure 2.10 The Windows 10 Update & Security window.

3. This will open the Firewall Allowed Apps window. Click on the **Allow another app** button (1) at the lower right of the window. If this button is grayed out and disabled you'll need to click the **Change settings** button (2).


Allowed apps
Control Panel > System and Security > Windows Defender Firewall > Allowed apps
Search Co...
2
Change settings
Details...
Remove
1
Allow another app...

Allow apps to communicate through Windows Defender Firewall

To add, change, or remove allowed apps and ports, click Change settings.

What are the risks of allowing an app to communicate?

For your security, some settings are managed by your system administrator.

Allowed apps and features:

Name	Domain	Private	Public	Group Policy
☑ @{Microsoft.DesktopAppInstaller_1.0.20921.0_x64_8...	☑	☑	☐	No
☑ @{Microsoft.DesktopAppInstaller_1.0.32912.0_x64_8...	☑	☑	☐	No
☑ @{Microsoft.Windows.Photos_2019.18114.17710.0_x6...	☑	☑	☑	No
☑ @{Microsoft.Windows.Photos_2020.20070.10002.0_x6...	☑	☑	☑	No
☑ @{Microsoft.WindowsStore_12009.1001.1.0_x64_8we...	☑	☑	☑	No
☑ Acrobat Notification Client	☑	☑	☑	No
☑ Adobe Notification Client	☑	☑	☑	No
☑ AllJoyn Router	☑	☑	☐	No
☑ App Installer	☑	☑	☑	No
☑ Avaya one-X® Communicator	☑	☐	☐	Yes
☐ BranchCache - Content Retrieval (Uses HTTP)	☐	☐	☐	No
☐ BranchCache - Hosted Cache Client (Uses HTTPS)	☐	☐	☐	No

Figure 2.11 The **Windows Firewall Allowed apps** window.

4. This will open the Add an app window. If you know the path to the PuTTY executable you can type it in the **Path:** box, or you can click the **Browse** button to find the PuTTY executable file.

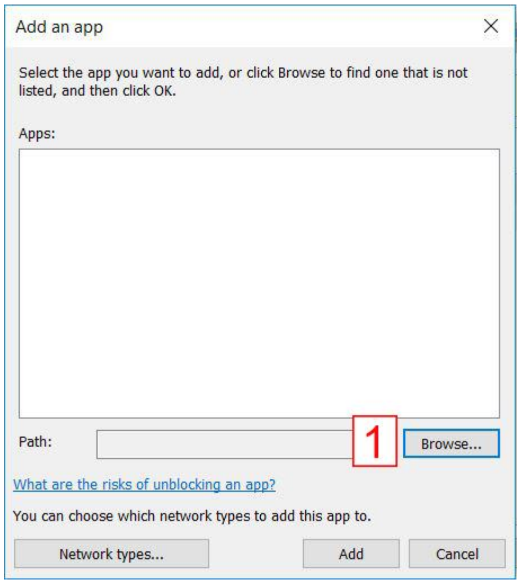
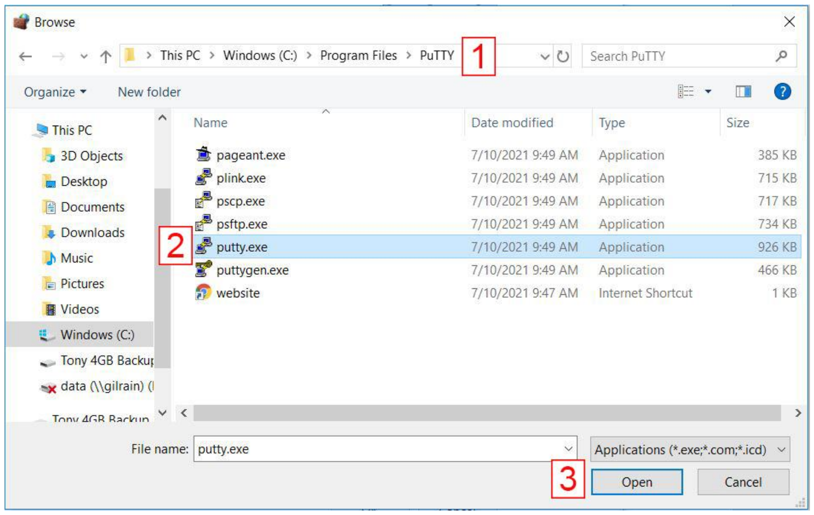

Add an app
Select the app you want to add, or click Browse to find one that is not listed, and then click OK.
Apps:
Path:
1
Browse...
What are the risks of unblocking an app?
You can choose which network types to add this app to.
Network types...
Add
Cancel

Figure 2.12 The **Add an app** window.

5. This will open the Browse window and allow you to navigate to the PuTTY folder, which is typically **C:\Program Files\PuTTY** (1). Open the folder, then select the **putty.exe** file (2). You can either double-click the putty.exe file, or select the file and click the **Open** button (3).

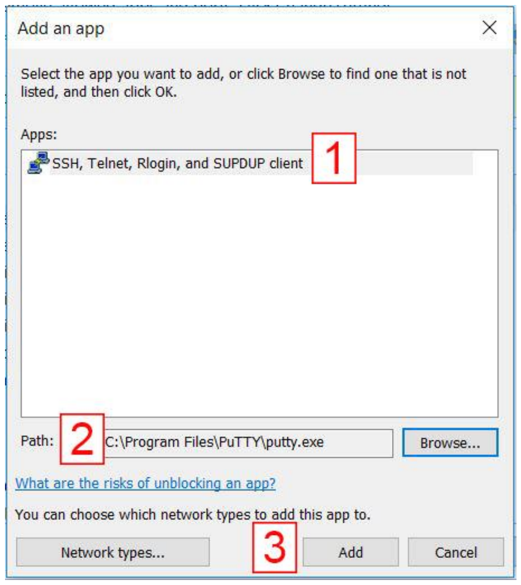


Name	Date modified	Type	Size
pageant.exe	7/10/2021 9:49 AM	Application	385 KB
plink.exe	7/10/2021 9:49 AM	Application	715 KB
pscp.exe	7/10/2021 9:49 AM	Application	717 KB
psftp.exe	7/10/2021 9:49 AM	Application	734 KB
putty.exe	7/10/2021 9:49 AM	Application	926 KB
puttygen.exe	7/10/2021 9:49 AM	Application	466 KB
website	7/10/2021 9:47 AM	Internet Shortcut	1 KB

File name: putty.exe
Applications (*.exe;*.com;*.icd)

Figure 2.13 Selecting the putty executable in the **Browse** window.

6. This will return you to the Add an app window. You should now see **SSH, Telnet, Rlogin and SUPDUP client** (1) in the list of **Apps:** and the path to the putty.exe file in the **Path:** box. If this is correct click the **Add** button, and your PuTTY/ssh connections should now be allowed through the Windows Firewall.



Select the app you want to add, or click Browse to find one that is not listed, and then click OK.

SSH, Telnet, Rlogin, and SUPDUP client

Path: C:\Program Files\PuTTY\putty.exe

You can choose which network types to add this app to.

Figure 2.14 The Add an app window after PuTTY has been selected.

Troubleshooting – Configuring the Windows Firewall to Allow PuTTY

As you start to use ssh and PuTTY you may notice that some characters sent by the Linux system look strange when they're displayed. All the normal text characters will display without problem, but some characters such as − or dash character may be displayed as â. The root cause of this is that the problematic characters are in the range 128-255 in the ASCII character set, and for many years these ASCII numbers were left undefined. Appendix A contains a copy of the ASCII table showing characters in the range 0-127. This meant that interpreting characters in the range 128-255 was left up to each device. There are several ways to fix this problem if it occurs[1], but one of the simplest is to type the following command after you've logged in:

```
setenv LANG “en_US”
```

Note that this is case sensitive and must be typed exactly as shown to have an effect. You will have to do this every time you log in, until you learn how to add it to one of your start-up files later in a later chapter.

[1] https://fixyacloud.wordpress.com/2020/01/26/how-to-fix-putty-showing-garbled-characters/

Changing Your Password

Once you have successfully logged in to a system for the first time, the next thing you should do is change your password. This is done using the `passwd` command. When you run the passwd command you'll be asked to enter your old password, to ensure you're the account owner, then prompted twice for the new password to ensure you typed it in correctly.

When the `passwd` utility runs the dialog will look something like the following. The items you need to type are displayed in bold text.

```
% passwd
Changing password for username.
Enter old password: yourOldPassword
Enter new password: yourNewPassword
Re-type new password: yourNewPassword
passwd: all authentication tokens updated successfully.
```

The system does some basic checking of the new password before accepting it, and will prompt you if your password doesn't meet the password rules for the system. For example, there's probably a minimum length rule, but there also may be rules that require both upper and lower case characters, etc.

The Linux/UNIX Filesystem

Now that you're able to login, the next thing to do is to start looking at and getting familiar with the main folders and files on a Linux system. One of the facts of life regarding modern computer systems like Windows or Linux and UNIX, is that they have hundreds of thousands of files. You don't need to know about all the files on a system as many of the files used by the Operating System files aren't of particular interest to a casual user. But there are a few system folders you should aware of, and you should know how to see the different drives connected to the system, plus you'll need to know how to view your own files and folders. So in this section you'll learn the commands for moving around the file system and viewing the files stored in a directory or folder, and you'll use this knowledge to look at the some of the main folders in the Linux/UNIX file system structure.

Basic Command Structure

In this chapter you will learn how to use several commands. To execute a command or run a program you simply type the command name. In general, the things you type on the command line will have this structure:

```
command  -options  arguments
```

You will always type the main command, but some commands may also have *options* and *arguments*. Options are typically designated by a dash "-", and they are used modify the things the command does.

Some commands also require extra information, called arguments to operate. The term argument doesn't refer to the disagreement type of argument; it's a programming term for things that you supply to the command. The arguments are typically things you want the command to work on. For example, a file that you want to delete would be an argument to the command for deleting files.

Exploring the Linux/UNIX file system, `ls`, `cd` and `pwd`

Ok, let's get started looking at the files and folders in the Linux/UNIX filesystem. One of the first things to do is become familiar with the layout of the files and directories in the Linux/UNIX file system. This is a lot like moving to a new city and trying to get oriented. You know the city probably has a grocery store, coffee shop, gas station, and hopefully a hospital or medical center. When you move somewhere new you need to figure out where the stores and services you want to use or need to use are located. Similarly, when you start learning about a new computer system you know the system files will be stored somewhere, and programs will be stored somewhere, and user files like documents, or music, or photos will also be stored somewhere. When you first start you won't know exactly where things are located, but in the city or on the computer you can start looking around until you get oriented.

With Windows Explorer, and any system with a GUI, you can move through the file system by clicking on different folders, and it's easy to tell where you are in the file system because Windows Explorer displays it graphically. But with DOS and the Linux/UNIX command line you have to use commands to move from directory to directory, and different commands to display the files and folders in a directory, and to figure out where you're currently located in the filesystem. In DOS the `CD` command is used to move around the file system and the `DIR` command is used to list or view the contents of the various directories. In Linux/UNIX, you use the `cd` command to move between directories or folders, the `ls` command to see the files inside a directory, and the `pwd` command to display your current location.

The `ls` command - listing the contents of a directory

The `ls` command is similar to `DIR` in DOS as it displays a list of most of the files in the directory. To run this command and see the contents of the directory you're currently located in type:

```
ls
```

When you first login to a Linux/UNIX system you're placed in your home directory. And since new users don't have many files in their home directory you may not see much if you're in your home directory, maybe nothing at all. It is possible that your system administrator created some files for you, and if so, their names will be displayed, but it's very likely you won't see anything displayed.

By default, the `ls` command displays a list of the files in the directory you're currently in. This is like Windows Explorer, which shows you the files in the currently open folder. But the `ls` command can also be used to display the files in a different directory, by supplying the path to the directory as an argument. For example:

```
ls /etc
```

This will display the files in the /etc directory. This is useful when you want to see the files in another directory, but you don't necessarily want to move there first. (You'll learn about the typical Linux directories, changing directories, and how to specify the path to a directory or file later.)

If `ls` doesn't list any files it doesn't necessarily mean there aren't any files in the directory. It's possible that the directory has some files that are called hidden files, especially in your home directory. On Linux/UNIX systems, any file that has a name that starts with a period or dot is considered hidden and will not be displayed by running `ls`. If you want `ls` to display all of files in a directory, including the hidden files, you need to use the `-a` (for all) option with the `ls` command by typing:

```
ls -a
```

Note – the hidden files aren't hidden for security reasons. As you can see (no pun intended) they're easy to display using `ls -a`. The hiding is done more for housekeeping purposes. That is, the hidden files are usually configuration files which are used to control your personalization settings for various programs. Hiding them from the normal `ls` display keeps them from cluttering up the display amongst your actual data files and programs. In addition, hiding them provides a little protection when you do something like run the command to delete or move all the files in your home directory. It's not great protection as you can easily override it, but it does provide a little protection as you do have to actively override the protection. This just makes it harder to accidentally delete or move the files. Linux/UNIX has other ways to protect files besides just hiding them, unlike DOS/Windows where the Hidden Attribute was often set to try and protect files.

To see more detailed information about each file, use the `-l` (for long listing) option for the ls command. That is, type:

```
ls -al
```

Note that the option order doesn't matter, that is `ls -al` and `ls -la` are equivalent. You'll learn exactly what this extra information displayed by the `-l` option represents later, but you could take a minute and try and decipher what this extra information might mean on your own. Compare this to what you see when you set Windows Explorer to show you the detailed listing for files.

The `cd` command (change directory)

The `cd` command is used to change the current directory and works much like the DOS/Windows `CD` command. For example, to move to the **/usr** directory you would type:

```
cd /usr
```

Typically, you provide the `cd` command with the name or path of the directory/folder you wish to move to. That is, you wouldn't just type `cd` by itself. But in some Linux distributions typing `cd` with no arguments will move you to your home directory. (You'll learn more about directory paths and your home directory below.)

The **`pwd`** command - print working directory

When a user is logged onto a Linux/UNIX system, they are always somewhere in the file system. With Windows, or if you're using Linux with a GUI, you can open a program like File Explorer and simply look to see where you're at in the file system. But if you're using the command line and typing commands you have to keep track of your location on your own. If you can't remember where you are you can use the `pwd` command as it displays the pathname of the current directory.

Absolute and Relative Paths

When you start working with the `cd` and `ls` commands, you'll be asking them to move you to a different folder, or possibly list the files in a different folder. Telling these commands about different directories or different folders is done using something called a path. There are two types of paths, absolute and relative, and when it comes to changing directories it's crucial to know the difference between absolute and relative paths, and when and how to use them.

The first concept to understand about both absolute and relative paths is that they're ways of providing directions to a folder or a file. An easy way to understand the difference between an absolute path and a relative path is to compare them to directions you'd give with a map.

Look at the following map, which we'll use to demonstrate absolute and relative paths.

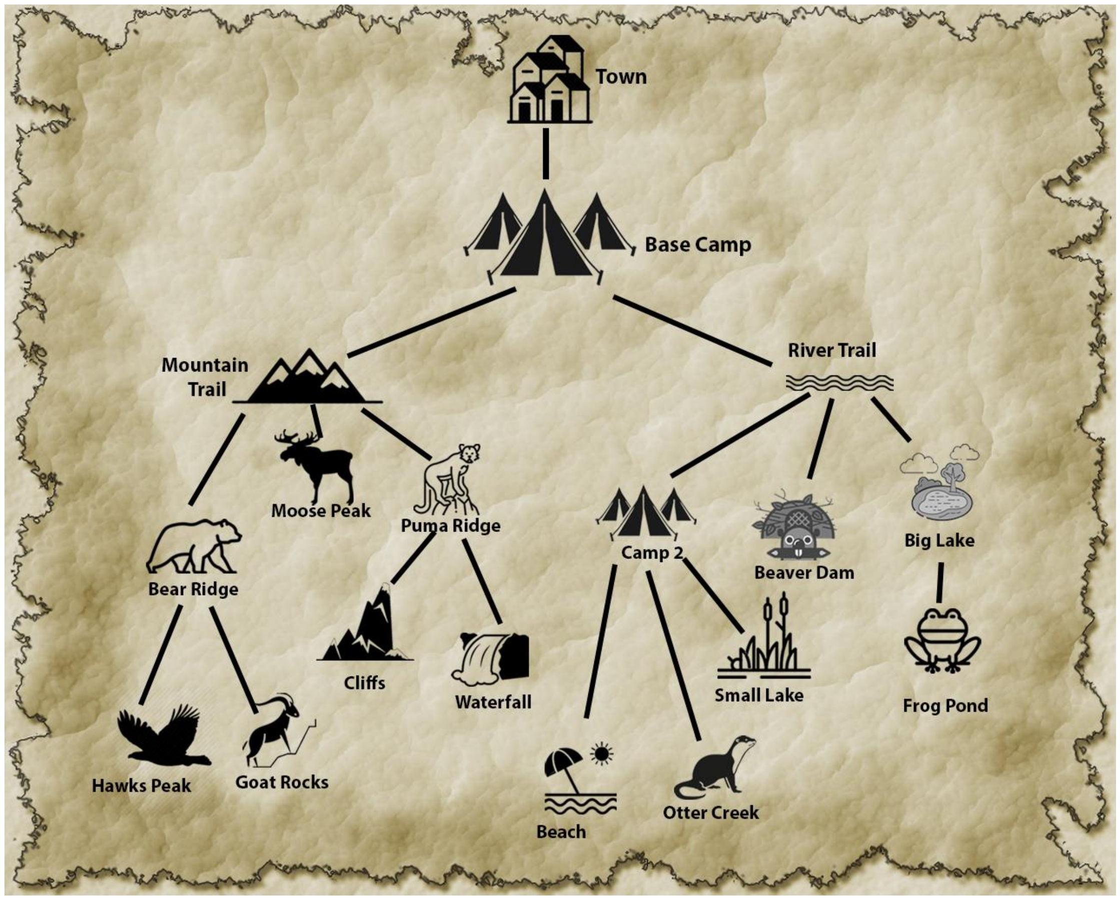


Figure 2.15 Map of the Linux Mountain National Park, which will be used to demonstrate absolute and relative paths.

The main things to note are:

1. The Town is at the top of the map. This is equivalent to the top directory, or as we call it in the Linux the root directory.

2. As you traverse the map leaving from Town, there's only one set of trails that will get you to any destination. This is much different than real life, where there are typically many different inter-connected roads and trails, and multiple routes you can take to get from one place to another. But having only one way to get to any destination is exactly like a computer's filesystem, where there's only one way to get down into a folder or directory.

3. Conversely, if you're at any location on the map, there's only one trail you can take that will take you back towards Town. There might be multiple trails out of any location, but only one will lead back towards Town. This is like the computer filesystem, where each folder can have multiple sub-folders, but each folder can only have one parent folder.

That is, there’s no way for a folder to be inside two different folders at the same time, it can only be inside a single folder.

Now say that you’re employed as the Linux Park Ranger in charge of the area covered by the map. You spend most of your time outdoors, travelling from place to place on the map checking conditions of the trails and facilities, and helping hikers and campers. When you meet people one of the main things they ask you for is directions.

The simplest directions you can provide will show how to move between adjacent locations. For example, say someone asks you how to get from Base Camp to the Mountain Trail, or from Bear Ridge to Hawks Peak, or from the Beaver Dam back to the River Trail. You may also be asked to provide more complicated directions, like how to get from Base Camp to the Beach, or from Hawks Peak to the Frog Pond. We’ll first show you how to provide directions between adjacent locations, and then move on to more complicated directions.

Moving Between Adjacent Locations
To provide directions for moving between adjacent locations you’re either going to tell them to go back towards Town, or which trail to take from the current location. In either case it’s just one hop on the map from the current location.

If the people you’re helping want to go somewhere that’s in the direction away from Town you simply tell them the name of the trail to take. For example, if you’re currently on the Mountain Trail and they want to get to Moose Peak you just tell them to go to Moose Peak. Or if you’re currently on the River Trail and they want to go to the Beaver Dam you just tell them to go to the Beaver Dam. It’s that simple.

If they’re going back towards Town it’s even simpler. You don’t need to specify which trail they should take, because there’s only one way back, you just say go back towards Town. And to make it even easier you’ve developed a shortcut or code for telling people to take the trail back to Town, which is to say or write .. or two dots or periods in a row. For example, if you’re currently at Hawks Peaks and want to get back to Bear Ridge the directions would be .. Or if you’re Bear Ridge and want to get back to the Mountain Trail the directions would also be just ..

More Complicated Directions, Making Multiple Hops
The next situation to discuss is when someone asks you to provide more complicated directions. That is, they want to get somewhere that isn’t adjacent to your current location. Or in technical terms the place they want to go is more than one hop away. When you provide directions for destinations that require multiple hops you have to choose between two methods of providing the directions, or maybe it would be more accurate to say you have to choose between two different starting points. You can always start the directions at Town, or you can start the directions from your current location. Everybody knows where Town is since they had to pass through the Town to get to any location on the map. So, you could always start your directions at Town and be assured that the people you’re helping will be able to get to their desired destination. For example, if someone wanted to get to the Big Lake you could provide these directions:

1. Start at Town and take the road to Base Camp
2. Take the River Trail

3. When you get to the end of the River Trail, take the Big Lake trail

You other choice is to start the directions from your current location. For example, say you and the people you're helping are at the Beaver Dam and they want to know how to get to the Beach. In this case you could provide these directions:

1. Start from here, and head back up the trail until it intersects the River Trail
2. At that point, take the trail to Big Lake

If you start the directions at Town, this would be like an absolute path. The thing that makes this an absolute path is that it always starts at the top of the map at the Town. With absolute paths it doesn't matter where you are currently located, you must go back to a fixed starting point, the Town, to start.

Your second option for providing directions would be to use relative paths, where you start them relative to your current location.

While you can use either absolute or relative paths to reach any location on the map, you also want to make your directions as easy to follow as possible. To decide which one is easier, you need to look at both your current location and the destination. In some cases, the choice will be obvious, but in other cases both sets of directions might be equally easy to follow.

Here are some examples of obvious choices:

A. Say you're at the Beach and someone wants directions to Otter Creek. In this case the relative directions will be much simpler to explain and quicker to follow.

Absolute Directions	**Relative Directions**
From Town, take the road to Base Camp	Head back up the trail until you reach Camp 2
Take the River Trail	Take the trail to Otter Creek
Take the trail to Camp 2	
Take the trail to Otter Creek	

B. Say you're at Goat Rocks and someone wants directions to Big Lake. In this case the absolute directions will be simpler to explain.

Absolute Directions	Relative Directions
From Town, take the road to Base Camp	Head back up the trail until you reach Bear Ridge Trail
Take the River Trail	Head back up Bear Ridge Trail until you reach the Mountain Trail
Take the trail to Big Lake	Head back up the Mountain Trail until you reach the Base Camp
	Take the River Trail out of Base Camp
	Take the trail to Big Lake

And here's an example where the choice isn't obvious because they're roughly equivalent:

C. Say you're at Bear Ridge and someone wants directions to Big Lake.

Absolute Directions	Relative Directions
From Town, take the road to Base Camp	Head back up the trail until you reach the Mountain Trail
Take the River Trail	Hike back up the trail until you reach Base Camp
Take the trail to Big Lake	Take the River Trail
	Take the trail to Big Lake

If we actually had to hike to get from one place to another then in this last example using the relative path would be easier. No one wants to walk to Town just to turn around and walk back again. But luckily (or sadly if you like hiking) moving around the computer file system doesn't involve any physical effort. The main point of the example is to demonstrate that sometimes there's not an obvious better choice between relative and absolute paths.

To make our directions a little more like the paths in a computer's filesystem let's introduce one new character or special symbol that we can use to make the directions more succinct and also take a closer look at using .. to move one step back towards Town.

The / character is used for three purposes:

1. If it's used by itself, it's a code for saying go to the Town. So, if you tell someone to go to / this means they should go back to the Town.

2. If the / is placed at the *start* of a set of directions it means the directions begin on the trails coming from Town and not from our current location. Since the Town is a fixed location this means we're using absolute directions. For example, **/Base Camp** means to start at the Town, and from there go to Base Camp. Your current location doesn't matter, if you see the / at the start of the directions the first thing you should do is go back to Town and start there.

3. If **/** is placed between two areas, it means to travel from the first area into the second area. For example, **River Trail/Camp 2/Beach** means to travel to the River Trail, then to Camp 2, and from there to the Beach.

Now let's take a closer look at **..** which represents the single trail out of any area. As explained above, .. is short hand for an instruction to head one hop back towards the top of the map or back towards Town. For example, if you were at Hawk Peak **..** would refer to the trail back to Bear Ridge. You don't have to specify Bear Ridge, because the only trail back towards Town from Hawks Peak is the one to Bear Ridge.

If you want to take multiple trails back towards Town you always represent each trail going towards Town with .. For example, the directions from Small Lake to the River Trail would be ../.. The first .. represents the first hop back towards Town which is the trail from Small Lake to Camp 2, the / is the delimiter between the trails, and the second .. means to take another hop back towards Town from Camp 2, which is the trail to the River Trail.

It's important to note that if our directions start with / they're absolute directions, but if they start with any other character, they're going to be relative directions. You should also note that when you're using a relative path you do NOT specify the current location. In a relative path the first thing you specify will either be .. if you want someone to head back towards Town, or the name of the place to go if the first step is to move down a trail away from Town.

Here are some example directions that make use of the / and .. characters:

A. The absolute path to get to Goat Rocks would be: **/Base Camp/Mountain Trail/Bear Ridge/Goat Rocks**. Since this starts with / it means to start to start at the Town.

B. The absolute path to get to the Beaver Dam would be: **/Base Camp/River Trail/Beaver Dam**. Since this starts with / it means to start to start at the Town.

C. The relative path to get from Puma Ridge to the Cliffs would simply be: **Cliffs**. Since this does NOT start with / it tells a person to travel from the current location. Note that if you gave someone the instructions /Cliffs it would mess them up. The / would tell them to go to Town, and once there they wouldn't be able to find the trail to the Cliffs. The trail name **Cliffs** only makes sense if you're currently located on Puma Ridge.

D. The relative path to get from the Mountain Trail to Goat Rocks would be: **Bear Ridge/Goat Rocks**. Since this does NOT start with / it tells a person to travel from the current location. Once again if you gave someone the instructions **/Bear Ridge/Goat Rocks** it would mess them up. The / would tell them to go to the Town, and once there they wouldn't be able to find the trail to Bear Ridge. The trail name **Bear Ridge/Goat Rocks** only makes sense if you're currently located on the Mountain Trail.

E. The relative path to get from the Base Camp to Small Lake would be: **River Trail/Camp 2/Small Lake**. Since this does NOT start with / it tells a person to travel from the current location.

F. The relative path to get from Otter Creek to the Beaver Dam would be: **../../Beaver Dam**. Dissecting this results in the following:

   - Since it doesn't start with a / it's a relative path, which means we start at the current location which is Otter Creek
   - The first **..** means to take the trail from Otter Creek up to Camp 2.
   - The **/** is a delimiter between instructions
   - The second **..** means to take the trail from Camp2 up to the River Trail
   - The **/** is a delimiter between instructions
   - **Beaver Dam** means to take the Beaver Dam trail

G. The relative path to get from Otter Creek to Goat Rocks would be: **../../../Mountain Trail/Bear Ridge/Beaver Dam**. Dissecting this results in the following:

   - Since it doesn't start with a / it's a relative path, which means we start at the current location which is Creek
   - The first **..** means to take the trail from Otter Creek up to Camp 2.
   - The **/** is a delimiter between instructions
   - The second **..** means to take the trail from Camp2 up to the River Trail
   - The **/** is a delimiter between instructions
   - The third **..** means to take the trail from Camp2 back up to the Base Camp
   - The **/** is a delimiter between instructions
   - **Mountain Trail/Bear Ridge/Goat Rocks** means to take the Mountain Trail followed by the Bear Ridge trial down to Goat Rocks.

   Notice that in this case it might be simpler to use an absolute path. That is **/Base Camp/Mountain Trail/Bear Ridge/Beaver Dam** seems much simpler to me than **../../../Mountain Trail/Bear Ridge/Beaver Dam**.

Using paths with Linux Commands

Now let's switch from our map analogy and look at how paths are used with Linux commands. To do this we'll use a filesystem layout that uses a set of folders with the same names as the areas on the map, organized in a similar fashion, as shown in the following two images. Both images display the same folder/directory configuration, they're just in different layouts to help you visualize how the folders are organized. Note that this is just a fictitious layout, a real Linux system has different directories. But we'll stick with this layout for this initial demonstration.

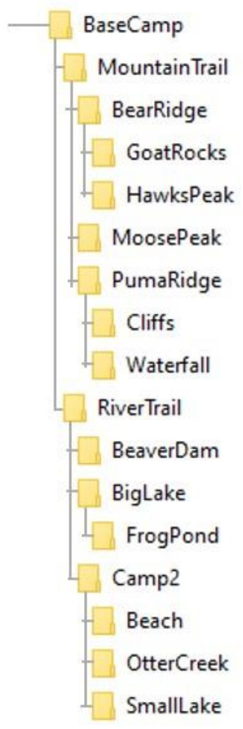


Figure 2.16 Viewing the Linux National Park Map as a set of folders or directories.

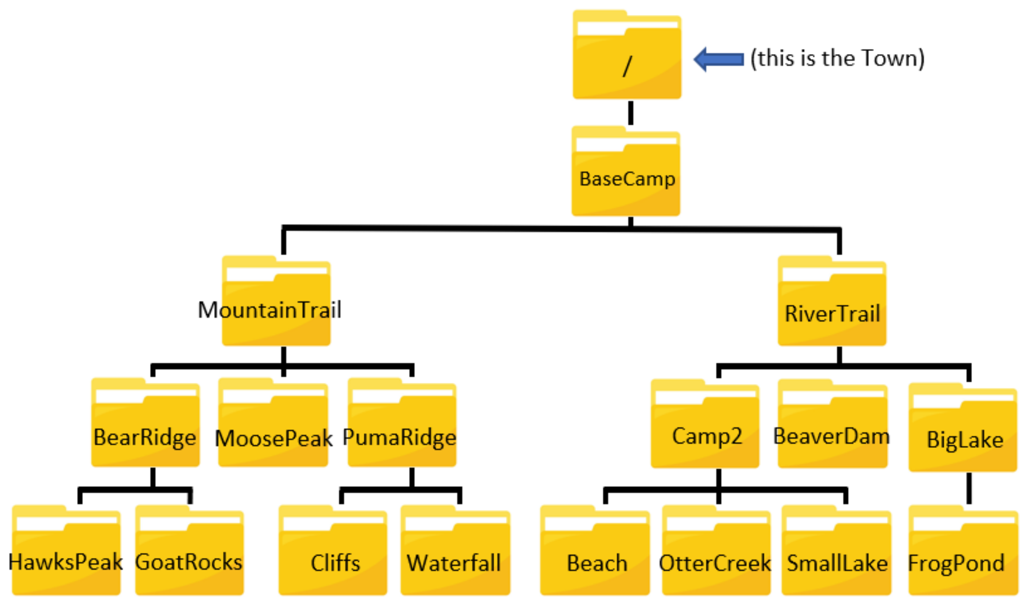


Figure 2.17 An alternate view of the Linux National Park Map as a set of folders or directories.

In a computer's filesystem the absolute and relative paths work just like they do with the map directions. The main differences from our map are:

1. On our map the top area has a name, Town, while in the filesystem it will just be referred to as /

2. The spaces in the area names have been removed. That is Otter Creek on the map is called OtterCreek in the filesystem. It is possible to have spaces in the folder names in the Linux filesystem, but file and folder names with spaces are harder to work with. So, I've removed the spaces from the names for this demonstration.

The commands we'll use for the demonstration are `cd`, for changing directories, and `ls` for displaying the files and folders in a directory.

Making one hop up or down
The first thing to demonstrate is using the `cd` command to move down one directory or move up one directory. To move down one directory, you simply give `cd` the name of the folder you want to move into. For example, say you're in the BaseCamp folder and you want to move down into the MountainTrail folder. To accomplish this you would type:

```
cd MountainTrail
```

To move up one directory you use the .. symbol. This is also referred to as the parent folder or directory. For example, if you are currently in the HawksPeak folder **..** would refer to the BearRidge folder and to move to BearRidge from HawksPeak you would use:

```
cd ..
```

In this example, or any time you move up one folder, you don't specify the name of the parent folder. Or this case you don't specific the name BearRidge, because the HawksPeak folder only has one parent directory, which is the BearRidge directory.

Note that if you're in the HawksPeak folder and use the command `cd BearRidge` it will result in an error because the `cd` command will be in the HawksPeak directory looking for a sub-directory named BearRidge, and won't be able to find BearRidge. Or if you use `cd ../BearRidge` it will also be an error because the `cd` command will start in the HawksPeak directory, follow the instruction .. and move up one directory into the BearRidge folder, and then look for a sub-directory named BearRidge; which will also be an error. When you want to move up one folder, or to move to the parent directory just type `cd ..`

If you want to use the `ls` command to display the content of a folder that's directly below the current folder you would type `ls` followed by the folder name. For example, assume you're in the BearRidge directory and you want to see a list of the files in the GoatRocks folder. To accomplish this you would type:

```
ls GoatRocks
```

If you're in the GoatRocks folder and you want to see what's in the BearRidge folder you would use:

```
ls ..
```

Once again you only use .. You don't use the name BearRidge or ../BearRidge because this would cause an error.

Making multiple hops
To move more than one directory in either direction you'll need to specify an absolute or relative path as the destination. When we use absolute and relative paths to describe locations on a computers filesystem we'll still use the `/` and `..` characters like we did with the map. But in this case, since we're talking about the filesystem and not a camping map, the `/` and `..` characters will take on these meanings:

1. **/** is used for three purposes:

    a. If it's used by itself, without any other folder name, it's a code for saying go to the top-level folder, which is also called the *root directory*. So, if you tell someone to go to / this means they should go back to the root directory. (The root directory gets its name from the fact that if the entire directory structure is flipped over, it can be viewed as a tree with branches made from the paths to the various folders. In this case the / directory would be like the tree's root, hence the name.)

    b. If the / is placed at the *start* of a path it means the path begins at the root directory and not from our current location. Since the root directory is a fixed location this means we're using absolute directions. For example, **/BaseCamp** means to start at the root directory, and from there go into the Base Camp folder. Your current location doesn't matter, if you see the / at the start of the path the first thing you should do is go back to the root directory.

    c. If **/** is placed between two folder names, it means to travel from the first folder or directory down into the second folder. For example, **RiverTrail/Camp2/Beach** means to move from the current directory into the RiverTrail folder, then to the Camp2 folder, and from there to the Beach folder.

2. **..** which represents the single path out of any folder which heads back towards the top directory.

3. If you want to take multiple hops back towards the root directory you always represent each parent directory with .. delimited by a /

    For example, to move from the SmallLake folder to the RiverTrail folder the command would be `cd ../..` The first .. represents the move from the SmallLake directory to its parent folder which is Camp2. The / is the delimiter between the folders, and the second .. means to move to the parent directory of the Camp2 folder, which is the RiverTrail folder.

Your Home Directory and the ~ (tilde) shortcut

In a minute we'll go through several examples of using the `cd` and `ls` commands in combination with relative and absolute paths. But before we do that there's one other important piece of information you need to be armed with. This regarding your *home directory*, which is the folder the OS places you in when you first login. This is similar to the Documents folder on a Windows system, as it's generally thought of as a place for a user to store their files. But it's important to note that on a Linux system this will be the folder that the system places you in when you first login.

The exact location of your home directory is set by the system administrator when your account is created. It's common practice on systems with multiple users to give each user their own home directory under the `/home` directory. It's also common practice to give the user's home directory the same name as the user. For example, the user **stesha** would be assigned the home directory `/home/stesha`, and each time Stesha logged in she would initially be in the `/home/stesha` directory. While using /home/username for home directories is common practice, it's not a requirement and the system administrator can choose to make the home directories anywhere in the filesystem, or they may even choose to not have home directories.

When you're changing directories or specifying directory paths the ~ or tilde character is used as a shortcut for your home directory. For example, if your home directory is `/home/jose` the path `~/junk` is equivalent to `/home/jose/junk`. Or, if you want a quick way to return to your home directory you can simply type `cd ~`

On some Linux distributions you can also move to your home directory by typing `cd` with no arguments. That is, simply type: `cd` But note this doesn't happen with every distribution, so I don't count on it and typically type `cd ~` since it only costs me the time of typing one additional character.

Examples of using absolute and relative paths with the `ls` and `cd` commands

Here are some examples of using absolute and relative paths with the cd command to change directories, and with the `ls` command to display the contents of different folders or directories. All the examples the following filesystem structure shown in the following diagram. Note that this is just a portion of the filesystem for a fictional Linux system. Most of the folders you'd find on an actual Linux system have been omitted to keep things simple enough to be usable in these examples.

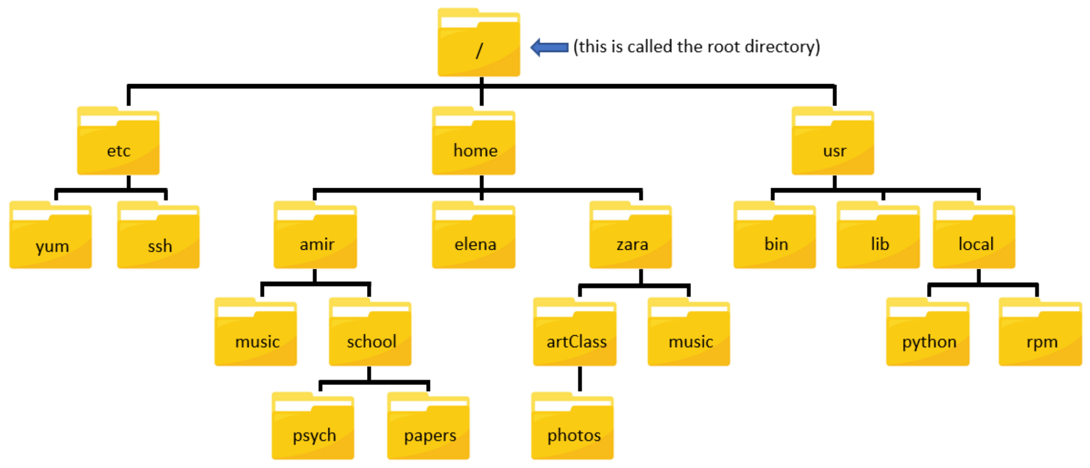


Figure 2.18 An example of some of the folders in a typical Linux install.

Absolute path examples

A. To move from any directory to the root directory the command would be:
```
cd /
```

B. To move from any directory to the /etc directory the command would be:
```
cd /etc
```

C. To move from any directory to the /etc/yum directory the command would be:
```
cd /etc/yum
```

D. To move from any directory to the /usr/local/python directory the command would be:
```
cd /usr/local/python
```

E. To move from any directory to the /home/zara/artClass/photos directory the command would be:
```
cd /home/zara/artClass/photos
```

Relative path examples

A. To move from the root directory to the /etc folder the command would be:
```
cd etc
```
Note – since etc is a sub-directory of the root directory you could also use:
```
cd /etc
```

B. To move from the root directory to the /etc/yum folder the command would be:
```
cd etc/yum
```

C. To move from the /etc directory to the root directory the command would be:
```
cd ..
```

D. To move from the /etc directory to the /home/amir directory the command would be:
```
cd ../home/amir
```

E. To move from the /etc directory to the /home/amir/music directory the command would be:
```
cd ../home/amir/music
```

F. To move from the /home/amir/music directory to the /home/amir/school directory the command would be:
```
cd ../school
```

G. To move from the /home/amir/school/psych directory to the /home/amir/school/papers directory the command would be:
```
cd ../papers
```

H. To move from the /home/amir/school/papers directory to the /home/amir directory the command would be:
```
cd ../..
```

I. To move from the /home/amir directory to the /home/zara directory the command would be:
```
cd ../zara
```

J. To move from the /home/amir/music directory to the /usr directory the command would be:
```
cd ../../usr
```

Note – in this case it would be simpler to use the absolute path and the command:
```
cd /usr
```

K. To move from the /home/amir/music directory to the /usr/local/rpm directory the command would be:
```
cd ../../usr/local/rpm
```

Note – in this case it would be simpler to use the absolute path and the command:
```
cd /usr/local/rpm
```

Examples using the ~ (home directory) shortcut
This set of examples will demonstrate using the ~ shortcut for your home directory. In this case assume that your home directory is /home/zara

A. To move from any directory to your home directory the command would be:
```
cd ~
```

B. To move from any directory to /home/zara/artClass the command would be:
```
cd ~/artClass
```

C. To move from any directory to /home/zara/artClass/photos the command would be:
```
cd ~/artClass/photos
```

The Linux UNIX File System
The Linux/UNIX file system is similar in many ways to FAT and NTFS, the file systems used by Windows; or the file and folder system used on the MAC. While there are technical differences in the way the file systems are structured and implemented that are of concern to system programmers or system administrators, to a user the Linux filesystem should look very familiar to a Windows or Mac user.

Like on Windows, the Linux/UNIX file system uses directories to organize files, and these directories can contain their own sub-directories. Of course, there are also a couple of significant differences that users will immediately see.

The first difference is that Linux/UNIX has a single top directory "/" which is called the root directory or just root. Other filesystem implementations, such as Windows may have multiple top directories. That is, they'll assign each drive a letter such as A: B: C: etc. and have one top level directory for each logical/physical drive mounted on the system. Linux and UNIX on the other hand don't use drive letters for different drives. Instead, they mount each drive as a subdirectory somewhere under the single root directory. For example, if you connect a USB drive to a Linux system it will be mounted in a sub-directory with a name like /media/usb.

Figure 2.19 A Highlander meme (Christopher Lambert)

The fact that the Linux filesystem has a single top level directory is very important. Remember that it's like the old Highlander movies … There can be only one.

The second difference between the Linux/UNIX filesystem and the Windows filesystem is that the directory names are not the same. Although Linux/UNIX systems have directories that are very similar in function to Program Files, WINNT or Windows, and Documents and Settings or Users, the names are different.

If you were to do an `ls -al` on the root (top level) directory of a Linux system you would see something like the following. (NOTE – your results will be similar but will probably not be exactly the same.)

```
drwxr-xr-x  18 root     root          1024 Apr 18 14:24 ./
drwxr-xr-x  18 root     root          1024 Apr 18 14:24 ../
dr-xr-xr-x.  5 root     root          4096 Oct 30  2019 boot
drwxrwxrwt   3 root     root          1024 Apr 18 16:15 etc
drwxr-xr-x   2 root     root          1024 Sep  4 03:23 mnt
drwxr-x--x   3 root     root          1024 Apr 16 09:44 proc
drwxr-xr-x   2 root     bin           2048 Apr 16 14:36 sbin
drwxrwxrwt   3 root     root          1024 Apr 18 20:15 tmp
drwxr-xr-x  18 root     root          1024 Apr 18 14:27 usr
drwxr-xr-x  14 root     root          1024 Apr 16 14:36 var
drwxr-xr-x. 99 root     root          4096 Jun 13 17:41 home
```

Those entries which are (sub) directories are preceded with a **`d`** character. If this were to be drawn or envisioned graphically it would look like

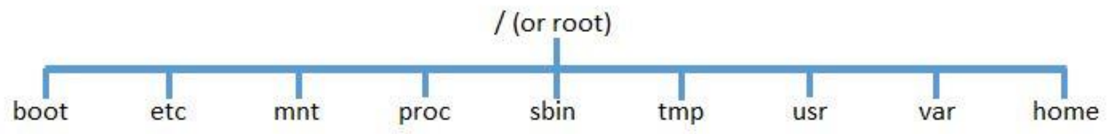


Figure 2.20 A graphic depiction of some of the top level Linux folders or directories.

Note – one of the things that confuses many Linux users, both new and experienced, is all the different bin and sbin folders used to store programs. Here's a quick explanation that may help.

/bin	This directory contains executable programs which are needed in single user mode and to bring the system up or repair it.
/sbin	Like /bin, this directory holds commands needed to boot the system, but which are usually not executed by normal users.
/usr/bin	This is the primary directory for executable programs. Most programs executed by normal users which are not needed for booting or for repairing the system and which are not installed locally should be placed in this directory.
/usr/sbin	This directory contains program binaries for system administration which are not essential for the boot process, for mounting /usr, or for system repair.

Like almost any subject in Linux and Compuer Science, there's a lot more to know about this. If you want to know more about the differences between the different bin and sbin directories you can do your own research and go down the rabbit hole. Just keep in mind that at this point you really just need a general idea of what the main Linux directories are used for. You don't need to memorize all the directory names and what they store.

What drives are connected to the system – the `df` and `lsblk` commands

As you look/snoop around any computer one of the things you might want to know is what hard drives or SSDs or removable drives are connected and available. On Windows you can easily find a list of all the storage devices using Windows Explorer, and each disk and device is assigned its own drive letter.

With the Linux and UNIX command line this information is a little trickier to find and interpret. One of the problems with the Linux/UNIX file system is that the drives and devices are not given their own drive letters. Remember that instead they're "mounted" to a folder somewhere in the main file system. So it's not obvious where the various drives or partitions have been mounted in the file system. Or for that matter, finding the removable drives and devices such as CD/DVD drives, USB drives, or floppy drives.

The second problem is that there are ways to display the drive information in Linux/UNIX, but it's not quite as user friendly and you need to know how to interpret the drive and device names.

There are a couple of commands that you can use to find the drive information but the df command is probably the easiest to use. The `df` (disk free) command shows you where different devices and partitions have been mounted as well as how much disk space is available on the various partitions and drives.

Here's an example of the output from the `df` command:

```
Filesystem          1K-blocks     Used  Available Use% Mounted on
/dev/sda1            37640236  4468804   33171432  12%  /
/dev/sda2              508588   296036     212552  59%  /boot
/dev/sdb1            18376704  5758340   12618364  32%  /home
```

The first column, labelled Filesystem, shows the disk or disk partition being used. This is the most user unfriendly part of the display, but once you know how to read this it should make sense. All the physical devices will start with /dev, and then have a string of 4 or more characters that identifies the specific device. The first two or three letters define the type of device. In the example all the devices start with **sd**, which used to stand for SCSI hard drive, but is now used to identify almost all drives including USB drives, SSDs, and both SCSI or SATA/PATA drives.

The next letter is the identifier for the physical drive, with **a** being the first drive, **b** the second drive, **c** the third drive etc. The last number identifies the partition on the drive. For example **sda1** is the first partition on the first drive, and **sda2** is the second partition on the first drive.

The example df output shows that there are two hard drives, **sda** and **sdb**, and that **sda** is divided into two partitions, **sda1** and **sda2**.

The next three columns of `df` output show the size of the drive or partition, how much is used, and what percent of the drive space is still free. The sizes are given in the number of blocks, where each block is 1024 bytes. These numbers are probably easier to understand in MBytes and GBytes, which you can approximate if you're a math fiend by dividing by 1000 for MBytes or 100000 for GBytes. Or an easier way to get user friendly numbers is to use the `-h` option with `df`, which will show the numbers in MBytes and GBytes. For example running `df -h` on the same system as above results in:

```
Filesystem              Size  Used Avail Use% Mounted on
/dev/sda1                36G  4.3G   32G  12% /
/dev/sda2               497M  290M  208M  59% /boot
/dev/sdb1                18G  5.5G   13G  32% /home
```

The last column shows what we really want to see, which is where the drive is mounted in the file system. The folders shown in the `df` output are called the mount point for each drive. In this case the second hard drive is mounted to the /home directory. If you move to the /home directory, or any directory below /home like /home/tests/test3, you'll actually be on the second hard drive.

The following image depicts how the drives or partitions are mounted in the filesystem. Notice that since /home is a mount point for /dev/sdb1, all the subdirectories of /home will be on the /dev/sdb1 drive. That is, as soon as you move into the /home directory you'll be on the sdb1 drive. And if you move into any subdirectory of /home or any of their subdirectories you'll be on the sdb1 drive.

You can think of this like changing drives in Windows by specifying a different drive letter. For example, if Windows assigns E: to the sdb1 drive, you'll be on the sdb1 drive any time you move to the E: drive. But in Linux and UNIX there are no drive letters, instead you change drives by changing to the directory where the drive is mounted in the filesystem.

Note that in this example if you move to the /boot directory you'll also move to a different drive partition. The last thing to note is that any folders that are not under /boot or /home will be on the /dev/sda1 drive.

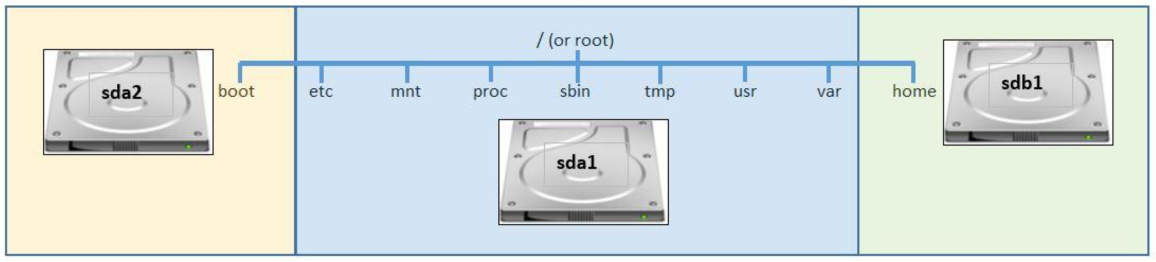


Figure 2.21 How disks are mounted to directories in the main filesystem, as opposed to being given separate drive letters.

The `lsblk` command is another command that can be used to view the connected drives and partitions. The name `lsblk` will make more sense if you think of it as a combination of the `ls` command and `blk`, which stands for block level devices. In Linux and UNIX devices are categorized as either block level, where a large block of data is read or written, or character level, where a single character is read or written. All the drives we want to see, like hard disk drives or SSDs are block level devices.

The following is an example of the output from `lsblk`:

```
% lsblk
NAME                       MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda                          8:0    0    36G  0 disk
├─sda1                       8:1    0    36G  0 part /
└─sda2                       8:2    0  497M  0 part /boot
sdb                          8:0    0    60G  0 disk
└─sdb1                       8:1    0    60G  0 part /home
```

This makes the relationship between the disks and partitions a little more obvious. But in any case, the important thing is that it shows the directories where the disks are mounted in the filesystem.

Removable Drives
As a user you probably don't really care what hard drives and partitions are being used, or where they're mounted. But, it is helpful to know where removable drives like USB thumb drives or external drives are mounted. Again, think of a computer running Windows. If you want to copy files to or from a thumb drive you'll need to know the drive letter. With Linux you'll need to know where the thumb drive is mounted in the file system, so the `df` or `lsblk` commands can be very helpful.

What's going on? The Linux/UNIX equivalent to task manager

As you continue snooping around a computer system one of the next things you might want to figure out is what programs or processes are running. On Windows systems you can run Task Manager and find out this information in a couple different levels of detail. On Linux/UNIX systems the `ps` (process status) command and the `top` command are used to show what's currently running. The `ps` command, without any command options, will show what processes you have running. You'll always see at least 2, the `ps` process and your shell process, but there may be others. The `top` command shows the information for all user and system processes, so like Task Manager in Windows, there's a lot more information. The following shows sample output from the `top` command.

```
top - 14:58:19 up 182 days, 23:53,  1 user,  load average: 0.01, 0.03, 0.05
Tasks: 116 total,   1 running, 115 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.2 sy,  0.0 ni, 99.8 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  7999996 total,  1036120 free,  4468396 used,  2495480 buff/cache
KiB Swap:  6291452 total,  6288360 free,     3092 used.  2706892 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
35057 bdover    20   0  154664   2516   1168 S   0.3  0.0   0:00.17 sshd
```

```
    1 root      20   0  128108   4932   2928 S   0.0  0.1  31:37.99 systemd
    2 root      20   0       0      0      0 S   0.0  0.0   0:05.48 kthreadd
    4 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 kworker/0:+
    6 root      20   0       0      0      0 S   0.0  0.0   0:12.48 ksoftirqd/0
    7 root      rt   0       0      0      0 S   0.0  0.0   0:04.47 migration/0
    8 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcu_bh
    9 root      20   0       0      0      0 S   0.0  0.0  10:43.29 rcu_sched
   10 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 lru-add-dr+
   11 root      rt   0       0      0      0 S   0.0  0.0   0:54.67 watchdog/0
   12 root      rt   0       0      0      0 S   0.0  0.0   0:41.86 watchdog/1
   13 root      rt   0       0      0      0 S   0.0  0.0   0:04.76 migration/1
   14 root      20   0       0      0      0 S   0.0  0.0   0:02.86 ksoftirqd/1
   16 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 kworker/1:+
   18 root      20   0       0      0      0 S   0.0  0.0   0:00.00 kdevtmpfs
   19 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 netns
   20 root      20   0       0      0      0 S   0.0  0.0   0:03.22 khungtaskd
```

The `top` command runs continuously until you stop it, dynamically updating the information as it changes. The top half of the screen contains statistics regarding open processes and resource usage, and the bottom half of the screen displays the dynamic list of currently running processes and their CPU usage. To stop or exit the `top` command hit the `q` key for quit.

Like Task Manager in Windows, this information isn't very exciting and isn't crucial information for a casual user. But the information and commands will be important and useful as you move into system administration.

Who are you?

When a user is given an account on a Linux/UNIX system there are several pieces of information about that person that are created. The `who`, `whoami` and `finger` commands can be used to find out information about yourself like your username (in case you forgot), where you logged in from, and how long you've been on the system. The following show example output from these commands:

```
% whoami
bdover

% w
 15:01:04 up 182 days, 23:56,  1 user,  load average: 0.00, 0.01, 0.05
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
bdover    pts/0    077-099-168-216  06:44   0.00s  0.07s  0.00s w

% who
bdover    pts/0        2021-08-17 06:44 (077-099-168-216.res.charter.com)

% finger
Login   Name  Tty    Login Time   Office      Phone    Host
bdover  Ben   pts/0  Aug 17 06:44 The Shire  888-8888 077.099.168.216
```

Some of these commands can also be used to find out information about other users on the system. This information and these commands are a little fun, but once again this isn't crucial to know and you don't need to memorize any of these commands.

Fun with the `finger` command
In the early days of UNIX and the Internet, before the web, users wanted a way to share their contact information, schedules, and possibly let other people know where they were at a given time. The `finger` command was designed to allow you see this information for other users. By default, the `finger` command will show you your finger information. If you haven't changed it yet, there won't be much to see, but you will see the various categories of information that finger will display.

You can use the `finger` command to find out information about other users as well. Actually, this is what it was originally designed for, since you hopefully already know your own finger info. To give another user the finger (yes, this is really what it's called, LOL) type the command:

```
 finger user
```

It's theoretically possible to finger users on other Linux/UNIX systems if you know their usernames and the hostname. The syntax for this is: `finger user@hostname`.   However, this usage of `finger` causes some security issues, so most Linux/UNIX administrators have disabled remote finger access.

Changing your finger information
You can change your own finger information by using the `chfn` command. It will prompt you for your password, then prompt you for your contact information.

```
% chfn
Changing finger information for bdover.
Name [Bilbo Baggins]: Muad'Dib
Office [The Shire]: Arrakis
Office Phone [888]: 999-888-7777
Home Phone [#7]: 222-111-0000

Password:
```

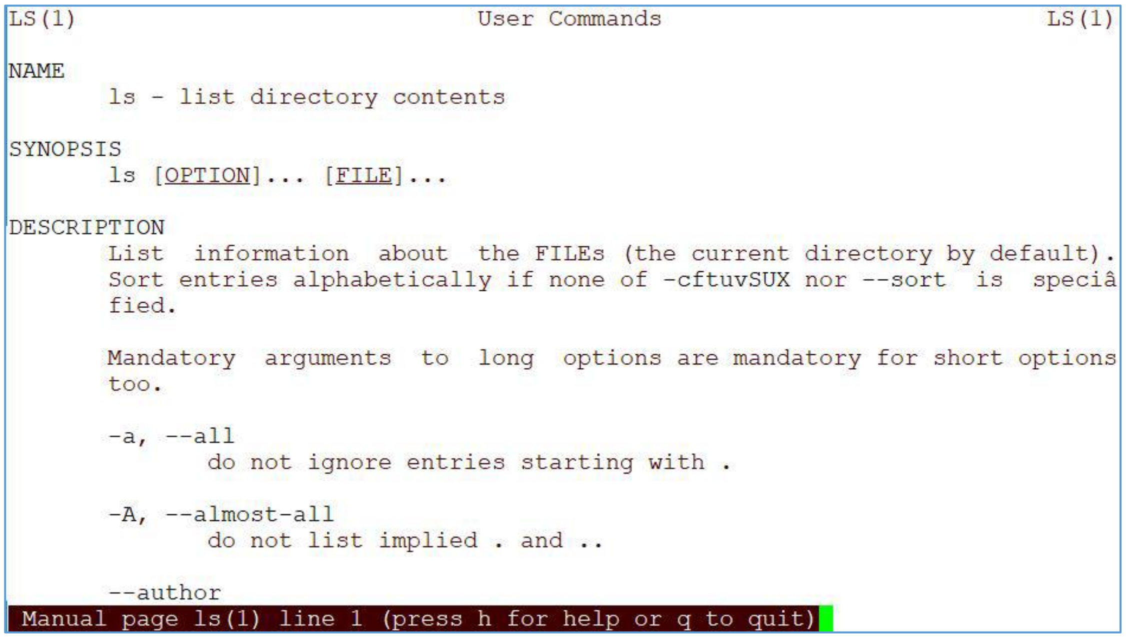
You can also change your plan by creating or editing a file named `.plan` that must be located in your home directory. (You will learn how edit your .plan file by moving files from your PC to your Linux/UNIX account in the next section, and how to use the editor later in the class.)

Getting Help

There are a couple of ways for getting help on Linux/UNIX systems. Of course, the most common method of getting help is to use the Internet. But there may be times when you need help and don't have Internet access. In those cases, you should know how to use some of the help systems built into Linux/UNIX.

The first method is to use a series of files called the *man pages* that contain information about all the various commands, programming libraries and data files on the system. The Linux/UNIX documentation used to come in a series of manuals, and the online man pages are simply the electronic versions of the hardcopy manual pages. The man pages are in a special format, not plain text, so you need to use a special utility program, called `man`, to look at them. The command name isn't meant to be sexist, it's just an abbreviation of manual.

To use the `man` command type `man` followed by the name of the command you want help with. For example, to see the man pages for the `ls` command type `man ls`



```
LS(1)                         User Commands                        LS(1)

NAME
       ls - list directory contents

SYNOPSIS
       ls [OPTION]... [FILE]...

DESCRIPTION
       List  information  about  the FILEs (the current directory by default).
       Sort entries alphabetically if none of -cftuvSUX nor --sort  is  speciâ
       fied.

       Mandatory  arguments  to  long  options are mandatory for short options
       too.

       -a, --all
              do not ignore entries starting with .

       -A, --almost-all
              do not list implied . and ..

       --author
 Manual page ls(1) line 1 (press h for help or q to quit)
```

Figure 2-22 The output from running `man ls`

The `man` command will display the manual page(s) for the specified command (or specified item). It displays the information a page (screen) at a time, along with a prompt. The "`:`" at the lower left of the screen represents the prompt for the `man` command. The `man` commands make it possible to do things like display the next page, go back, search for a string, etc.

At a minimum you need to know that pressing `<space>` (the space bar) will display the next man page, and to quit from `man`, you have to type `q`. To see a list of all the `man` commands you would think that you should type:

```
man man
```

At least this is what I always thought. However, as it turns out the man page for the `man` command doesn't show the different keys to hit; it shows you other information, but not the command keys. It also turns out that the `man` command uses a Linux/UNIX utility named `less` to control the scrolling and display of the man page. The `less` utility is similar to the `more` utility, and the name is another UNIX insider pun/joke. So, to see the keys you would hit to control the scrolling in the `man` utility you should type: `man less`

Help finding the right command with `apropos`

The problem with the `man` command is that it requires you to know the name of the command you want to use. It's like the problem of using the dictionary to help you spell a word. You can only look up the spelling for a word if you already know how to spell it. So, if you know you want to do something in Linux but don't know the command name then `man` is no help. Luckily most Linux/UNIX systems also contain a database of the available commands and keywords associated with the commands. There are two commands you can use to get help from the command database.

The first command is `apropos`, which will search the keywords and find any commands that match. To use apropos you type the command name followed by a word or set of words that describe what it is you would like to do. For example, to find commands that would edit files type `apropos edit` The `apropos` command searches through its database and returns all the commands that have the characters **edit** in them. Here's a small sample of the results from `apropos edit`

```
elfedit (1)          Update the ELF header of ELF files.
ex (1)               Vi IMproved, a programmers text editor
gex (1)              Vi IMproved, a programmers text editor
grub2-editenv (1)    Manage the GRUB environment block.
gview (1)            Vi IMproved, a programmers text editor
```

Note that sometimes `apropos` won't be much help. That is, some commands, such as `cd`, are actually part of a larger program called the shell. You'll learn much more about the shell later in the course, but for now you can think of it as a program like Word. Your shell program starts when you login, and it's the program that handles any commands you type. If the command you type, like `cd`, is built into the shell code, the shell just executes the code for that command. This is like running Word, and then asking it to find and replace some text. Word has the code to do this built-in and won't have to start another program. But if the command you type, like `cat`, is not built into the shell code, the shell will ask the OS to load and run the command for you. This is like running Word, and then wanting to play a music file, in which case you'll need to start another program. The point of this is that if you type `apropos cd` the information returned won't be of much help, and until you get a little more Linux experience it will probably just be confusing.

The other problem with `apropos` is that it does a "dumb" search of all the man pages and returns any line from any page that contains the characters you passed to `apropos`. For example, if you type `apropos ls` it returns dozens of lines, matching things like `fai`**`ls`**`afe` or `Uti`**`ls`**. In this case what's returned most likely won't be helpful.

If you give `apropos` more than one keyword it will find any commands that match any of the words in your list. There is no way to make `apropos` match all the words in a phrase, but later you will learn how to combine commands to make this happen. As a side note, if you use the `man` command with the `-k` option (for keyword) it will act just like `apropos`.

Is this the right command? Using `whatis`

The second utility that uses the command database is `whatis`, which is like the `man` command but returns a single line result instead of the entire man page. It does this by searching the command database and then returning the single line description for the command. This can be helpful when you think you know what a command does but may not be sure. You can use `whatis` to check your memory, without having to read the entire man page. The only problem is that just like the `man` command, `whatis` requires that you remember a valid command name to even get started.

Here's an example showing the output from running `whatis less`:

```
% whatis less
less (1)              - opposite of more
less (3pm)            - perl pragma to request less of something
```

What Version of Linux/UNIX is running?
Another piece of information that you might want to know is which Linux distribution is being used. You already know you're using Linux, but do you know which distribution? Or which version of the specific distribution? Just like with Windows, where most users don't know or care whether they're running Windows Home or Windows Enterprise, the information about the Linux distribution isn't crucial to know as a user. But it will be important as you move into system administration as there are some significant differences between distributions. Just in case you want to know which specific release of Linux/UNIX is being run there is one command, `uname`, and one file, `/etc/issue` that will display this information. There's a second file that may exist on some systems but is not universally available. On RedHat systems this file is named `/etc/redhat-release`

Summary
In this section you learned how to do the same kind of things you might do if you went to a friend's house and had 10 minutes to snoop around or explore their Windows based computer, only in this case the computer is running Linux or UNIX. Specifically you learned the following:

1. How to use the `ls` command to look at the files in a directory
2. How to use the `cd` command to change directories
3. The difference between absolute and relative paths, and how to use them
4. The general structure of the main directories in the filesystem
5. Various ways to get help with Linux commands
6. Commands for discovering what other users may be on the system, and what programs or processes may be running.

Review Questions

The purpose of these questions is to help you review your understanding of the material and exercises presented in this chapter. You should look at the questions, but you do NOT need to turn in the answers.

1. How do your commands get from the PC to the Linux server?
2. Is there a program running on the Linux server to handle your incoming network connections?
3. Can two Linux/UNIX users have the same account name? Can two Linux/UNIX users have the same password?
4. When you run the PuTTY program, its code/instructions are loaded into memory on a computer. Is this computer the Windows machine you're sitting at, or the Linux computer?
5. When you log on to the Linux computer, and run a program such as `passwd`, where is this program run, on the Windows machine you're sitting at, or the Linux computer?
6. What are some of the features of a strong password?
7. On a Linux system, are there any programs running besides user programs?
8. Do all programs have to be run by someone? That is, does a user need to start every program running on a computer?
9. Will the files and directories on every Linux/UNIX system be exactly the same? Will they be similar?
10. Is it better to use absolute paths or relative paths?
11. What character is used to start all absolute paths?
12. How to move up one directory? That is, how do you move to the parent directory?
13. On Linux systems the main OS file is named Vmlinuz-*something*, and it's typically located in the /boot directory. What is the specific name of this file on the CBC Linux server? How big is it? Do you think this file contain the entire OS? Can you name the Windows equivalent of this file?
14. Linux/UNIX supports multiple disks and/or partitions. How does Linux/UNIX identify each partition, as a drive letter, or as a subdirectory? Is this better or worse than drive letters? What are some advantages and disadvantages?
15. How do you know if a subdirectory is on a different disk or partition? Does it matter?

16. Can you see what other users are doing on a Linux system? Can they see what you are doing? Can you find out any personal information about other users?

17. What are some useful options for the `ls` command? How do you find a complete list of options?

18. Can you use `cd` or `ls` on every directory? Why or why not?

19. Use `ls -al` on your home directory. What files do you see? What are . and .. ?

20. Can you identify the other files in your home directory? Why don't they show up under `ls`?

21. How do you get help on Linux/UNIX commands?