Foundations of Cyber Security

Linux

Command Line Interface

AA Sako

LINUX LAB MANUAL

COMMAND LINE INTERFACE

FOUNDATIONS OF CYBER SECURITY

A.A. SAKO

Preface

This book is a Lab Manual containing exercises that provide invaluable hands-on experience with the Linux and UNIX command line interface (CLI). While it's possible for some people to become proficient in a subject like Linux by simply reading about it, most people will see a tremendous increase in the information they can absorb and retain if they can put what they learn to practice. This Lab Manual was designed as a companion to the book **Linux and UNIX – Command Line Interface**, which contains detailed explanations about Linux and UNIX concepts and commands.

This Lab Manual has been designed for individuals who are beginning with Linux and UNIX, and presents the material in a logical order for someone who is starting out with Linux. That is, the material starts by assuming that the person using the book has no previous experience. This is much different than the many Linux books that are aimed at preparing someone for passing a certification exam such as Linux+ or LPIC. These certification books assume that the person reading the book already has experience and present the material in the same order as the modules in the exam.

While this Lab Manual can be used as an aide in preparing for certifications such as Linux+ or LPIC its primary purpose is to help those who are new to Linux and UNIX and want to learn how to use the command line.

Objectives

The main objective of this book is to help individuals prepare for working in Linux or UNIX Administration to learn how to use the Linux and UNIX command line. The exercises in this Lab Manual provide hands-on experience typing commands and combining individual commands in what are called command pipelines. The commands and command pipelines are the basis of programs called shell scripts, which are widely used in Linux Administration and Security. To accomplish this objective, the Lab Manual provides exercises that reinforce the following concepts, skills and tools:

Basic Commands and Skills

- Connecting and logging in to a Linux/UNIX System
- Changing passwords
- Changing directories and listing directory contents
- Using absolute vs relative paths
- The general file system layout
- Viewing information about users and processes
- Getting help with commands
- Creating, renaming, and deleting files and directories
- Sharing files with Windows based systems, and dealing with different End of Line characters
- Interpreting and setting file and directory permissions
- Changing default file and directory permissions
- Using the vi editor, including ex mode

Working with the shell

- Selecting a shell
- Filename wildcards (globbing)
- Protecting special characters with quotes and whacks
- Running multiple commands with ;
- Identifying stdin and stdout
- Command redirection
- Commands pipelines
- Displaying, setting, changing and deleting shell variables
- Aliases
- Shell startup files
- Working with the shell search path
- Displaying, using and saving the shell command history
- Job processing
- Using pipeline commands such as cut, sort, grep, etc.
- Creating and reading regular expressions

Shell Programming

- Specifying the shell interpreter (shebang)
- Adding comments
- Debugging shell scripts
- Producing output with echo
- Adding shell commands
- Reading user data
- Using variables
- Performing calculations with expr and bc
- Making decisions with if statements
- Boolean Logic
- For loops
- While and until loops
- Utilizing shell script repositories

Features

This Lab Manual has several features that make it unique including:

Graduated Exercises – The exercises are graduated, starting with very easy multiple choice and short answer problems that can be used to check comprehension. The exercises then progress into "type this" exercises, which allow you to become familiar with the various commands and their options. The last level of exercises require you to solve problems that you may see in real life, providing an opportunity to put what you've learned into practice. Each section also contains review questions, which provide a way to test comprehension.

Companion Website and Videos – People learn in different ways, and some can pick up all the knowledge they need just by reading the text in this book. But there are also some concepts that are much simpler to understand if you can see them demonstrated, as opposed to reading through a dense explanation. The book has a companion website full of videos that explain concepts or show you how to use the commands you'll learn about. The videos provide another resource to help you learn, and can either reinforce what's in the book, or provide an alternate explanation that's easier to follow than the same written explanation. The videos are broken into small digestible chunks, each explaining a single concept, as opposed to being hour long recordings of someone standing at a whiteboard and lecturing. This makes it easier to find the information you need as you can simply pick the appropriate video as opposed to having to search through a longer lecture video to find the correct spot.

Companion Book – While this Lab Manual can be used in any classroom and with any learning resources such as books or videos, it has been designed to be a companion to the textbook, **Linux Command Line Interface.** The textbook, which is available for purchase at TonySako.com, fully explains the concepts and commands used in the assignments and exercises in this Lab Manual.

Acknowledgements & Dedication

I would like to give thanks to Richard Grandy who gave me my first UNIX account in 1983, Ron Stephens for my first job assignment as a UNIX network admin, Rich Cummins for trusting me to work as his CIO, and to the thousands of students who have taken my classes and helped me shape and improve the material in this book.

I would like to give special thanks to my parents Jim and Stella, Sara and Dee for lovingly shaping me into the nerd I am today, and to my brother Gus and my sister Kate for taking me seriously every once in a while. Finally, I dedicate this book to my boys Noah and Josh; and especially to my wonderful wife Yvonne who still listens and acts interested to my most obtuse, nerdy and boring explanations.

Additional Materials

There are additional student resources that can be found at my website http://tonysako.com including:

- Videos that explain much of the background material as well as demonstrate how to use the commands and command options
- Data files used in the exercises, along with installation notes
- Answers to the questions in the assignments and exercises

Table of Contents

Introduction to Linux and UNIX

The exercises in this chapter have been designed to reinforce your knowledge and prepare you to do the following:

1. Define the major functions provided by an Operating System.
2. Describe the similarities and differences between UNIX and Linux.
3. Provide a description of the Linux and UNIX OS.
4. Compare and contrast Linux/UNIX to other operating systems.
5. Compare features and the main directories in the Linux/UNIX file system to the Microsoft Windows file system.

1.1. Find out who owns UNIX. Note that this is asking about UNIX, not Linux. You will probably have to do some searching on the Internet to find the answer. Enter your answer in the space provided:

____

1.2. Do some research and find information about at least three different flavors of Linux or Linux Distributions. If possible, discern how are they different from UNIX? How are they different from each other? How are they similar? Enter at least 3 URLs that you found during your research:

____

____

____

1.3. Do some research and find out how much it costs to purchase a version of UNIX, and how much it costs to purchase a version of Linux.

UNIX purchase price: ______________________________

Linux purchase price: ______________________________

1.4. Find at least three web sites that have UNIX/LINUX basic tutorials. Enter their URLs.

____

____

____

1.5. See if you can discover what version of UNIX/LINUX is being run on the CBC server. The answer isn't critical, so don't worry if you can't find it. What is critical is that you're able to use the Internet to search for Linux/UNIX commands that will perform a specific function.

____

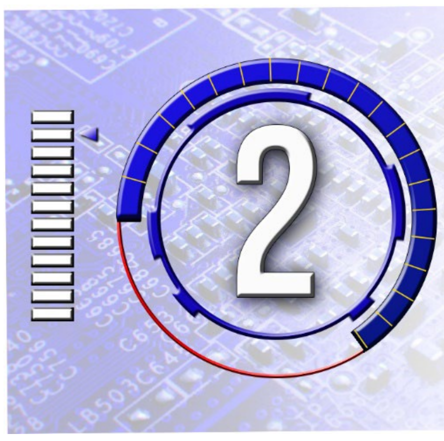
1.6. True or False. Linux can be run on any computer that has hardware capable of running Microsoft Windows. That is, Linux does not require any special hardware.

1.7. True or False. You must know how to type commands to use Linux as it does not have a GUI or desktop interface like Windows, Linux only has a Command Line Interface.

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. List the major tasks provided by any Operating System.
2. Name two other Operating Systems, besides UNIX/Linux.
3. Why are UNIX and Linux important? That is, why should you bother learning them?
4. Name some strengths and weaknesses of Linux.
5. Would you rather have OS source code, like Linux, or get it from vendor, like Windows? Name the advantages and disadvantages of each.
6. Is it easier to copy a file by drag and drop, or by typing a command? That is, would you rather use a GUI interface or a command line interface?
7. How big is the Linux OS vs Windows? That is, what are the minimum disk space and memory required to install and run Linux? What part of an OS requires the most disk space and memory, a command line interface where the user types commands, or the GUI interface?
8. Who owns UNIX?
9. Who owns Linux?
10. Where was UNIX invented?
11. Where was Linux invented/developed? Who was the initial developer?
12. What is relationship between Linux and UNIX?
13. Name the advantages and disadvantages to running Linux vs. running a commercial version of UNIX.
14. What new concerns appear when sharing a computer system between many users, as opposed to a typical Windows home system where there is typically one user at a time using the computer?



Getting Started: Logging In and Basic Navigation

The exercises in this chapter have been designed to reinforce your knowledge and provide hands on experience connecting to a Linux/UNIX server and using commands so you can do the following:

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.

2.1 Login to the Linux Server

This set of exercises has been designed to provide you experience using ssh to connect to a Linux/UNIX server, and then login. You can either use ssh from a Command Window, or use an application like PuTTY to make the connection and login. Detailed instructions for using ssh from a Command Window or using PuTTY can be found in the text book or on the website TonySako.com.

Before starting this assignment, ensure you have the following three pieces of information which are required to access your account.

(1) **Username** - This is your individual unique identification for the CBC Linux server. The Linux server is case sensitive, so make sure that you understand exactly which letters are upper and lower case.

(2) **Password** - You will be given an *initial* password that you ***must change*** once you login. When you change your password, the Linux server will force you to choose a strong password, which means: at least 8 characters long, a mix of alphanumeric and non-alphanumeric characters, a mix of upper and lower case alpha characters, and no dictionary words.

(3) **Linux Server DNS Name** - This is required to make the network connection between your computer and the Linux server.

Do **NOT** forget your username or password or you will be unable to access the Linux server. I know that in this day and age, and during your classes, you probably have to set up new accounts on a regular basis, so it can be hard to keep track of them all. If you are the type of person who tends to forget your passwords, and if you don’t have a secure way to store them, I suggest you write them down in the space provided below.

Username: ________________________________
Password: ________________________________
Host Name: cs223linux.columbiabasin.edu

A. Using the method of your choice use ssh to connect to the Linux Server and login.

- Remember that the first time you make the network connection you will be asked about storing and using the ssh encryption key. Make sure and do this.

- When you see the login prompt enter your user name. Remember that Linux and UNIX are case sensitive, and your username is all lowercase. Ensure that the caps lock is off and make sure you use lower case for everything unless you are absolutely sure you want upper case.

- When you see the password prompt enter your password. Once again, remember that Linux and UNIX are case sensitive. Also remember that for security reasons some systems don't display anything as you type your password. If this happens simply type your password and hit the `<Enter>` when finished.

- If you enter your username and password correctly you will be logged in. You may see some messages from the system followed by a command prompt, or you may just see the command prompt. The prompt is the systems way of telling you that it's waiting for you to type a command. On the CBC Linux server the default prompt includes your username and the name of the host and will look something like the following, where the *nn* is your unique number:

```
[studentnn@cs223linux ~]%
```

- Create a screenshot showing you successfully logged in.

B. Once you have successfully logged in to a system, the first thing you should do is change your password. This is done via the **`passwd`** command. You will be asked for your old password, then twice for your new password to ensure you typed it in correctly. At the prompt type: `passwd`. When the passwd utility runs the dialog will look something like the following. The items you need to type are displayed in bold text.

```
username@ cs223linux ~] % 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 you type in and will prompt you if your password doesn't meet some minimum parameters. For example, it must meet some minimum length, it must contain and upper case and lower case letter, etc.

Create a screenshot showing that you have run the `passwd` command and successfully changed your password.

C. The last thing you need to do is log out. This is done by typing the command: `exit` This will log you out and end your ssh network session. If you're running PuTTY it will also close the PuTTY window. There's nothing to turn in for this step.

2.2 Practice with `cd`, `ls` and relative and absolute paths

This set of exercises has been designed to provide you practice working with the `cd` and `ls` commands and relative and absolute paths. You need to be able to use these commands as the ability to change directories and know where you are in the filesystem are basic but critical skills. Just think how hard it would be for a Windows user if they couldn't use Windows Explorer to change folders and the only items they could access were those in their Documents folder.

To start this set of exercises login to your account on the Linux server and change to the **/home/cdPractice** directory by typing:

```
cd /home/cdPractice
```

Remember that Linux is case sensitive, and note that the **P** in **cdPractice** is capitalized.

For each of the following, either use the `cd` command to move into the specified directory, then use the `ls` command to display the files in the directory; or you can just use the `ls` command to list the files in the specified directory. Use the information from the output of the `ls` command to answer the question(s). Write your answer(s) in the space provided.

Note that the `cd` command will return an error message if you try to move to a directory that doesn't exist, which happens if you make a mistake typing the directory. For example, if I want to change into the directory **goodTyping** but misspell the directory name and type `cd badTyping` the error message will be:

```
badTyping: No such file or directory.
```

If you type the `cd` command and receive a similar error message it means you need to check how you typed the directory name and fix any errors. Also note that all the files you'll encounter in this exercise are empty because I just made them up for the exercises, so base your answers on the filenames, not the actual content inside each file.

A. Move to the sub-directory **BaseCamp** and run the `ls` command. How many files end with the **.txt** extension?

   ____

B. Move to the sub-directory **MountainTrail** and run the `ls` command. Note that you do NOT want to return to the /home/cdPratice folder, you should be in /home/cdPratice/BaseCamp when you start this. How many files end with the **.txt** extension?

   ____

C. Move to the sub-directory **MoosePeak** and run the `ls` command. Note that you should be in /home/cdPratice/BaseCamp/MountainTrail when you start this. How many files are pictures of wildlife? That is, how many files in this folder start with an animal name and end with **.jpg** or **.png**?

   ____

D. Move to the directory **/home/cdPractice/BaseCamp/MountainTrail/PumaRidge**. You can use either the absolute path or move up one folder and then down into PumaRidge. Once you're in the folder run the `ls` command. How many files start with the letter **p**?

____

E. Move to the sub-directory Waterfall. And run the `ls` command. How many files have the word **fall** as part of their name?

____

F. Move to the directory **/home/cdPractice/BaseCamp/MountainTrail/PumaRidge/Cliffs**. You can use either the absolute path or move up one folder and then down into Cliffs. Once you're in the folder run the `ls` command. How many files have the word **cliff** as part of their name?

____

G. Move to the directory **/home/cdPractice/BaseCamp/MountainTrail/BearRidge/GoatRocks** and run the `ls` command. How many files are **mp3** files?

____

2.3 Practice with `cd`, `pwd` and relative and absolute paths

This set of exercises has been designed to provide you practice working with the `cd` and `pwd` commands and relative and absolute paths. In each exercise you will use the `cd` command to move to a directory, and then use the `pwd` command to verify that you're in the correct location. You can use either relative or absolute paths to change directories, and you can make the change using one `cd` command or use multiple `cd` commands. The point of the exercise is to ensure that you can use the `cd` command to move around the filesystem. Remember that if you mistype the directory or path name the `cd` command will return an error. If this happens correct your typing and try again.

Note that while the instructions for each exercise direct you to create a screenshot showing you successfully completed the steps, you can create one screenshot at the end of all the exercises in this section. This will save you some work but still show that you were able to change to the specified directories. Let me know if `pwd` ever reports that you're in a different directory than the one you changed to, because this indicates you've found a rip in the time-space continuum.

A. Move to the directory **/home/cdPractice/BaseCamp/RiverTrail** and run the `pwd` command. Create a screenshot showing the results of the `pwd` command, which will show you are in the correct directory.

B. Move to the directory **/home/cdPractice/BaseCamp/RiverTrail/Camp2** and run the `pwd` command. Create a screenshot showing the results of the `pwd` command, which will show you are in the correct directory.

C. Move to the directory /**home/cdPractice/BaseCamp/RiverTrail/Camp2/SmallLake** and run the `pwd` command. Create a screenshot showing the results of the `pwd` command, which will show you are in the correct directory.

D. Move to the directory **/home/cdPractice/BaseCamp/RiverTrail/Camp2/Beach** and run the `pwd` command. Create a screenshot showing the results of the `pwd` command, which will show you are in the correct directory.

E. Move to the directory **/home/cdPractice/BaseCamp/RiverTrail/Camp2/OtterCreek** and run the `pwd` command. Create a screenshot showing the results of the `pwd` command, which will show you are in the correct directory.

F. Move to the directory **/home/cdPractice/BaseCamp/RiverTrail/BeaverDam** and run the `pwd` command. Create a screenshot showing the results of the `pwd` command, which will show you are in the correct directory.

G. Move to the directory **/home/cdPractice/BaseCamp/RiverTrail/BigLake/FrogPond** and run the `pwd` command. Create a screenshot showing the results of the `pwd` command, which will show you are in the correct directory.

2.4 Practice with `ls` and relative and absolute paths

This set of exercises has been designed to provide you practice working with the `ls` command and relative and absolute paths. In each exercise you will use the `ls` command to display a directory listing without actually moving to the directory. For example, if you're instructed to use `ls` to display a directory listing for **/home/cdPractice/jokes** you would use the command `ls /home/cdPractice/jokes`, as opposed to using the `cd` command to change to the directory and then running the `ls` command.

A. Use the `ls` command to create a directory listing for **/home/cdPractice/jokes/cartoons**. Create a screenshot showing the results of the `ls` command.

B. Use the `ls` command to create a directory listing for **/home/cdPractice/LOTR/elves**. Create a screenshot showing the results of the `ls` command.

C. Use the `ls` command to create a directory listing for **/home/cdPractice/hackingTools/malware**. Create a screenshot showing the results of the `ls` command.

D. Use the `ls` command to create a detailed directory listing showing all the files in **/home/cdPractice/hackingTools/malware**. (Remember that the `-a` option directs `ls` to show all files and the `-l` option tells `ls` to display the long or detailed information.) Create a screenshot showing the results of the `ls` command.

E. Use the `ls` command to create a detailed directory listing showing all the files in **/home/cdPractice/hackingTools/portmap**. Create a screenshot showing the results of the `ls` command.

2.5 More Practice with `cd` and relative and absolute paths

This set of exercises has been designed to provide you additional practice working with the `cd` command and relative and absolute paths. In particular these exercises should help you determine when and when not to use a / at the beginning of a path.

A. Assume you are in the folder **/home/cdPractice/jokes** and you want to move to the **/home/cdPractice/jokes/lawyer** folder. Which of the following would accomplish this?
   a. `cd lawyer`
   b. `cd /lawyer`
   c. `cd ../lawyer`
   d. None of the above

B. Assume you are in the folder **/home/cdPractice** and you want to move to the **/usr** folder. Which of the following would accomplish this?
   a. `cd usr`
   b. `cd /usr`
   c. `cd ../usr`
   d. None of the above

C. Assume you are in the folder **/home/cdPractice/LOTR** and you want to move to the **/home/cdPractice/LOTR/rings** folder. Which of the following would accomplish this?
   a. `cd rings`
   b. `cd /rings`
   c. `cd ../rings`
   d. None of the above

D. Assume you are in the folder **/home/cdPractice** and you want to move to the **/etc** folder. Which of the following would accomplish this?
   a. `cd etc`
   b. `cd /etc`
   c. `cd ../etc`
   d. None of the above

2.6 More Practice with `cd`, `ls` and relative and absolute paths

This set of exercises has been designed to provide you additional practice working with the `cd` and `ls` commands and relative and absolute paths. In each exercise you will be told where you're currently located in the filesystem, and then directed to provide the command to either move to a directory or list the content of a directory. You can use either relative or absolute paths, but you must only use a single command. For example, if you're currently in the /home/cdPractice directory and the instructions say to display all of the files in the /etc/yum directory the answer would be:

```
ls -al /etc/yum
```

or if you want to do it the hard way:

```
ls -al ../../etc/yum
```

A. Assume you are in the folder **/home/cdPractice/jokes**. What command would you use to move to the **/home/cdPractice/jokes/lawyer** folder?

B. Assume you are in the folder **/home/cdPractice/jokes/lawyer**. What command would you use to move to the **/home/cdPractice/jokes** folder?

C. Assume you are in the folder **/home/cdPractice/jokes/lawyer**. What command would you use to move to the **/home/cdPractice/jokes/chuckNorris** folder?

D. Assume you are in the folder **/home/cdPractice/jokes/lawyer**. What command would you use to move to the **/home/cdPractice/LOTR/mordor** folder?

E. Assume you are in the folder **/home/cdPractice/jokes/lawyer**. What command would you use to move to the **/home** folder?

F. Assume you are in the folder **/home/cdPractice/hackingTools**. What command would you use to see a detailed list of all the files to the **/home/cdPractice/hackingTools/passwordCracking** folder?

G. Assume you are in the folder **/home/cdPractice/hackingTools/passwordCracking**. What command would you use to see a detailed listing of the **/home/cdPractice/hackingTools** folder?

H. Assume you are in the folder **/home/cdPractice/hackingTools/passwordCracking**. What command would you use to see a detailed listing of the **/home/cdPractice/hackingTools/injection** folder?

I. Assume you are in the folder **/home/cdPractice/hackingTools/passwordCracking**. What command would you use to see a detailed listing of the **/home/cdPractice/jokes/cartoons** folder?

2.7 The treasure hunt

This exercise has been designed to provide you experience applying what you've learned about `cd`, `ls` and relative and absolute paths. The goal of this exercise is to find a file hidden in a set of directories. The name of this file is **startingPoint.txt** and it's located in one of the sub-directories or the **/home/cdPractice** folder. You'll need to search through all of folders to find the file. You can either:

- Use the `cd` command to move to each directory and sub-directory under **/home/cdPractice**, and then use the `ls` command to display the files in the folder.

- Use the `ls` command on each folder (and sub-directory) under **/home/cdPractice** without changing folders. This requires specifying the path to the folder as an argument to the `ls` command.

Note that later in the class you'll learn easier ways to search for files. But for now, you should use this as an opportunity to practice with the `cd` and `ls` commands.

Write the full path to the directory that holds the file in the space provided:

____

2.8 This exercise is designed to help you become familiar with the main folders in the Linux/UNIX filesystem, and give you some perspective by making an association with the main files used by Microsoft Windows. Using the `cd` and `ls` commands, move around the Linux/UNIX file system and explore the various files and folders. See if you can find the equivalents to the following standard Windows folders. Note that this isn't super critical for you to know as a user. Just as it isn't crucial for a Windows user to know about all the directories on a Windows system. In either case, Windows or Linux, casual users just need to know enough to use the computer to do things like browse the Internet to watch funny cat videos, give away all their personal information on social media, or edit documents. But if you want to become a power user or system admin you'll need some better knowledge of the file system structure and where programs or system configuration files are stored. So, spend a little time on this exercise and see if you can get a general idea of how the Linux filesystem is organized, but don't worry about figuring out or trying to memorize what every single Linux directory holds.

You may have to do some Internet research or reading to figure out the Linux/UNIX names for these directories. On some Linux distributions you can start by typing `man hier`. This runs the `man` command, which is used to display the online documentation which is known as the manual pages. If available this will display the `hier` page, which many Linux systems use to explain the various folders/directories. Unfortunately, the `hier` man page is not available on every Linux distribution so you may receive an error message when you run the command. However, you can still find the same information by doing an Internet search for "**Linux hier**".

Fill in the Linux directory that is the equivalent to each of the following Windows directories:

Documents and Settings	__________________________
Program Files	__________________________
WINNT (or WINDOWS)	__________________________

2.9 This exercise provides experience determining which drives and devices are connected to a Linux/UNIX system Use the `df` command to see the partitions on this system. Create a screenshot showing the results. Don't worry about the output from the command, it probably won't make much sense at this point. It will become clear when you take the Linux Administration class.

And here's something to think about. Will every Linux/UNIX system have these same partitions? You don't have to answer this but give it some thought.

2.10 This set of exercises provides practice viewing the running processes with the `ps` command, which is similar to the information displayed by Windows Task Manager.

A. Running `ps` with no arguments shows you the processes that you are currently running. Type `ps` and create a screenshot or write down the results. (We will talk about what these processes are later in the quarter.)

B. To see all the processes being run by all users you must run `ps` with the `-A` option. Try running `ps -A` and inspect the results. Create a screenshot of the last screen.

Notice that you get a lot of information, more than will fit on one screen. When this happens, you can take the results from the `ps` command and hand them to another command called `more`. This is done by using the **|** character which is called a pipe in Linux/UNIX. For example, if type `ps -A | more` (You'll learn more about piping later.) The `more` command displays a screen of information, and then waits for you to tell it what to do. Hitting the spacebar tells more to display the next page.

C. Next try the `top` command by typing `top`. This is a little more like the Window's Task Manager. Create a screenshot of the output. Try and decipher what this command is showing you. Does it look like the process information for all users or just for you? When you're finished you can type **q** to quit the program.

2.11 This exercise will provide experience using commands that display information about your account, and other user accounts. Type the following commands and try and decipher the information displayed by each command. Create a screenshot showing the results after you run the last command

`whoami`	______________________
`who am i`	______________________
`w`	______________________
`who`	______________________
`who -u`	______________________
`finger`	______________________
`env`	______________________

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.

2.12 This exercise provides experience using the finger command to display information about other users with accounts on the system. Try using finger for some of the other students in the class. You can get their usernames by running the `w` or `who` commands.

You can also use the `-l` option for `finger` to see more information. Create a screenshot showing the finger information for at least one other user.

2.13 This exercise will provide you practice changing your finger information

A. Type the command `chfn`

B. After changing your finger information, verify that it has changed by typing `finger`

C. Create a screenshot showing your new finger information.

2.14 In the space provided list at least 5 commands that you can use while in the `man` utility (or `less`), along with a brief description of the behavior. You can find this information by typing `man less`, or by doing an Internet search for something like "Linux less command".

Key	Description
<space>________	_display the next screen's worth of information_______________
________________	____
________________	____
________________	____
________________	____
________________	____

2.15 This exercise provides experience using the `man` command. Type the following command, and create a screenshot showing the first page of the display:

```
man passwd
```

2.16 This exercise provides a demonstration of a stupid Linux trick. The following command uses incorrect syntax to get the man command to display a funny error message. There's nothing to turn in for this exercise, just type the command and enjoy.

```
man: why did you kick out your roommate?
```

2.17 This exercise provides experience using the `apropos` command.

A. Type the following command and create a screenshot: `apropos edit`

B. Type the following command and create a screenshot: `apropos delete`

2.18 This exercise provides experience using the `whatis` command.

Type the following commands and create screenshots showing the results:

`whatis apropos` ______________________________

`whatis cat` ______________________________

2.19 Getting help with `man, apropos and whatis`

This exercise provides practice deciding which Linux utility to use to get help. In the space provided, write down what you would type to successfully accomplish to each of the following tasks or scenarios. Of course, the first thing I would type would be google, so for this exercise assume that you're on a Linux computer that doesn't have Internet access and you must choose between `man`, `apropos`, or `whatis`. If you're not sure which command would be best, I suggest you login to your Linux account try them out. And keep in mind that there may be more than one command that will provide you with the information you're looking for.

A. You want to know which command to use to create a new directory

____

B. You want to know the options that can be used with the `ps` command

____

C. You want to know the options that can be used with the `ls` command

____

D. You want to know if the `passwd` command is used to change your password

____

E. You want to know if the `chfn` command is used to change your finger information

____

F. You want to know which command to use to send email

____

2.20 This exercise provides experience using commands that will provide information about the Linux distribution being used. Knowing this information isn't crucial if you're new to Linux, it just provides an idea of which distribution is being used. If you are an experienced user it can be important as there are minor differences between distributions.

Type the following commands and fill out the information below or create a screenshot. Note –this assumes the server is running a distribution based on RedHat.

```
cat /etc/redhat-release     ______________________________
uname –r                    ______________________________
```

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?

Files, Directories, and Permissions

The exercises in this chapter have been designed to reinforce your knowledge and provide hands on experience using the commands to create, copy, rename and delete files and folders, and use Linux/UNIX permissions so you can do the following:

1. Use the various commands for displaying or viewing the contents of a file.
2. Use FTP/scp to move files between a Linux/UNIX system and a PC or other computer.
3. Create, copy, rename and remove files and directories.
4. Read and change file and directory permissions.
5. Set and interpret the default permissions for new files and directories.

3.1 This set of exercises provides practice with the `cat` command to display files.

A. To get started practicing using `cat` to display the content of a file you will move to a directory that has a few files you can read. Use the `cd` command to change to the **/home/cdPractice/LOTR/hobbit** directory. Next use the `cat` command to display the file **A-names.txt** by typing:

```
cat A-names.txt
```

Create a screenshot showing the output from the command.

Next, display the file **B-names.txt** and create a screenshot showing the output from the command.

You can use the `cat` command to display the other files or stop when you figure out how the `cat` command works and start to get bored reading Hobbit names. You don't have to create screenshots for any of the other files.

B. You can also use `cat` to display files that are in a different directory by using relative or absolute paths. The syntax for using paths with `cat` is almost exactly the same as using paths with the `cd` and `ls` commands. The main difference is that with the `cat` command the path must end with a file name, not a directory/folder name.

For example, make sure you are back in your home directory by typing:

```
cd ~
```

then display the file **/home/cdPractice/LOTR/hobbit/A-names.txt** by typing:

```
cat /home/cdPractice/LOTR/hobbit/A-names.txt
```

For each of the following write the command you would use to display the designated file, assuming that you are in your home directory.

```
/home/cdPractice/LOTR/rings/ringNames.txt
```

____

```
/home/cdPractice/jokes/chuckNorris/cn.txt
```

____

```
/home/cdPractice/jokes/lawyer/neverTell
```

____

For each of the following write the command you would use to display the designated file, assuming that you are in your home directory. Note that this portion of the exercise is just a "thought exercise" designed to provide you practice in a theoretical situation. These files do not exist, and if you try and actually run the command to display the non-existant files you will get an error.

A file named **hazCheez** in the directory above the current directory

____

A file named **naptime** which is in a sub-directory of the current directory. The sub-directory name is **videos**.

____

C. The `cat` command has an option `-n` that can be used to display line numbers on each line. Some, but not all versions of `cat` also have a `-b` option only adds a number to non-blank lines. Use `cat -n` to display the file **/home/cdPractice/jokes/chuckNorris/cn.txt** with added line numbers by typing:

```
cat -n /home/cdPractice/jokes/chuckNorris/cn.txt
```

How many lines are in the file? ____________________________________

Try the `-b` option on **cn.txt**, and if it works use the output to answer the following question. If the line count is the same with the `-b` option as it is with the `-n` option it means the `-b` option is not available in the version of `cat` on this server.

How many non-blank lines are in the file? ______________________________

3.2 This exercise provides practice using the `more` command.

A. Use `more` to view the following files. Create a screenshot after more'ing the last file in the list.

- **/home/cdPractice/LOTR/rings/ringNames.txt**
- **/home/cdPractice/jokes/chuckNorris/cn.txt**
- **/home/cdPractice/jokes/lawyer/neverTell**

B. For files that contain more than a screens worth of information which command makes it easier to read the file contents, `cat` or `more`? (Circle your choice.)

3.3 This exercise provides practice using `more` and `less` to display the output from other commands.

A. Use the `ls` and `more` commands to display the files in the following directories, one screen at time. You can also use the `less` command instead of `more` if you prefer. Make a screenshot after displaying the last directory.

- **/home**
- **/usr/local**
- **/boot**

B. What would you type to see the contents of the following directories, one screen at a time?

/etc ______________________________

/usr/sbin ______________________________

/dev ______________________________

/usr/lib ______________________________

3.4 This exercise provides practice determining whether to use `cat`, `more`, or `less` to view a file, or to pipe the output from a command through `more` or `less`.

A. Assume you are in the folder **/home/cdPractice/jokes/lawyer** and you want to read the file **neverTell**. Which of the following would accomplish this?

```
a. ls neverTell
b. ls neverTell | more
c. more neverTell
```

B. Assume you are in your home directory and want to see a listing of the files in the **/etc** directory. Which of the following would accomplish this?

```
a. ls /etc | more
b. more /etc
```

C. Assume you are in the folder **/home/cdPractice/jokes/lawyer** and want to see a listing of the files in the **/usr/lib** directory. Which of the following would accomplish this?
   a. `ls /usr/lib | more`
   b. `more /usr/lib`

D. Assume you are in your home directory and want to read the file **/home/cdPractice/LOTR/rings/ringNames.txt**. Which of the following would accomplish this?
   a. `ls /home/cdPractice/LOTR/rings/ringNames.txt`
   b. `ls /home/cdPractice/LOTR/rings/ringNames.txt | more`
   c. `less /home/cdPractice/LOTR/rings/ringNames.txt`

E. Assume you are in your home directory and want to see a listing of the currently running processes. Which of the following would accomplish this?
   a. `ps -aux | more`
   b. `more ps -aux`

3.5 This exercise provides practice viewing Files with `head` and `tail`. Use the `head` and `tail` commands to accomplish the following tasks. Create a screenshot after running each command. (You don't actually need to know what's in these files. They're just being used to give you practice displaying files.)

A. Display the first 10 lines of the file **/etc/vimrc**

B. Display the last 10 lines of the file **/etc/vimrc**

C. Display the first 15 lines of the file **/etc/protocols**

D. Display the first 5 lines of the file **/etc/protocols**

3.6 Now apply what you've learned about `cat`, `more`, `less`, `head` and `tail`. In the space provided write what you type to accomplish the designated task, using the appropriate command. Note that there may be more than one correct way. Assume that you are in your home directory and **assume that all files exist**. Also note that this is just a "thought exercise" designed to provide you practice in a theoretical situation. These files and directories may not exist, and if you try and actually run the command to display the files it may result in an error.

1st 15 lines of **/etc/Muttrc** ________________________

Last 5 lines of **/etc/init.d/README** ________________________

The file **/usr/lib/rpm/macros**, one page at a time ________________________

Last 20 lines of the file in the current directory named **junk** __________________

A file named **fix.sh** located in the folder **myPrograms** which is a sub-directory of the current folder. You want to read this file one page at a time __________________

The contents of the **/bin** directory, one page at a time __________________

3.7 Treasure Hunt Part 2

In the last chapter, you went on a treasure hunt where you simply looked at file names. In this exercise you will go on a treasure hunt where you will need to not only look at file names, but also read the content of files. Each file in the treasure hunt will contain a portion of the information you'll need to determine the final answer as well as directions or clues to the next file.

To start, find file named **startingPoint.txt**. It's in one of the directories under **/home/cdPractice**. (This is the same file you found in the previous section of the Lab Manual.) Use `cat` or `more` or `less` to display the file. When the file is displayed the first line will contain a character or two that you will need to build the answer to this exercise. Ensure that you record the information from this file as you'll need it to build the final answer.

After you have found the file and written down the characters, use the instructions in the file as they will lead you to the next file in the chain of files. Finding the next file will require using the cd command to move the various directories, using the ls command to view the file names, and possibly using the `cat` or `more` or `less` commands to display the contents of files. Continue following the clues until you find the entire answer. When you have found the complete answer enter it in the space below.

____

3.8 The Detective

This exercise has been designed to provide you with additional practice changing directories and reading files. In this exercise you will pretend you're working as a system administrator at a college. One of the systems you're responsible for is a Linux system that's used by students for their classes. At the end of the quarter you're contacted by the campus security administrator who suspects that one of the students is involved in a series of phishing attacks. You're tasked with checking the student's home directory for clues that this user may have been involved in the crime. This requires looking at the files in the student's home directory to see if you can find any incriminating evidence.

The student’s username is **krimNL**, and their home directory is **/home/cdPractice/krimNL**. Use what you know about the `cd`, `cat`, `more` and `less` commands to read through the files and look for evidence. If you find any files that contain incriminating evidence list the file name below, and create a screenshot showing the evidence.

(Note - if you know anything about computer forensics you'll know that there are several steps in the forensic process that should/must be completed before jumping in and looking at the evidence. But this isn’t a forensics class, and the point of this exercise is to provide you practice changing directories and reading files, so you can simply start snooping around and not worry about search warrants or creating forensics disk images.)

List of files containing incriminating evidence:

____

____

____

____

____

3.9 This exercise provides practice using the `touch` command to create files. Use the `touch` command to create the following files. Note – you must be in your home directory to create these files. If you try to do this while you’re in most other directories the OS will prevent you from creating the files.

spiderman

tacos

covid

scuba

Make a screenshot showing the contents of your home directory after the files have been created. You can use the `ls` command to show the files.

3.10 This exercise provides practice using the `cat` command to create files and add content to the file.

A. Change to your home directory by typing `cd` or `cd ~`

B. Put the following text in a file named **hello** by typing `cat > hello` and then entering the following text. Note that **`<CTRL-d>`** means hit the **<CTRL>** and **d** keys at the same time.

```
echo Hello world
<CTRL-D>
```

C. Use the `cat` or `more` command to display the file **hello** on the screen and check your input. Create a screenshot showing the file's contents.

D. Put the following text in a file named **dog** by typing `cat > dog` and then entering the following text.

```
echo The cow says meow
echo The sheep says tweet tweet
echo The chicken says woof
<CTRL-D>
```

Use the `cat` or `more` command to display the file **dog** on the screen and check your input. Create a screenshot showing the file's contents. (Give yourself 50 extra bonus points if you knew I was going to combine cat and dog.)

E. Put the following text in a file named **truth.txt** by typing `cat > truth.txt` and then entering the following text:

```
echo Tony is the coolest guy I know
echo I believe everything he says,
echo I will give him all my money
echo UNIX rules !
<CTRL D>
```

F. Use the `cat` or `more` command to display the file **truth.txt** on the screen and check your input. Create a screenshot showing the file's contents.

3.11 This exercise provides practice deleting files using the `rm` command.

A. Create three files named **file1**, **file2** and **file3**. You can make empty files using the `touch` command or add text to the files using the `cat` command.

B. Do an `ls` to ensure that the files exist and create a screenshot.

C. Once you're sure the files exist delete them. Do another `ls` to ensure they are gone, and then create another screenshot.

3.12 This exercise provides you with experience trying to create files with special characters in the file name. For each of the following potential filenames use the `touch` command to try and create the file. If the file name doesn't contain any characters that have special meaning to the Linux shell the `touch` command will create the file. But if the file name has any problematic characters the `touch` command will return an error and the file won't be created. Note that this exercise only demonstrates that some characters have special meaning to the Linux shell, it won't flag characters such as – that may cause problems with some programming languages or other utility programs. Use the results from each test to mark the appropriate section in the table. Also mark whether the file would be hidden from the `ls` command, unless the `–a` option is used.

	File Created	Error Creating File	Hidden
watchFace.sh			
watch.face.007			
wazzup?			
mousepad#47			
TopSecret!!!			
tamale-recipe-67			
myPhone@home			
.to-do.list			
.alarm settings			
(shh quiet).mp3			
Bhodi's-beach			
$money			
&yet			

3.13 This exercise provides you with experience trying to create files with spaces in the file name.

A. Use the `touch` command to create the files with the following filenames.

Padishah Empire

Bela Tegeuse

Salusa Secundus

Hosnian Prime

B. Use the `ls` command to verify the files exist and that you only created 4 new files. If you created 8 files you did something wrong, and you should delete those 8 files, then try adding quotes around the file names. Create a screenshot showing your work.

C. Use the `rm` command to delete the files you created in Step A.

D. Use the `ls` command to verify the files have been deleted. Create a screenshot showing your work.

3.14 Since you don't always know what kind or type of data is in a file by looking at the name there's a command called `file` that will help. Type in the following to see a demonstration of the `file` command. See if you can decode what the `file` command is telling you about the various file types. Type the following commands and record what the type of file reported. Create a screenshot after running the last command.

```
file /etc/passwd
file /usr/bin/zforce
file /dev/tty0
file /usr/share/man/man1/kill.1.gz
file /home
```

Note that the point of this exercise is for you to realize that file extensions aren't mandatory in Linux/UNIX and they don't work the same way as they do in Windows. It's not critical for you to be able to decipher the information returned by the `file` command.

3.15 This exercise provides practice creating directories. Ensure you are in your home directory, and then use the `mkdir` command to create the following directories. When you are finished run the `ls` command to ensure that the directories exist, and make a screenshot.

bin

project1

project2

jokes

jokes/knock-knock

jokes/puns

3.16 This exercise provides practice deleting directories. Try `rmdir` on the directories you made in the previous exercise. If `rmdir` does not work, then try `rm -r`
When you are finished run the `ls` command to ensure that all of the directories have been deleted, and make a screenshot.

3.17 This exercise provides experience transferring files between a Windows computer and a Linux computer using scp, or in this case WinSCP.

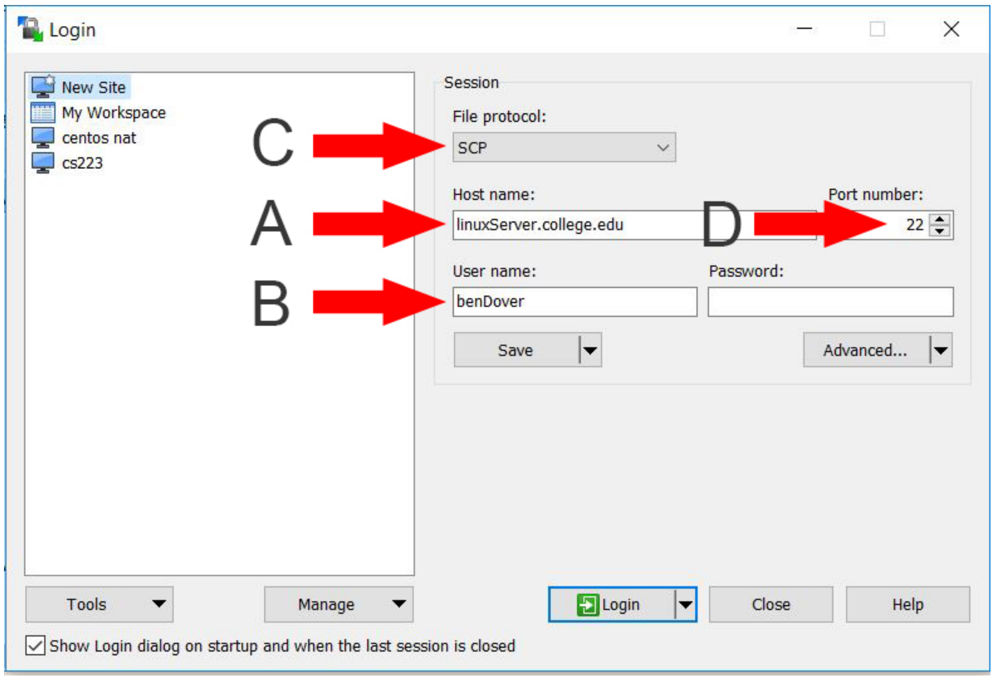
A. The first thing to do is download and install WinSCP. WinSCP is free and can be downloaded from many places on the Internet. But like PuTTY, many of the copies of WinSCP on the Internet have been infected with malware so I suggest you get a copy from the original developers at: **https://winscp.net/eng/download.php**. If this link is no longer valid, do an Internet search.

B. The other thing you'll need before starting is a test file to move back and forth. Use the word processor of your choice on the PC, NotePad, Word, whatever, and create a text file on your PC or home computer. Put the following text in the file.

   ```
   echo "Hello World"
   ```

   Make sure and save the file as plain text, with the name **hello**.

C. Start WinSCP and enter the **Host Name**, and your **User Name**. The **Host Name** must be set to **cs223Linux.columbiabasin.edu** The **User Name** must be set to ***your* user name**, the one that you use to log in to the system with PuTTY. You can optionally add your **Password**, but only do this on a PC that is NOT in a public place, like your home computer or personal laptop. If you add the **Password** in one of the computers in the lab it's possible for other users to pull up the settings for your session which will include your password. Ensure the **File protocol** option is set to **SCP** and the **Port number** is set to **22**.

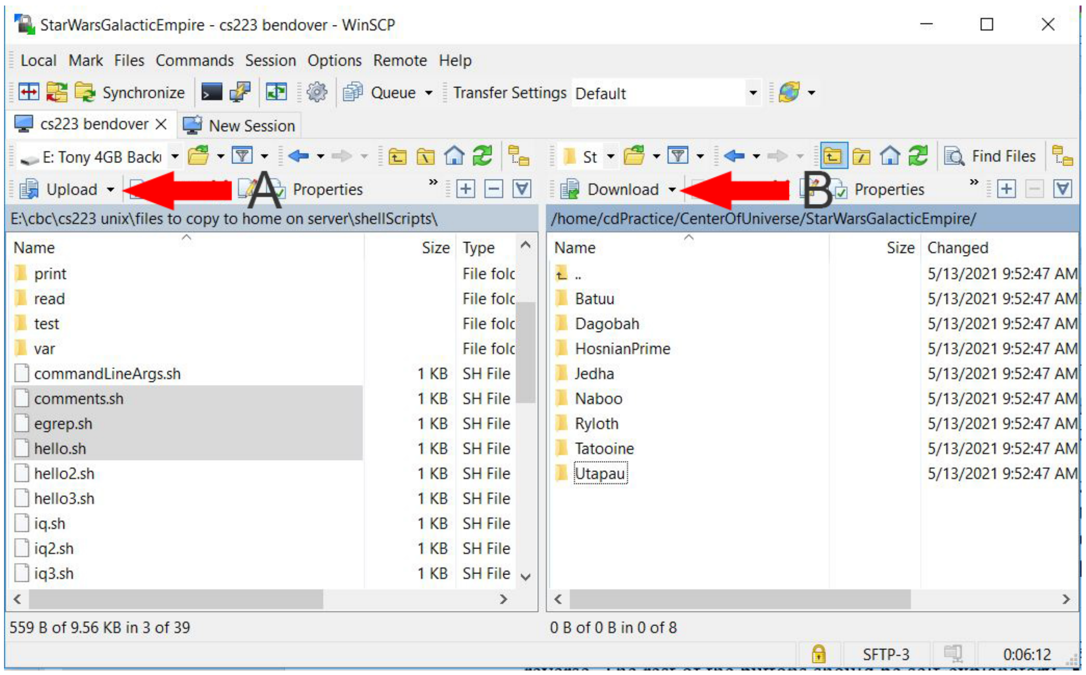


Session
File protocol: SCP
Host name: linuxServer.college.edu
Port number: 22
User name: benDover
Password:
☑ Show Login dialog on startup and when the last session is closed

Figure 3.1 Setting up a connection with WinSCP.

D. When you are finished, click the **Login** button, and WinSCP will connect you to the Linux computer. If you didn't already enter your password, you will be prompted to type it in. Once connected, you will see something similar to Figure 3.2 where the frame on the left of the WinSCP window represents the files on your PC, and the frame on the right shows the files in your home directory on the Linux server. You can navigate through the folders on both systems much like you do in Windows Explorer. That is, you move down into a folder by clicking the folder icon. To move up a folder click on the folder with the name **..** which will always be the top folder in the list.

E. You can move files from the PC to the Linux host by left-clicking on the PC file and then clicking the **Upload** button (**A**) of the window, or by simply double-clicking the file in the PC portion of the window. You can move multiple files by using the `<ctrl>` or `<shift>` keys to select multiple files, then clicking the right arrow.


StarWarsGalacticEmpire - cs223 bendover - WinSCP
Local Mark Files Commands Session Options Remote Help
Synchronize
Queue
Transfer Settings Default
cs223 bendover
New Session
E: Tony 4GB Backı
Upload
A
Properties
Find Files
Download
B
Properties

E:\cbc\cs223 unix\files to copy to home on server\shellScripts\

Name	Size	Type
print		File folc
read		File folc
test		File folc
var		File folc
commandLineArgs.sh	1 KB	SH File
comments.sh	1 KB	SH File
egrep.sh	1 KB	SH File
hello.sh	1 KB	SH File
hello2.sh	1 KB	SH File
hello3.sh	1 KB	SH File
iq.sh	1 KB	SH File
iq2.sh	1 KB	SH File
iq3.sh	1 KB	SH File

559 B of 9.56 KB in 3 of 39

/home/cdPractice/CenterOfUniverse/StarWarsGalacticEmpire/

Name	Size	Changed
..		5/13/2021 9:52:47 AM
Batuu		5/13/2021 9:52:47 AM
Dagobah		5/13/2021 9:52:47 AM
HosnianPrime		5/13/2021 9:52:47 AM
Jedha		5/13/2021 9:52:47 AM
Naboo		5/13/2021 9:52:47 AM
Ryloth		5/13/2021 9:52:47 AM
Tatooine		5/13/2021 9:52:47 AM
Utapau		5/13/2021 9:52:47 AM

0 B of 0 B in 0 of 8

SFTP-3 0:06:12

Figure 3.2 WinSCP showing files on the PC host computer and the Linux computer.

Moving files from the Linux host to the PC is done following the same process, but by clicking the **Download** button (**B**) instead. The rest of the buttons are fairly self-explanatory, or you can search the Internet for help with WinSCP.

F. Move the **hello** file you created on your PC to your Linux account. It may also have the name **hello.txt** if you saved it as plain text. You may have to navigate to the folder containing your **hello** file in a folder if it's not currently displayed in the left panel. Create a screenshot of the WinSCP window showing the file on your Linux account.

G. When you are done, click the **Exit** button in the lower right-hand corner. If you're working in a public place, make sure and quit or close the window so that others cannot access your account.

3.18 This set of exercises provide practice using the `cp` command to copy files.

A. To start this exercise ensure you're in your home directory using the following command:

```
cd ~
```

B. Create a file named **file1** and add some text to it by typing:

```
cat > file1
echo Letter B
<CTRL-D>
```

You can actually add whatever text you want to the file; just make sure it has something in it. Remember that `<CTRL-D>` means to hit the `<CTRL>` key and the `D` key at the same time, and that this will tell `cat` that you are done adding text to the file.

C. Verify that **file1** exists and has the text you added.

D. Make a copy of **file1** and name the destination **file2** by typing:

```
cp file1 file2
```

E. Verify that **file2** exists and has the same content as **file1**. Create a screenshot showing the content of **file2**.

F. Create two additional copies of **file1**, naming them **fileA** and **fileB**. Verify that **fileA** and **fileB** exist and have the same content as **file1**. Create a screenshot showing the content of **fileA** and **fileB**.

3.19 This set of exercises provide practice using the `cp` command to copy files.

In the space provided write what you would type to copy the specified file to a new file. The new file must have the specified name. Note that this is just a "thought exercise" designed to provide you practice in a theoretical situation. These files may not exist, and if you try and actually run the command to copy the files it may result in an error.

A.	File to copy: **tull.tab**	Copy to: **flute.txt**	___________________
B.	File to copy: **watch.c**	Copy to: **watch.c.bak**	___________________
C.	File to copy: **hdmi-list**	Copy to: **hdmiList7**	___________________

3.20 This set of exercises will provide you with experience copying files between directories.

A. To start this exercise ensure you're in your home directory using the following command:

```
cd ~
```

B. Verify that the file **/etc/yum.conf** exists by using `cat` or `more` or `less` to display the file.

C. Make a copy of **/etc/yum.conf**, and name the copy **yum.bak** by typing:

```
cp /etc/yum.conf yum.bak
```

D. Verify that **yum.bak** exists in your home directory and has the same content as **/etc/yum.conf**. Create a screenshot showing the content of **yum.bak**.

E. Move to the /etc directory by typing:

```
cd /etc
```

F. Copy the file **os-release** to your home directory by typing:

```
cp os-release /home/studentxx/os-info.txt
```

where *studentxx* is your user id.

G. Verify that the file **os-info.txt** exists in your home directory and has the same content as **/etc/os-release**. Create a screenshot showing the content of **os-info.txt**.

H. In this exercise you'll be in one directory, and copy a file from a second directory to a third directory. To start ensure you're in the **/etc** directory by typing:

```
cd /etc
```

I. Copy the file **/home/shellScripts/hello.sh** to your home directory by typing:

```
cp /home/shellScripts/hello.sh /home/studentxx/myHello.sh
```

where *studentxx* is your user id.

J. Verify that the file **myHello.sh** exists in your home directory and has the same content as **/home/shellScripts/hello.sh**. Use cat to display **myHello.sh** and create a screenshot.

3.21 This set of exercises provides additional experience copying files between directories.

A. Assume that you are in your home directory and you want to copy some files. In the space provided write what you would type to copy the file specified in **File to copy:** to a file in your home directory with name specified in the **Copy to:** column. Note that this is just a "thought exercise" designed to provide you practice in a theoretical situation. These files may not exist, and if you try and actually run the command to copy the files it may result in an error.

i. File to copy: **/var/log/cron** Copy to: **cron.log**

____

ii. File to copy: **/home/shellScripts/sqrt.sh** Copy to: **sqrt.sh.example**

____

iii. File to copy: **/home/ftpDemo/iq.sh** Copy to: **myIQdemo.sh**

____

B. Assume that you are in the directory **/media/USB** and you want to copy some files to your home directory. In the space provided write what you would type to copy the file specified in **File to copy:** to a file in your home directory with name specified in the **Copy to:** column. Your answer should have a just a filename for the source file, and specify the path to your home directory along with the filename in the destination. Note that this is just a "thought exercise" designed to provide you practice in a theoretical situation. These files may not exist, and if you try and actually run the command to copy the files it may result in an error.

i. File to copy: **/media/USB/resume**
Copy to: **resume**

____

ii. File to copy: **/media/USB/foodList.thur**
Copy to: **Thursday.meals.txt**

____

iii. File to copy: **/media/USB/songs/oingoBoingo.mp3**
Copy to: **privateLife.mp3**

____

C. Assume that you are in the directory **/home** and you want to copy some files to your home directory. In the space provided write what you would type to copy the file specified in **File to copy:** to a file in your home directory with name specified in the **Copy to:** column. Your answer should specify the path and filename for both the source destination, since they are both in different directories than the one you're currently in. Note that this is just a "thought exercise" designed to provide you practice in a theoretical situation. These files may not exist, and if you try and actually run the command to copy the files it may result in an error.

i. File to copy: **/tmp/CD-Collection**
Copy to: **forSale**

____

ii. File to copy: **/media/USB/calorieCounter.sh**
Copy to: **iAmHungry**

____

iii. File to copy: **/home/tsako/picture.jpg**
Copy to: **goober.jpg**

____

3.22 This exercise provides practice copying directories.

A. To start this exercise ensure you're in your home directory using the following command:

```
cd ~
```

B. Verify that the directory **/etc/udev** exists by using `ls` to display the directory's content.

C. Make a copy of **/etc/udev** and name the copy **myUdev** by typing:

```
cp -r /etc/udev myUdev
```

D. Verify that **myUdev** exists in your home directory and has the same content as /etc/udev by using the ls command. Create a screenshot showing the content of your **myUdev** directory. After you've made the screenshot use `rm -r myUdev` to delete the directory.

3.23 This exercise provides additional practice copying directories. Assume that you are in your home directory and you want to copy a directory. In the space provided write what you would type to copy the directory specified in **Directory to copy:** to your home directory with name specified in the **Copy to:** column. Note that this is just a "thought exercise" designed to provide you practice in a theoretical situation. These files may not exist, and if you try and actually run the command to copy the files it may result in an error.

A. Directory to copy: **/home/pathTest**
Copy to: **myPathTest**

____

B. Directory to copy: **/var/gopher**
Copy to: **gopherBackup**

____

C. Directory to copy: **/etc/pam.d**
Copy to: **pamStuff**

____

3.24 This exercise provides practice copying files and folders using . to represent the current directory.

A. Assume that you are in your home directory and you want to copy some files. In the space provided write what you would type to copy the file specified in **File to copy:** to a file in your home directory with name specified in the **Copy to:** column. Note that this is just a "thought exercise" designed to provide you practice in a theoretical situation. These files may not exist, and if you try and actually run the command to copy the files it may result in an error.

   i. File to copy: **/var/log/cron**
      Copy to: **cron**

      ____

   ii. File to copy: **/home/shellScripts/sqrt.sh**
      Copy to: **sqrt.sh**

      ____

   iii. File to copy: **/home/ftpDemo/iq.sh**
      Copy to: **iq.sh**

      ____

B. Assume that you are in your home directory and you want to copy a directory. In the space provided write what you would type to copy the directory specified in **Directory to copy:** to your home directory with name specified in the **Copy to:** column. Note that this is just a "thought exercise" designed to provide you practice in a theoretical situation. These files may not exist, and if you try and actually run the command to copy the files it may result in an error.

   i. Directory to copy: **/home/pathTest**
      Copy to: **pathTest**

      ____

   ii. Directory to copy: **/var/gopher**
      Copy to: **gopher**

      ____

iii. Directory to copy: **/etc/pam.d**
Copy to: **pam.d**

____

3.25 Copying Directories with Lab Manual Files. The /home directory contains several files that you will need to perform some of the upcoming Lab Manual exercises. There are several hundred files, so they have been organized into the following sub-directories:

cutExercises
grepsedPractice
numPerm
pathTest
quoteExercises
regexpressions
shellScripts
symPerm
viExercises
wildcardExercises

Since you will need these files to complete some of the later exercises, and you also need practice copying directories; this is the perfect time to copy these directories to your home directory. Use the `cp` command to make a copy of each of these directories in your home directory. Remember to use the `-r` option. When you are finished do an `ls` of your home directory, and create a screenshot showing that you successfully copied the Lab Manual practice files.

3.26 This exercise provides practice renaming files using the `mv` command.

A. To start this exercise ensure you're in your home directory using the following command:

```
cd ~
```

B. Create a file named **mvTest**, and add some text to it by typing:

```
cat > mvTest
echo It's Moving Day
<CTRL-D>
```

You can actually add whatever text you want to the file; just make sure it has something in it. Remember that `<CTRL-D>` means to hit the `<CTRL>` key and the `D` key at the same time and that this tells `cat` that you are done adding text to the file.

C. Verify that **mvTest** exists and has the text you added.

D. Move (or rename) **mvTest** by changing the name to **mvDay** by typing:

```
mv mvTest mvDay
```

E. Verify that the file **mvDay** exists in your home directory and has the same content as **mvTest** had. Create a screenshot showing the content of mvDay.

3.27 This exercise provides additional practice renaming files using the `mv` command. Assume that you are in your home directory and you want to move or rename a file. In the space provided write what you would type to move the file specified in **File to move:** to the name specified in the **Move to:** column. Note that this is just a "thought exercise" designed to provide you practice in a theoretical situation. These files may not exist, and if you try and actually run the command to move the files it may result in an error.

A. File to move: **margin.234.dat**
Move to: **marginData**

____

B. File to move: **codeGreen.cpp**
Move to: **codeGreen.orig.cpp**

____

C. File to move: **seagate.hub.T549**
Move to: **Seagate2TB**

____

3.28 This exercise provides practice using the `mv` command to move or rename directories.

A. To start this exercise ensure you're in your home directory using the following command:

```
cd ~
```

B. Create a directory named **wrongDir**.

```
mkdir wrongDir
```

C. Create a file named **file1** inside the **wrongDir** directory using the `touch` command.

```
touch wrongDir/file1
```

D. Verify that **wrongDir** exists and contains the file you added by using the `ls` command:

```
ls wrongDir
```

E. Move (or rename) **wrongDir** by changing the name to **rightDir** by typing:

```
mv wrongDir rightDir
```

F. Use `ls` to verify that the directory **rightDir** exists in your home directory and contains **file1**. Create a screenshot showing the directory listing for **rightDir**.

3.29 This exercise provides additional practice using the `mv` command to move or rename directories. Assume that you are in your home directory and you want to move or rename a directory. In the space provided write what you would type to move the directory specified in **Directory to move:** to the name specified in the **Move to:** column. Note that this is just a "thought exercise" designed to provide you practice in a theoretical situation. These directories may not exist, and if you try and actually run the command to move the directories it may result in an error.

A. Directory to move: **jokesAndPuns**
Move to: **funnyStuff**

____

B. Directory to move: **peanuts**
Move to: **pay.2020**

____

C. Directory to move: **manuals**
Move to: **autoManuals**

____

3.30 This exercise provides practice reading file permissions. Decode each of the following sets of permissions. Specify what the settings are for the owner, group and world

	Owner	Group	Others/World
`-rwxrw-r--`	__________	__________	__________
`-r-xr-xr-x`	__________	__________	__________
`-rwxr-xr-x`	__________	__________	__________
`-rw-r-----`	__________	__________	__________

3.31 This exercise provides practice interpreting the output from the `ls -al` command. The following table contains a portion of the detailed file listing for an imaginary directory. For each file listed complete the table by entering the following information. The first entry has been filled out as an example of what should be entered for each file.

- Specify whether the listing is for a file or a directory. If it's a file put an **f** in the **f/d** column of the table. If it's a directory enter a **d**.
- The permissions for the owner. Enter the rwx symbols for the owner in the **u** column of the table.
- The permissions for the group. Enter the rwx symbols for the group in the **g** column of the table.
- The permissions for "others". Enter the rwx symbols for others in the **o** column of the table.
- Who owns the file? Enter the owner's name in the **owner** column of the table.
- What group is associated with the file? Enter the group name in the **group** column of the table.

	f/d	u	g	o	owner	group
`-rw-r--r--. 1 root root`	f	rw-	r--	r--	root	root
`-rw-r--r--. 1 u2 u2`						
`drwxr-xr-x. 2 bako admin`						
`-rwxr-xr-x. 2 nuno students`						
`-rw-rw-r--. 1 alia students`						
`-rwxrw----. 1 bob mktg`						
`-rwxr--r--. 1 jose security`						

3.32 This exercise provides additional practice interpreting the output from the `ls -al` command. Answer the question using the following file details:

```
-rw-r--r--   1  orange   fruit  0 Jan 22 13:14 brass.mp3
```

A. Would the user **orange** be able to read the file? Yes / No

B. Would the user **orange** be able to edit the file? Yes / No

C. Would the user **orange** be able to delete the file? Yes / No

D. Would the user **orange** be able to run the file? Yes / No

E. Would members of the **fruit** group be able to read the file? Yes / No

F. Would members of the **fruit** group be able to edit the file? Yes / No

G. Would members of the **fruit** group be able to delete the file? Yes / No

H. Would members of the **fruit** group be able to run the file? Yes / No

I. Would users other than **orange** that are not members of the **fruit** group be able to read the file? Yes / No

J. Would users other than **orange** that are not members of the **fruit** group be able to edit the file? Yes / No

K. Would users other than **orange** that are not members of the **fruit** group be able to delete the file? Yes / No

L. Would users other than **orange** that are not members of the **fruit** group be able to run the file? Yes / No

3.33 This exercise provides additional practice interpreting the output from the `ls -al` command. Answer the question using the following file details:

```
-r-xr-xr--   1  sergey   mktg    0 Jan 22 13:14 poster12
```

A. Would the user **sergey** be able to read the file? Yes / No

B. Would the user **sergey** be able to edit the file? Yes / No

C. Would the user **sergey** be able to delete the file? Yes / No

D. Would the user **sergey** be able to run the file? Yes / No

E. Would members of the **mktg** group be able to read the file? Yes / No

F. Would members of the **mktg** group be able to edit the file? Yes / No

G. Would members of the **mktg** group be able to delete the file? Yes / No

H. Would members of the **mktg** group be able to run the file? Yes / No

I. Would users other than **sergey** that are not members of the **mktg** group be able to read the file? Yes / No

J. Would users other than **sergey** that are not members of the **mktg** group be able to edit the file? Yes / No

K. Would users other than **sergey** that are not members of the **mktg** group be able to delete the file? Yes / No

L. Would users other than **sergey** that are not members of the **mktg** group be able to run the file? Yes / No

3.34 This exercise provides additional practice interpreting the output from the `ls -al` command. Answer the question using the following file details:

```
-rwxrwxr--   1  sun   admin     0 Jan 22 13:14 poster12
```

A. Would the user **sun** be able to read the file? Yes / No

B. Would the user **sun** be able to edit the file? Yes / No

C. Would the user **sun** be able to delete the file? Yes / No

D. Would the user **sun** be able to run the file? Yes / No

E. Would members of the **admin** group be able to read the file? Yes / No

F. Would members of the **admin** group be able to edit the file? Yes / No

G. Would members of the **admin** group be able to delete the file? Yes / No

H. Would members of the **admin** group be able to run the file? Yes / No

I. Would users other than **sun** that are not members of the **admin** group be able to read the file? Yes / No

J. Would users other than **sun** that are not members of the **admin** group be able to edit the file? Yes / No

K. Would users other than **sun** that are not members of the **admin** group be able to delete the file? Yes / No

L. Would users other than **sun** that are not members of the **admin** group be able to run the file? Yes / No

3.35 This set of exercises provides additional practice interpreting the output from the `ls -al` command.

A. Change to the directory **/home/permissions** and run the `ls -al` command

B. List the owner of each of the following files:

beebs	
cwu	
ewu	
popcorn	
setup	
soapy.c	
toaster	

C. Assume that the user **benDover** is in the **n00b** group. Can **benDover** read the following files?

beebs? Yes / No

setup? Yes / No

soapy.c? Yes / No

toaster? Yes / No

D. Assume that the user **benDover** is in the **n00b** group. Can **benDover** edit or delete the following files?

beebs? Yes / No

setup? Yes / No

soapy.c? Yes / No

toaster? Yes / No

E. Assume that the user **benDover** is in the **n00b** group. Can **benDover** run or execute the following files?

beebs? Yes / No

setup? Yes / No

soapy.c? Yes / No

toaster? Yes / No

F. Assume that the user **carli** is in the **n00b** group. Can **carli** read the following files?

beebs? Yes / No

popcorn? Yes / No

setup? Yes / No

soapy.c? Yes / No

toaster? Yes / No

G. Assume that the user **maghar** is in the **n00b** group. Can **maghar** read the following files?

   beebs? Yes / No

   popcorn? Yes / No

   setup? Yes / No

   soapy.c? Yes / No

   toaster? Yes / No

3.36 This set of exercises is designed to provide practice changing file permissions symbolically.

A. To start this exercise ensure you're in your home directory using the following command:

```
cd ~
```

B. Create a file named **perms**.

```
touch perms
```

C. Check the permissions on the file by using the `ls -al` command.

```
ls -al perms
```

The permissions will most likely be `-rw-rw-r--`. Note that the default permissions, those you see, may vary from this slightly. You'll learn how to control your default permissions later in this section.

D. Change the permissions by adding the execute permission for the file owner by typing:

```
chmod u+x perms
```

E. Change the permissions by removing the write permission for the group by typing:

```
chmod g-w perms
```

F. Use the `ls -al` command to verify that the permissions have been set correctly and create a screenshot.

3.37 Changing permissions symbolically. This set of exercises is designed to provide practice changing file permissions symbolically. In this exercise you will copy a folder containing several files and then change permissions on the files using symbols.

A. To start this exercise ensure you're in your home directory. Copy the folder **/home/symPerm** to your home directory using the command:

```
cp –r /home/symPerm ~
```

Move into your copy of the **symPerm** directory using the command:

```
cd symPerm
```

B. Change the permissions on the following files. Note that when multiple changes are required you can try and do it with one `chmod` command, or run `chmod` multiple times. When you are finished with all the changes in steps i – v , run the `ls –al` command to display the permissions, and create a screenshot.

i. File: **always.mp3**
Changes: remove other read, and add group write

ii. File: **cheesecake.recipe**
Changes: add user execute, add group execute, and add other execute

iii. File: **contacts**
Change: remove group read, remove other read

iv. File: **fudd.sings.mp3**
Change: add user execute, add group write, add group execute, remove other read

v. File: **mower.cpp**
Change: add user execute, add group execute, add other execute

3.38 This set of exercises is designed to provide practice changing file permissions symbolically. Assume that you want to change permissions on the following files. In the space provided write what you would type to add or remove the specified permissions. Note that this is just a "thought exercise" designed to provide you practice in a theoretical situation. These files may not exist, and if you try and actually run the command to change the permissions it may result in an error.

i. File: **jokesAndPuns**
Changes: add user write, remove other read

____

ii. File: **peanuts**
Changes: add user execute, add group write, add other read

____

iii. File: **manuals**
Changes: add group read, add other read

____

iv. File: **protein**
Change: add owner execute, add group execute, add other read, add other execute

____

3.39 This exercise provides practice converting numbers from binary to decimal. In the space provided write the decimal equivalent for each of the following binary numbers. The first answer has been provided as an example.

010_2	-w-________
100_2	___________
101_2	___________
001_2	___________
000_2	___________
111_2	___________
011_2	___________

3.40 This exercise provides practice converting sets of permissions to binary numbers. In the space provided write the binary number that represents the given set of permissions. The first answer has been provided as an example.

- w -	010________
r - -	___________
r – x	___________
- - x	___________
- - -	___________
r w x	___________
- w x	___________

3.41 This exercise provides practice calculating decimal numbers from a desired set of permissions. In the space provided write the decimal number that represents the given set of permissions. The first answer has been provided as an example.

- w -	2__________
r – x	___________
- w x	___________
- - x	___________
- - -	___________
r - -	___________
r w x	___________

3.42 This exercise provides practice calculating the three numbers required to set permissions with the `chmod` command. Write down the number you would provide `chmod` to set the following sets of permissions.

	Owner	Group	Others/World
`-rwxrwxr-x`	___________	___________	__________
`-rwxr-xr-x`	___________	___________	__________
`-r-xr-xr--`	___________	___________	__________
`-rw-r--r--`	___________	___________	__________

3.43 Changing permissions numerically. This set of exercises continues to provide practice changing file permissions using numbers. In this exercise you will copy a folder containing several files and then change permissions on the files using numbers.

A. To start this exercise ensure you're in your home directory. Copy the folder **/home/numPerm** to your home directory using the command:

```
cp -r /home/numPerm ~
```

Move into your copy of the numPerm directory using the command:

```
cd numPerm
```

B. Change the permissions on the following files so that they end up with the specified set of permissions. When you are finished with all the changes in steps i – v , run the `ls -al` command to display the permissions, and create a screenshot.

i. File: **always.mp3**
Change to: -rw-rw----

ii. File: **cheesecake.recipe**
Change to: -rwxr-xr-x

iii. File: **contacts**
Change to: -rw-r-----

iv. File: **fudd.sings.mp3**
Change to: -rwxrw----

v. File: **mower.cpp**
Change to: -rwxr-xr-x

3.44 For each of the following, enter what you type to use numbers to change the permissions on the specified file(s). For example, if you are directed to set the permissions on the file **yoda12** so that the owner had all permissions, the group had read permissions and the rest of the world had no permissions you would enter `chmod 740 yoda12`**.** Note that this is just a "thought exercise" designed to provide you practice in a theoretical situation. These files may not exist, and if you try and actually run the command to change the permissions it may result in an error.

A. Set the permissions on `watchList` so the owner has read and write permissions, the group and world have read permission.

____

B. Set the permissions on `answers` so the owner has full permissions, the group has read and write permission, the world has read permission

____

C. Set the permissions on `decrypt` so the owner has read and write permissions, the group has read and write permission, and the world has no permissions.

____

D. Set the permissions on `kasper` so the owner and group have full permissions, and the world has no permissions

____

3.45 Testing Read and Write File Permissions. This exercise was designed to demonstrate how the read and write permissions function. In this exercise you will change the read and write permissions on a file, and then try to read or delete the file. The ability to write, edit and delete a file are all controlled by the write permission. And at this point, since you haven't learned how to use an editor, it will be easier to test the write permission by deleting files than by trying to write to a file.

A. To start this exercise ensure you're in your home directory using the following command:

```
cd ~
```

B. Create a file named **rwTest** and add some content to it by typing:

```
cat > rwTest
What do you call 8 hobbits?
A hobbyte
<ctrl-d>
```

You can add any text you want, just make sure that you have something in the file that you can read.

C. Check the permissions on the file by using the `ls -al` command. Remove all permissions for the owner (in this case you) by typing:

```
chmod 000 rwTest
```

D. Test the read permission by trying to read the file using the one of the following commands: `cat, more, less`. For example:

```
cat rwTest
```

Were you able to read the file or did you receive an error? If you received an error did the error message make sense?

E. Give yourself read permission by typing:

```
chmod 400 rwTest
```

F. Test the read permission by trying to read the file using the one of the following commands: `cat, more, less`. For example:

```
cat rwTest
```

Were you able to read the file or did you receive an error?

G. Do another `ls -al` to verify the permissions on the file, and to verify that you do not have write permission. Now test the write permission by trying to delete the file by typing:

```
rm rwTest
```

Were you able to delete the file or did you receive an error? If you received an error did the error message make sense?

H. Give yourself write permission by typing:

```
chmod 600 rwTest
```

Note – you could also use a permission of 200 to accomplish this.

I. Do another `ls -al` to verify the permissions on the file, and to verify that you have write permission. Test the write permission by trying to delete the file by typing:

```
rm rwTest
```

Were you able to delete the file or did you receive an error? If you received an error did the error message make sense? Create a screenshot showing your work to this point.

3.46 This exercise provides experience creating and executing a shell script.

A. Ensure that you are in your home directory by typing: `cd ~`

B. Create a file named **hello.sh** by typing the following:

```
cat > hello.sh
echo "hello world"
echo "the current date and time is: "
date
<ctrl-d>
```

C. Try and run the file by typing:

```
./hello.sh
```

This will result in an error message saying "Permission Denied". This is a pretty straightforward error message, telling you that you don't have execute permission.

D. Give yourself execute permission. I shouldn't have to tell you how to give yourself execute permission, but just in case you do this by typing: `chmod 700 hello.sh`

E. Try and execute the file by typing: `./hello.sh`

This time it should run successfully and you should see something similar to the following, although your date will certainly be different.

```
hello world
the current date and time is:
Fri Mar 14 15:05:27 PDT 2020
```

F. Create a screenshot showing the results.

3.47 This exercise provides a way to test the file and directory permissions. You will create a directory, change the permissions; and then try to do several things like copy files into the directory, list the directory contents or execute a script that resides in the directory. You will then change the directory permissions and repeat the testing steps, until you've tried all 8 possible settings for the directory permissions.

Before you start, let me warn you that this is a little complicated and there are a lot of steps to the testing. If you just type commands without knowing what you're looking for the results probably won't make much sense and you won't get a lot out of the exercise. So, before you test each directory setting take a few seconds and think about what you think you should be able to do with the directory permission setting before you start running the tests.

Using the `mkdir` command, create a subdirectory in your home directory named **ptest**. For this exercise, do the `chmod` command using the numbers indicated, try and run the commands, and write down the results or error messages. Think carefully about what you are doing, for example, if you cannot copy the file `hello.sh` into the directory ptest, or `cd` into **ptest**, you may get results anyway!

a. `cd ~` [Ensure you're in your home directory.]

b. `mkdir ptest` [Make a new subdirectory to use for testing.]

c. `chmod 000 ptest` [Set the permissions on the test directory to 000.]

d. `cp hello.sh  ptest/hcopy.sh` [Try and copy the shell script you made above into the test directory. The directory permissions will control whether or not this will be successful or cause an error.]

e. `ls ptest` [Try and list the contents of the test directory. The directory permissions will control whether or not this will be successful or cause an error.]]

f. `cat ptest/hcopy.sh` [Try and view the contents of the file you copied into the test directory. This won't work if you were not able to copy the file using the `cp` command in step d or if the directory permissions prevent you from reading files.]

g. `ptest/hcopy.sh` [Try and run the shell script. The directory permissions will control whether or not this will be successful or cause an error.]

h. `cd ptest` [Attempt to move into your test directory. The directory permissions will control whether or not this will be successful or cause an error.]

i. `ls` [List the directory contents. If the `cd` in step h. above failed, you will see your home directory. If you are still in your home directory try `ls ptest` instead. The directory permissions will control whether or not this will be successful or cause an error.]

j. `cd ~` [Get back to your home directory. This command is superfluous if step h. failed and you never left.]

k. `rm ptest/hcopy.sh` [Get rid of hcopy.sh. If you were not able to make the copy of **hcopy.sh** to start with in step d., this won't work.]

l. Create a screenshot showing the results of all your tests.

m. Repeat step c. through step l., but this time change the permissions on the ptest directory to 100. In other words, run the following commands:

```
chmod 100 ptest
cp hello.sh ptest/hcopy.sh
ls ptest
cat ptest/hcopy.sh
ptest/hcopy.sh
rm ptest/hcopy.sh
cd ptest
ls
cd
```

n. Keep repeating the set of instructions in step m., but in subsequent tests use 200, 300, 400, 500, 600, and 700 for the permissions on **ptest**. This will allow you to see the results of the tests for every possible combination of permissions. Create a screenshot each time after the last `cd` command in the group.

3.48 What is your current umask value?

____

3.49 What are your default file permissions? There are two ways to figure this out. The first is to type the `umask` command, and then subtract the number displayed from 666. The second is to create a new file and then check what the permissions are. You just checked the umask value in the previous exercise.

A. Use the `touch` command to create a new file then use the `ls -al` command to check the permissions on the new file. Record the permissions on the new file in the space below.

____

B. Next, convert the permissions on the new file to their numeric value.

____

C. Finally, subtract the number in Step B from 666. Record this value in the space below.

____

D. Does the value from Step C match the output from the `umask` command? Yes / No

3.50 What are your default directory permissions? Once again there are two ways to figure this out. The first is to type the `umask` command, and then subtract the number displayed from 777. The second is to create a new directory and then check what the permissions are.

A. Use the `mkdir` command to create a new directory then use the `ls -al` command to check the permissions on the new directory. Record the permissions on the new file in the space below.

____

B. Next, convert the permissions on the new directory to their numeric value.

____

C. Finally, subtract the number in Step B from 777. Record this value in the space below.

____

D. Does the value from Step C match the output from the `umask` command? Yes / No

3.51 For each of the following numbers, write down what default permissions would result.

	files	directories
`umask 066`	____________	____________
`umask 222`	____________	____________
`umask 024`	____________	____________

3.52 Write down the number you would provide `umask` if you wanted to have the following sets of default file permissions.

`-rw-rw-r--`	____________
`-rw-r--r--`	____________
`-r--r-----`	____________

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. What command(s) would you use to view the file /etc/inetd.conf ?
2. How much of a file does `cat` display? That is, what happens if you use cat to display a file that has more lines than can be displayed in the current window?
3. How is the `more` command different than the `cat` command?
4. What is the command to view the first 10 lines of the file /etc/passwd?
5. If you use the `head` command how many lines will you get if you do not specify a number? That is, what is the default?
6. What is the command to view the last 10 lines of the file /etc/passwd?
7. What command would you use to quickly create a new blank file?
8. True or False. Linux/UNIX files use and require all file names to have the same three letter file extensions as Windows file names.
9. What command is used to create directories?
10. How are Linux/UNIX directories different than Windows folders?
11. What command(s) can be used to remove directories?
12. True or False. The `rmdir` command will remove a directory and all its sub-directories.
13. What command and option you would use to remove files, but have the utility ask you before deleting the file.
14. What command and option you would use to remove a directory along with any of its subdirectories?
15. What do you have to do to tell the `cp` command to copy something to the current directory and have the destination have the same name as the source?
16. What command is used to rename files?

17. What command and option(s) are used to copy a directory and all its subdirectories?

18. How do you see a display of file permissions or directory permissions?

19. What are the three letters used to indicate the possible permissions for the file's owner, and what do they stand for?

20. Besides the file's owner, what two other groups can permissions be assigned to?

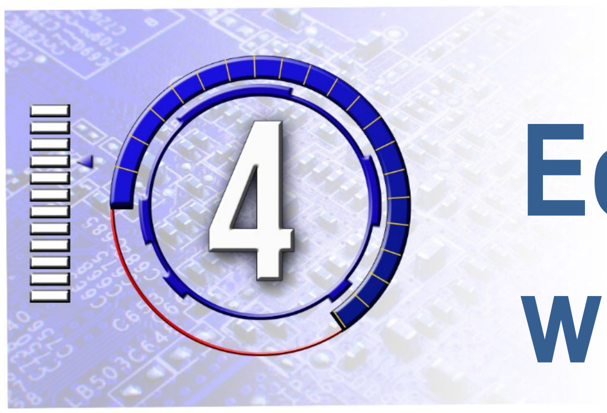
21. What are the two possible methods used by `chmod` for changing file and directory permissions?

22. Which method of changing permissions is more efficient for changing multiple permissions?

23. Would you use `chmod` or `umask` to change permissions on a file that already exists?

24. True or False. It's impossible to set your default file permissions so that you have execute permission.

25. If it's impossible to change your default permissions so that you have execute permission, how do you ever execute a file?



Editing Files with vi

The exercises in this chapter have been designed to reinforce your knowledge and provide hands on experience using the vi editor so you can do the following:

1. Start vi, perform basic editing tasks, and save files.
2. Switch between Input Mode and Command Mode in vi.
3. Perform advanced editing tasks using Line Commands

COPYING DATA FILES FOR THIS SECTION OF EXERCISES

The files from **/home/viExercises** must be copied your home directory to complete the exercises in this chapter. You may have already copied these files in one of the exercises in the previous section, but if you haven't you should copy them before starting the exercises. Remember that you will need to use the `-r` option with the `cp` command to copy a directory. For example:

```
  cp -r  /home/viExercises  ~
```

The exercises in this chapter have been designed to provide you practice learning vi, but it isn't always the best resource if you need to look something up quickly, or if you want to become an expert in vi. There's always a tradeoff between materials meant for learning a subject versus quick reference materials. The learning material contains longer explanations and background material that are helpful when you're first learning a new skill, but all this extra material can be difficult to use if you need to look up something up quickly. The reference materials assume you already know the basics and just need a quick way to look up commands and options without all of the extra things like exercises to wade through.

If you want or need a vi reference guide, one that allows you to quickly look up commands, or a complete reference there's one at the end of the vi chapter in the book, or you can always use the Internet. Or you can read the tutorial available on some Linux distributions which is accessed by typing: `vimtutor`

4.1 The best way to learn about `vi` is to use it. This section gives step by step instructions for creating a file. In each step you see an action to perform and then the necessary keystrokes. Don't be concerned with complete accuracy, instead try and concentrate on learning the concepts of using `vi` to create a file, how to move between command and input modes, and how to save your edited files. If you run into difficulties, you can quit and start over by pressing `<esc>` to make sure you're in Command Mode; then typing : **q!** and pressing `<enter>` to quit.

A. Make sure you are in your home directory, or one of your own sub directories; so that you have permission to create a file.

B. Start `vi` by typing **vi** and pressing `<enter>`.

C. Go into Insert mode to place characters on the first line by pressing the **a** key (don't press `<enter>`) . This is the command to append characters to the first line, so you will not see the character "**a**" on-screen.

D. Add lines of text to the buffer. Type:

   Things to do today.
   a. Practice vi.
   b. Buy Tony pizza for lunch.
   c. Switch to AT&T long distance to thank them for inventing UNIX.
   d. Spend more hours practicing vi.
   e. Read old Dilbert cartoons about UNIX, finally "get it".

   You can use the `<Backspace>` key to correct mistakes on the line you're typing. Don't worry about being precise here: this example is for practice. You will learn other ways to make changes in some of the later exercises.

E. Switch from Insert mode back to Command Mode, by pressing the `<esc>` key. You can press `<esc>` more than once without worry; if you're already in Command Mode nothing happens although you may hear a beep.

F. Save the text you entered in a file called **vipract.1** by typing **:**`w vipract.1` and press `<enter>`. The characters **:w vipract.1** appear on the bottom line of the screen (the *status line).* (The characters should not appear in the text on your screen. If they do, you're still in Insert Mode.) The **:w** command tells vi to write the text in the buffer to the specified file. In this case, this command *saves or writes* the buffer to the file **vipract.1**. When it's done writing the file vi will confirm the action on the status line. You should see something similar to the following on the status line:

```
"vipract.1" [New File] 5 lines, 136 characters
```

   This statement confirms that the file **vipract.1** has been created, that its a new file, and that it contains 5 lines and 136 characters. Your character count may be different if you didn't type the information exactly as specified (spaces count as characters). Create a screenshot showing your work to this point.

G. Exit vi by typing **:q** and press `<enter>`.

   Note you should still be in Command Mode before and when you type :**q**. If this is successful the :q will be displayed on the status line, but as soon as you press `<enter>` vi will terminate and you will return to the shell prompt. If you're in Insert Mode when you type **:q** you'll see the characters added to the current text. If this happens make sure and hit the `<esc>` key before typing **:q**.

4.2 Editing a file that already exists. This is done by typing vi followed by the name of the file you want to edit and pressing `<enter>`. Just make sure the file you want to edit is in the folder you're currently in. If not, either move to the folder or give vi the path to the file.

A. Try this on the file you created in the previous exercise by typing: `vi vipract.1`

B. Move to the bottom of the file and add the following text.

Things to do tomorrow.
a. Practice vi some more.
b. Buy Tony Chinese food for lunch.
c. Call Al Gore and thank him for inventing the Internet

C. When you are done save the file and then exit. Use `cat` or `more` to display the file and create a screenshot.

4.3 This exercise demonstrates some of the ways that can be used to change to Insert Mode from Command Mode in vi.

A. Open the file **insertTest**. If you start vi and see an empty file it means one of two things, you haven't copied the folder containing the file insertTest, or you copied the folder but you're in a different directory. To fix these problems copy the folder **/home/viExercises** to your home directory using the command:

```
cp -r /home/viExercises ~
```

Then use the `cd` command to move into the directory.

B. Position the cursor somewhere near the center of the text buffer.

C. Type one of the characters shown in the left column table below. These commands will cause you to enter Insert Mode at which point anything you type will be added to the buffer. You will remain in Insert Mode until you hit the `<esc>` key, which will take you back to Command Mode.

D. For each character in the left column, pay attention to where the cursor is before you hit the key, and where the cursor ends up after you hit the key. Complete the table by adding a brief description of what happens when you hit each key.

`a`	
`i`	
`A`	
`I`	
`o`	
`O`	

4.4 This exercise demonstrates various methods for moving the cursor in vi. Open the file **moveTest**. Position your cursor somewhere near the center of the file, then type one of the preceding characters. In the space below, write down how each of these keys affects the position of the cursor.

`h`	One character left
`j`	One line down
`k`	One line up
`l`	One character right
`e`	
`w`	
`3w`	
`b`	
`3b`	
`0`	
`$`	
`+`	
`-`	
`)`	
`(`	
`G`	
`gg`	
`<ctrl+f>`	Move down (forward) one screenful of lines.
`<ctrl+b>`	Move up (backward) one screenful of lines.
`<ctrl+d>`	Move down 1/2 screenful of lines.
`<ctrl+u>`	Move up 1/2 screenful of lines.
`M`	
`H`	
`L`	

4.5 This exercise provides practice deleting text from Command Mode in vi.

Open the file **deleteTest** and position the cursor somewhere near the center of the document. Ensure that you are in Command Mode and then type the following character(s). In the space below, describe what happens when you hit the corresponding key(s):

x	
X	
D	

4.6 This exercise provides practice deleting blocks or sections of text in vi.

Open the file deleteTest and try using the V command to highlight and delete a block of text. Does the version of vi in this distribution support this operation?

Yes	
No	

4.7 This exercise provides practice using the **u** command in vi to undo the last edit.

A. Open the file **undoTest**

B. Enter your name and age in the first two lines where indicated. Delete the prompts that says (put your name here) and (put your age here). Create a screenshot showing the changes.

C. Use the undo command to undo all the deletes. Create a second screenshot showing the restored text.

4.8 This exercise provides practice using vi to change and replace text.

A. Open the file **changeTest** and make the following changes:

- In line 1 change the word "**barn**" to "**brain**"
- In line 2 change "**ate three**" to "**lately**"
- In line 2 change "**feel**" to "**seem**"
- In line 3 change "**bunny**" to "**funny**"
- In line 4 change "**y**" to "**why**"
- In line 5 change "**this guy**" to "**the sky**"

B. When you are finished making the changes, create a screenshot showing the corrected text.

4.9 Practice using basic editor commands. Once again, the best way to learn vi is to use it yourself. Use what you've learned so far to create the following documents. Remember the point of these exercises is to provide you practice; no one is going to check the information you put in the documents to make sure it's all completely accurate.

A. Use `vi` to create a simple resume. Name the file **myResume**. Create a screenshot of the document you're finished.

B. Use `vi` to create a copy of your schedule for this quarter. Name the file **mySchedule.** At a minimum include the class number and time. Use tabs to make the columns line up. Create a screenshot of the document you're finished.

C. Use `vi` to create a "cheat sheet" of `vi` commands. Name the file **myReference.** Create a screenshot showing the finished document. At a minimum include the following. And for you type A personalities, keep it to 20 commands or less. The point of this exercise is just to provide you practice using `vi`, it's not actually to build a complete reference.

- Two commands for switching from Command Mode to Insert Mode
- The key to hit to return from Insert Mode back to Command Mode
- Two commands for replacing text
- Two commands for saving and or exiting the editor

4.10 This exercise provides practice using the `vi` commands to copy and paste lines of text. Practice copy/cutting and pasting by opening the file **copyAndPaste** and doing the following:

A. Copy the top 10 lines and paste them at the end of the file
B. Cut the lines 11-20 and paste them at the end of the file
C. Create a screenshot showing the last lines of the file.

4.11 Write down the command to accomplish each of the following actions. You can assume you're in Command Mode.

A. Copy the text from the current position to the end of the line

____

B. Cut the text from the current position to the end of the line

____

C. Copy the current line and the next 7 lines

____

D. Cut the current line and the next 7 lines

   ____

E. Paste any copied text below the current line

   ____

4.12 This exercise provides practice searching for text in `vi`. Open the file **searchTest** and get familiar with the content. Note that all the lines are numbered.

A. Find the first occurrence of the string **Tony** using

```
/Tony
```

Create a screenshot.

B. Find the next occurrence by typing the following.

```
/
```

Create a screenshot.

C. Use the arrows keys (or the method of your choice) to move to the last line in the file. Find the closest previous occurrence of the string **Tony** by typing:

```
?
```

Create a screenshot.

D. Use the arrows keys (or the method of your choice) to move up to the top of the file. Highlight all occurrences of the string Tony by typing:

```
/tony/p
```

Create a screenshot.

4.13 This exercise will provide practice using the search and replace command.

A. Open the file **replace1**.

B. Ensure that you're on line 1 and in Command Mode. Type the following command:

```
:s/mac/windows
```

Notice how much of the line changes.

C. Change the remaining occurrences of **mac** to **windows** by using the following:

```
:s/mac/windows/g
```

D. Change all occurrences of **`fender`** to **`gibson`**. Ensure that all the strings in the file are changed.

E. Change all occurrences of **`garmin`** to **`suunto`**. Ensure that all the strings in the file are changed.

F. Move to the last line in the file. Change the occurrence of **`rein`** to **`rain`**. Hint 1 – if you don't get what you expect you can use the `u` command to undo the last change. Hint 2 – to ensure the command only finds the string **rein**, add a space on either side of **rein**; but also make sure you add the spaces back in with **rain**.

G. Move to line 9. Change the occurrence of **`invent`** to **`invention`**.

H. Create a screenshot showing the file with all your replacements.

4.14 This exercise provides practice specifying line number ranges in vi. For each of the following ranges, write down the lines that would be affected by search and replace command:

`. , 10`	
`1 , $`	
`. , $`	
`-5 , $`	
`1 , +5`	
`/tony/ , $`	

4.15 For each of the following, write down the appropriate line number ranges:

	the entire file
	10 lines before current line to 3 lines after
	start of file to current line
	current line to end of file
	next line with the word "bond" to the end of the file
	start of file to 6 lines past current line

4.16 Test running an external command inside vi by doing the following:

A. Open the file **exCommands**

B. Ensure that you're in Command Mode

C. Inspect the file to see the order of the lines

D. Type `:%!sort`

E. Inspect the file to see the order of the lines. Has the order changed?

F. Type `:%!cat -n` (The `-n` option tells cat to add line numbers to the each line being sent to stdout.)

G. Inspect the file. Has a line number been added to each line? Create a screenshot showing the file after it has been sorted and the line numbers added.

4.17 This exercise provides practice using the `:set` command.

A. Open the file **setNumbers**

B. Ensure you’re in Command Mode

C. Turn on line numbers using the `:set number` command

D. Determine which line contains the text **cheesecake.** That is what is the number of the line? It’s the last line in the file, you need to figure out what the line number is. This will be easy to determine if you have the line numbers turned on, more difficult and leave a lot of fingerprints on your screen if you don’t.

cheesecake line number: ___________________________

4.18 What version of vim/vi is being on your system? To determine this either use the `cd` command to move the `/usr/share/vim` folder and do an `ls`, or use the command `ls /usr/share/vim`

vim/vi version number: ____________________________

4.19 Start `vi` and change the colorscheme. Use **darkblue** if it is available on your system. If it's not available choose any of the available colorschemes. Create a screenshot showing `vi` with the new colorscheme.

4.20 This exercise provides practice saving your `vi` configuration settings.

A. Ensure that you are in your home directory

B. Edit the file **.vimrc** (It most likely does not exist, so you will have to create a copy.)

C. Add the set command to display line numbers, and change your colorscheme

D. Save the file

E. Start `vi` and verify that the set commands were executed, and the color scheme was changed.

F. Create a screenshot showing the contents of your **.vimrc** file.

Note – if you don't like the new colorscheme, delete it out of your **.vimrc** file after completing this exercise.

Metacharacters – Wildcards, Pipes, and Quotes

The exercises in this chapter have been designed to reinforce your knowledge of metacharacters and provide hands on experience using metacharactes with Linux/UNIX commands so you can do the following:

1. Use globbing or wildcards to match multiple file names.
2. Display shell variables.
3. Print tabs and newlines using the `echo` command.
4. Use the various types of quotes and whacks to protect metacharacters from taking on special meaning.
5. Describe stdin and stdout, and identify them for any command or set of commands.
6. Explain `noclobber`, and how to set or unset it.
7. Type multiple commands on the same line.
8. Use redirects and pipes.
9. Use the `cat`, `head`, `tail`, `grep`, and `cut` commands to start a command pipeline.
10. Use the `cut`, `paste` and `join` commands to manipulate data by columns.
11. Use the `sort`, `sed`, and `tr` commands to change data in a pipeline.

COPYING DATA FILES FOR THIS CHAPTER

The next several exercises will provide you with experience in dealing with the wildcards used for matching filenames and pathnames. You will need a set of data files for the following exercises. If you haven't already, copy the directories **/home/wildcardExercises**, **/home/quoteExercises**, and **`/home/cutExercises`** to directories under your home directory. Remember that you'll need to use the `–r` option with `cp` to copy the folders. For example:

```
cp –r /home/wildcardExercises   ~

cp –r /home/quoteExercises   ~

cp –r /home/cutExercises ~
```

Note - the explanations in the book describe how the metacharacters work in the c-shell (csh) or tcsh, and the exercises in this chapter have also been designed to work with the csh/tcsh shell. If you're not already in the csh or tcsh shell you can temporarily change by typing `tcsh` at the command prompt. This will last until you type `exit` or close PuTTY. Or you can change your login shell so that each time you login you will automatically be in bash. This is done by typing `chsh`, and then specifying `/bin/tcsh`. If you prefer to use a different shell like the Bourne shell (sh) or Bourne again shell (bash) keep in mind that most of the metacharacters will work the same as in tcsh but there are a few whose behavior is shell dependant.

5.1 This set of exercises will provide you with experience in dealing with the ***** wildcard. As you go through each exercise remember that the point is for you to see what filenames or characters will be matched by this wildcard. You will be using the `ls` command, but these wildcards will work with any command that works with files, such as `cp`, `rm`, etc.

A. Ensure that you are in your wildcardExercises directory when working on the following exercises. First move to your home directory by typing:

```
cd ~
```

B. Next, move to the **wildcardExercises** subdirectory by typing:

```
cd wildcardExercises
```

C. Use the `ls -al` command to familiarize yourself with the files in the directory. You may want to pipe it through `more` to ensure that you see all the files. Create a screenshot showing the files in the directory.

D. Test the `*` wildcard by typing:

```
ls *
```

Which files are displayed? Which files are not displayed? (You don't need to write down your answers to these questions, but you should be able to formulate a response.)

E. Type: `ls .*`

Which files are displayed? Which files are not displayed? (You don't need to write down your answers to these questions, but you should be able to formulate a response.)

F. Type: `ls *.*`

Which files are displayed? Which files are not displayed? (You don't need to write down your answers to these questions, but you should be able to formulate a response.)

G. Type: `ls prog*`

Which files are displayed?

5.2 This set of exercises provides more practice using the * wildcard.

A. What would you use to display a listing of all the files that start with the letter **`f`**? (Note – this is looking for files that start with just the letter **f** without the question mark. The question mark is part of the punctuation for the question.)

____

B. What would you use to display a listing of all the files that start with the letters **`file`**? (Note – this is looking for files that start with just the letters **file** without the question mark. The question mark is part of the punctuation for the question.)

____

C. What would you use to display a listing of all the files that start with the characters **`.j`**? (Note – this is looking for files that start with a **.** followed by a **j** without the question mark. The question mark is part of the punctuation for the question.)

____

5.3 This set of exercises provides practice using the ***** wildcard with the `cd` command to autocomplete folder names.

`A.` Ensure that you're in your copy of the **wildcardExercises** folder.

`B.` Display the folders that start with the characters **aa** by typing: `ls aa*` How many folders are there?

`C.` Assume you want to change into the folder **aaron**? Would the command **cd aa*** accomplish this? Yes / No

`D.` What is the minimum string you would be able to use to change into the folder **aaron**?

`E.` Assume you want to change into the folder **aardvark**? Would the command **cd aard*** accomplish this?
Yes / No

`F.` What is the minimum string you would be able to use to change into the folder **aardvark**?

5.4 The next several exercises will provide you with experience in dealing with the `?` wildcard. As you go through each exercise remember that the point is for you to see what filenames or characters will be matched by this wildcard.

`A.` Test the `?` wildcard which is used to match any single character by typing:

```
ls ?
```

Which files are displayed?

`B.` Type: `ls prog?`

Which files are displayed?

`C.` Type: `ls prog1.?`

Which files are displayed?

`D.` Type: `ls prog?.o`

Which files are displayed?

E. Assume that you have several different versions of your resume. You've named the files `resume`*`n`*`.txt`, where *`n`* is a single digit number. What would you use to display a listing of all these files?

____

F. Assume that you have several different versions of a file containing top secret information. You've named the files `.junk`*`n`*, where *`n`* is a single digit number. What would you use to display a listing of all these files?

____

5.5 The next several exercises will provide you with experience in dealing with the `[xyz]` and `[a-n]` wildcards which are used to match a single character from a set or range. As you go through each exercise remember that the point is for you to see what filenames or characters will be matched by this wildcard.

A. Test the `[`*`abc`*`]` wildcard which is used for matching a single character from a set, by typing:

```
ls f[123]
```

Which files are displayed?

B. Test the `[`*`x-y`*`]` wildcard which is used for matching a single character from a range of characters by typing:

```
ls data[A-Z]
```

Which files are displayed?

C. Assume that you have several different versions of file containing tables of data from stress tests performed on your hydraulic flux pernambulator. You've named the files `table`*`nn`*, where *`n`* is a number followed by a single lower-case text character such as 1a, or a 2 digit number.

What would you use to display a listing of all the files from the first set of test runs? These are the files that start with the characters **table1**, followed by any lower-case character. You should not display files with names such as **table11**, **table12**, etc.

____

D. What would you use to display a listing of all the files from the third set of test runs? These are the files that start with the characters **table3**, followed by any lower-case character. You should not display files with names such as **table32**, **table33**, etc.

____

5.6 The next several exercises will provide you with experience in dealing with the `{pat1,pat2,pat3}` wildcard which is used to match different strings of characters . As you go through each exercise remember that the point is for you to see what filenames or characters will be matched by this wildcard method. Also remember that you do NOT want any space on either side of the comma being used as the delimiter between patterns.

A. Test the `{pat1,pat2,pat3}` wildcard which is used for matching a strings from a set, by typing:

```
ls resume1.{old,txt}
```

Which files are displayed?

B. Test the `{pat1,pat2,pat3}` wildcard which is used for matching a strings from a set, by typing:

```
ls resume1.{gesa,pnnl}
```

C. What would you use to display a listing of all the files that start with the characters **resume1.** followed by the characters **cbc** or the characters **pnnl**?

____

D. What would you use to display a listing of all the files that start with the characters **resume2.** followed by the characters **bak** or the characters **old**?

____

5.7 This set of exercises will provide you with experience in combining wildcards used for matching file names.

A. Test combining the `*` wildcard with the `{pat1,pat2,pat3}` wildcard, by typing:

```
ls *.{txt,old}
```

Which files are displayed?

B. Test combining the `*` wildcard with the `[abc]` wildcard, by typing:

```
ls f*.[oc]
```

Which files are displayed?

C. Test combining the `?` wildcard with the `{pat1,pat2,pat3}` wildcard, by typing:

```
ls resume?.{txt,old}
```

Which files are displayed?

D. What would you use to display a listing of all of the files that start with the letter `p` and end with `.c` or `.o`?

____

E. What would you use to display a listing of all of the files that start with the string `cv` or `resume` and end with `.cbc` or `.gesa`?

____

5.8 This set of exercises will provide you with experience in dealing with the wildcards used for matching path names. You can use some of the same wildcards as with the file names, but there is one new wildcard ~ for matching the path to your home directory. The official name of this character is tilde, but it's commonly called a twiddle.

As you go through each exercise remember that the point is for you to see what path names will be matched by each wildcard.

A. Move to the **/home** directory.

B. Use the `pwd` command to print the current working directory, then create a screenshot.

C. Type: `ls ~`

Which files are displayed?

- The files in the /home directory
- The files in my home directory
- The ls command returns an error and no files are displayed

D. Type: `ls ~/.*`

Which files are displayed?

- The hidden files in the /home directory
- The hidden files in my home directory
- The ls command returns an error and no files are displayed

`E.` Move to the `/etc` folder by typing: `cd /etc`

Now type: `cd ~`

Which directory do you move to? Use the `pwd` command to print the current working directory, then create a screenshot.

`F.` Move down into the `wildcardExercises` folder by typing: `cd wild*`

Note – if you have more than one directory in your home directory that starts with the characters **wild** then you may either change into the first directory that starts with **wild** or it may will result in an error message. The result varies by distribution.

`5.9` This set of exercises will provide you with experience in using the file and path wildcards with commands other than `ls`.

A. What would you use to copy all the files that start with the string **`table1`** or **`table2`** to a directory named **`oldData`**?

____

B. What would you use to delete all the files that start with the string **`f1`** ? Note – take care if you test this command as it will delete several of the test files.

____

C. What would you use to delete all the files that start with the string **`cv`** or **`resume`** and end with **`.old`** or **`.bak`**? Note – take care if you test this command as it will delete several of the test files.

____

5.10 In this exercise you will gain experience displaying the shell variables. Display the all of the variables being stored and used by your shell process by typing the following:

```
set | more
```

Read through the list of variables and see if you can decipher what they're being used for. Don't worry if you can't figure this out, it's not critical for you to know at this point in your Linux career. The purpose of this exercise is simply for you to see that your instance of the shell process is using several variables. After you've read through the list, create a screenshot.

5.11 In this exercise you will gain experience displaying the values stored in shell variables.

A. Display the value of the `$user` and `$cwd` shell variables by typing:

```
echo my username is: $user and the current directory
is: $cwd
```

What is displayed?

B. Change to the /etc directory and display the values held in the variables again by typing the following commands:

```
cd /etc
echo my username is: $user and the current directory
is: $cwd
```

Create a screenshot showing the results.

C. What command would you use to display the shell variable **uid**, which holds the User ID associated with your account.?

____

D. What command would you use to display the shell variable **home**, which holds the path to your home directory?

____

5.12 This exercise provides practice using the \ character to continue commands across multiple lines. Note – the `<enter>` means to hit the **Enter** key. Type the following and use the results to answer the questions.

```
echo Hello \  <enter>
 World <enter>
```

A. Is the `echo` command executed before or after you enter the second line? Before / After

B. Does the shell display anything different after you type the "\" to let you know that you are continuing the command line? Yes / No

C. If so, what character(s) are displayed?

____

5.13 This exercise provides additional practice using the \ character to continue commands across multiple lines. Note – the `<enter>` means to hit the **Enter** key. Type the following:

```
echo the guitar player for Guns and Roses  \  <enter>
was nicknamed Slash \  <enter>
He was really whack. \ <enter>
Slash ... Whack ... Get it<enter>
```

Create a screenshot showing your work.

5.14 In this set of exercises, you will gain experience with using `\t` and `\n` with the `echo` command to print tabs and newlines.

A. Note - For this exercise to work you probably need to be in the tcsh shell because on some distributions the `echo` command works a little differently in the bash shell. If you're in the bash shell you can still do the exercise and use the `echo` command, but you'll have to add the `-e` option to the `echo` command as a switch to tell it to expand the \t and \n character sequences, as opposed to printing them literally.

If you don't know which shell you're currently in type the `ps` command. It will show you what processes you are currently running and output something similar to:

```
  PID TTY          TIME CMD
31717 pts/0    00:00:00 tcsh
33480 pts/0    00:00:00 ps
```

If you see that you're running tcsh you'll be ok to proceed with the exercise and can skip to Step B. If the display looks like the following and you see bash instead of tcsh, you'll need to switch to tcsh.

```
  PID TTY          TIME CMD
31717 pts/0    00:00:00 bash
33480 pts/0    00:00:00 ps
```

You can temporarily switch to tcsh by typing:

```
tcsh
```

When you're done with the exercises you can return to bash by typing:

```
exit
```

B. Remember that the `echo` command prints strings. Type the following, and ensure you include the double quote characters around the entire string. If you leave them off, you won't get the expected results.

```
echo "Hello World!"
```

Now add some tabs by typing the following.

```
echo "\tHello \tWorld!"
```

Note how the `\t` characters affect the display? Do you see the `\t` characters, or did they cause something else to happen? Next, add a newline by typing:

```
echo "\tHello \n\tWorld!"
```

What is displayed? Do you see the `\t` and `\n` characters, or did they cause something else to happen? Create a screenshot showing the output from the `echo` command.

C. Apply what you've just learned by creating the following display using a *single* `echo` command. Hint: use tab characters to get the columns to line up, and newlines to do multiple lines with a single echo command. Create a screenshot showing your command and the result.

College	Phone Number
CBC	547-0511
WSU	372-7250

Note – you will need the files from **/home/quoteExercises** to complete the next few exercises. You must copy this directory to your home directory and move into it to complete the assignments

5.15 . This exercise provides experience using \ to protect characters with special meaning.

A. Use the `ls` command to see which files will be matched by the ***** wildcard by typing:

```
ls  *
```

Next, use the `echo` command to display this same list of files by typing:

```
echo  *
```

Now, use the \ before the `*` to protect it from being expanded, and get the `echo` command to display just the splat by typing:

```
echo  \*
```

If you see the complete list of files again it means that there's a space between the `\` and the `*`. Create a screenshot showing the result of the last command.

B. Use the `echo` command to see the current value of the `$cwd` variable by typing:

```
echo $cwd
```

Now, use the \ before the `$` to protect it from being interpreted as a variable by typing:

```
echo  \$cwd
```

Create a screenshot showing the result.

C. Use the `echo` command to add tabs and newlines to the text being displayed by typing. (Remember to add the quotes, otherwise you will get unexpected results.)

```
echo "Name:\t Ben Dover \nEmail:\t bendover@gmail.com"
```

Now, use the \ before the \ to protect the `\t`'s and `\n`'s from being interpreted as tabs and newlines by typing:

```
echo "Name:\\t Ben Dover \\nEmail:\\t bendover@gmail.com"
```

Create a screenshot showing the result.

D. The output from previous exercise is a little crazy, as this is something you'd never do in real life. However, it does demonstrate how the \ protects the following character from taking on any special meaning, even if the next character is also a whack! This exercise demonstrates a more practical use of using the \ to protect another \. Assume you're trying to create a short tutorial on using tabs and newlines with the echo command so you type:

```
echo "\t: inserts a tab and \n inserts a newline"
```

In the space below enter what you would use to fix the command so that it displays the **\t** as a whack followed by a **t**, and so that it displays the **\n** as a whack followed by an **n**.

____

5.16 These exercises provide additional practice protecting file name wild cards with a whack.

A. Assume you want to use the `touch` command to create a file named **`whyWorry`**`?`. That is, you want a question mark to be the last character in the file name. Which of the following would accomplish this?

- `touch whyWorry?`
- `touch whyWorry\ ?`
- `touch whyWorry\?`
- `touch whyWorry?\`
- There is no way to accomplish this

B. Assume you have downloaded some files and find one named **`f*`**. That is, the file name is **f** followed by a splat. How would you delete this one file without deleting all the files that start with the letter **f**?

- `rm f*`
- `rm f?`
- `rm f\ *`
- `rm f\*`
- `rm f*\`
- There is no way to accomplish this

5.17 This set of exercises will demonstrate whether enclosing a string inside single quote or double quote characters protect the various file and path wildcards from expanding or not. In each exercise, you just need to determine whether the wildcard was expanded or not. If it is expanded, you will see a list of files that match the filename and wildcard. If it is not expanded you will see either an error message saying that there are no matching files, or a list of files that match the exact characters you have specified. Use the results from the exercises to answer the questions at the end of the exercise. You don't need to create screenshots for any of these exercises, but make sure that you answer the questions in the last Step.

A. Ensure that you are in your **quoteExercises** directory when working on the following exercises. You can use wildcards to help change directories. First move to your home directory by typing: `cd ~`

Next, move to the **quoteExercises** subdirectory by typing: `cd quo*`

B. Next, we'll check how quotes affect wildcard expansion for matching filenames. First try the `*` wildcard without any quotes. Type: `ls *`

Which files are displayed?

C. Type: `ls '*'` ← These are single quotes

Which files are displayed? Is there any difference between using single quotes and using no quotes at all?

D. Type: `ls "*"` ← These are double quotes

Which files are displayed? Is there any difference between using double quotes and using no quotes at all?

E. Type: `ls j*`

Which files are displayed?

F. Type: `ls  'j*'` ← These are single quotes

Which files are displayed?

G. Type: `ls "j*"` ← These are double quotes

Which files are displayed?

H. Type: `ls  'junk*'` ← These are single quotes

Which files are displayed?

I. Type: `ls "junk*"` ← These are double quotes

Which files are displayed?

J. Type: `ls "junk?"` ← These are double quotes

Which files are displayed?

K. Type: `ls 'junk?'` ← These are single quotes

Which files are displayed?

L. Use the results from the previous steps to answer the following questions:

- True or False. Double quotes protect the `*` character and prevent it from being interpreted as a wildcard and from being expanded.
- True or False. Single quotes protect the `*` character and prevent it from being interpreted as a wildcard and from being expanded.
- True or False. Double quotes protect the `?` character and prevent it from being interpreted as a wildcard and from being expanded.
- True or False. Single quotes protect the `?` character and prevent it from being interpreted as a wildcard and from being expanded.

You probably noticed that there isn't any difference between using single quotes or double quotes when it comes to wildcard expansion. That is, "junk*" and 'junk*' expand to the same thing. So why have two different sets of quotes? The reason is that there are some other characters, besides the filename wildcards, that have special meaning to the shell, and the quotes tell the shell whether they should have special treatment or whether they're just another text character.

5.18 In this set of exercises, you will gain experience displaying the values stored in shell variables, and then attempting to protect the `$` character using single quotes and double quotes. Use the results from the exercises to answer the questions at the end of the exercise. You don't need to create screenshots for any of these exercises, but make sure that you answer the questions in the last step.

A. Now do some testing to see if you can use the different quote characters to protect the $ and prevent the shell from expanding it. Test the effect of double quotes by typing:

`echo "$user $cwd"` ← These are double quotes

What is displayed? Is there any difference between using double quotes and using no quotes at all?

B. Now test the effect of single quotes by typing:

`echo '$user $cwd'` ← These are single quotes

What is displayed? Is there any difference between using single quotes and using no quotes at all?

C. Use the results from the previous steps to answer the following questions.

- True or False. Double quotes protect the `$` character and prevent it from being interpreted as part of a shell variable.

- True or False. Single quotes protect the `$` character and prevent it from being interpreted as part of a shell variable.

5.19 In one of the previous exercises you used the `\t` and `\n` sequences to print tabs and newlines. Remember that in those exercises you were directed to ensure that double quotes were used. In this set of exercises you will review using `\n` and `\t` in quoted strings, and then see the effects of using no quotes. Create a screenshot at the end of this exercise.

A. Test how the slash sequences are affected by double quotes. Type:

```
echo –e "\tHello \n\tWorld!"
```

What is displayed? Are the `\t` and `\n` characters displayed or expanded?

B. Test how the slash sequences are affected by single quotes. Type:

```
echo –e '\tHello \n\tWorld!'
```

What is displayed? Are the `\t` and `\n` characters displayed or expanded?

C. Test how the slash sequences are affected by using no quotes. Type:

```
echo –e \tHello \n\tWorld!
```

What is displayed? Are the `\t` and `\n` characters displayed or expanded?

D. Use the results from the previous steps to answer the following questions.

- True or False. Double quotes protect the `\n` and `\t` characters and prevent them from being interpreted as newlines and tabs.
- True or False. Single quotes protect the `\n` and `\t` characters and prevent them from being interpreted as newlines and tabs.
- True or False. Using no quotes protects the `\n` and `\t` characters and prevents them from being interpreted as newlines and tabs.

5.20 This exercise provides practice using \ to protect quote characters. In the space provided, write the command you would use to print each of the specified strings.

A. This is a double quote "

____

B. That's Grandma's grammar book

____

C. He's "literally" cooked

____

D. Don't pay $100

____

5.21 This set of questions provides you with feedback on your ability to use characters that have special meaning to the shell, and how to protect or escape those characters. Note that these are "thought" exercises and the files and folders referred to in the exercises may not exist.

A. Which of the following wildcards matches any single character?
   i. *
   ii. ?
   iii. ~
   iv. %
   v. None of the above

B. Which of the following wildcards matches any number of any character(s)?
   i. *
   ii. ?
   iii. ~
   iv. %
   v. None of the above

C. Which of the following wildcards matches any single character from the set **asdfghjkl**?
   i. (asdfghjkl)
   ii. {asdfghjkl}
   iii. [asdfghjkl]
   iv. <asdfghjkl>
   v. None of the above

D. Which of the following wildcards matches any single uppercase character?
   i. (A-Z)
   ii. {A-Z}
   iii. [A-Z]
   iv. <A-Z>
   v. None of the above

E. Which of the following wildcards matches the characters **old** or the characters **new** ?
   i. (old,new)
   ii. {old,new}
   iii. [old,new]
   iv. <old,new>
   v. None of the above

F. Assume you want to see a listing of all files in the current directory that start with the characters **bird**. Which of the following would accomplish this?
   i. ls bird(any)
   ii. ls bird#
   iii. ls bird?
   iv. ls bird*
   v. none of the above

G. Assume the directory you're currently in has sub-directories named **james**, **jamming**, **jamaal**, **january** and **jamestown**. You want to move into the **jamestown** directory. Would the command `cd jame*` accomplish the desired directory change?
   i. Yes
   ii. No

H. Assume you want to see a detailed listing of all the files in a directory, including the hidden files. Would the command `ls -al *` accomplish this?
   i. Yes
   ii. No

I. Assume you want to ignore my advice and create a file with spaces in the name. That is, you want to create a file named **best day ever** . Which of the following would accomplish this?
   i. `touch best day ever`
   ii. `touch 'best day ever'` (this uses single quotes)
   iii. `touch "best day ever"` (this uses double quotes)
   iv. `touch best\ day\ ever`
   v. It's not possible to create a file with this name

5.22 These exercises provide experience using the `date` command and setting the format of the output. This is a preliminary step to using back ticks to do command substitution. Create a screenshot when you have completed all the steps in this exercise.

A. Type the following to see what `date` returns by default, without any special formatting:

```
date
```

B. Type the following to have the `date` command display the current day of the week.

```
date "+%A"
```

C. Type the following to have the `date` command display the current day of the week followed by the year.

```
date "+%A %Y"
```

D. Type the following to have the `date` command display the numbers for the current month and day.

```
date "+%m%d"
```

E. Type the following to have the `date` command display the current time in 12 hour format.

```
date "+%r"
```

5.23 These exercises provide experience formatting the output of the `date` command.

A. In the space provided enter what you would type to have the `date` command display the numbers for the current month and year.

____

B. In the space provided enter what you would type to have the `date` command display the numbers for the current hour (in 24 hour format) and minute.

____

C. In the space provided enter what you would type to have the `date` command display the 2 digit numbers for the current month, day and year.

____

5.24 These exercises provide experience using the `expr` command. . This is a preliminary step to using back ticks to do command substitution.

A. Which of the following is the correct way to use the expr command to add 33 and 7890? Circle the correct answer

- `expr 33 \+ 7890`
- `expr 33\+7890`
- `expr 33+7890`
- `expr 33 + 7890`

B. Which of the following is the correct way to use the expr command to subtract 33 from 7890? Circle the correct answer
- `expr 7890 \- 33`
- `expr 7890\-33`
- `expr 7890-33`
- `expr 7890 - 33`

C. Which of the following is the correct way to use the expr command to multiply 33 times 66? Circle the correct answer
- `expr 33 \* 66`
- `expr 33\*66`
- `expr 33*66`
- `expr 33 * 66`

5.25 These exercises provide experience using the `expr` command.

A. In the space provided enter what you would type to have the `expr` command add 89 to 100.

____

B. In the space provided enter what you would type to have the `expr` command subtract 89 from 100.

____

C. In the space provided enter what you would type to have the `expr` command divide 144 by 12.

____

D. In the space provided enter what you would type to have the `expr` command multiple 16 by 16.

____

5.26 These exercises provide experience using the back ticks to combine commands.

A. Assume that you want to use the `touch` command to create a file with a filename that starts with **mySecretDiary.*mmdd*** where *mm* is the 2 numbers representing the current month and *dd* is the 2 numbers representing the current day. For example, if the current date is April 1 then the file name would by **mySecretDiary.0401**. In the space provided enter what you would type to combine the `date` command and the `touch` command to accomplish this.

____

B. Assume that you want to use the `rm` command to delete a file named **tempAccounts.*mmddyy*** where *mm* is the 2 numbers representing the current month, *dd* is the 2 numbers representing the current day, and *yy* is the 2 numbers for the current year.

____

5.27 This set of exercises provides experience running multiple commands, separating them with the **;** character.

A. Display the contents of the / directory by typing:

```
ls /
```

Next display the contents of /home by typing:

```
ls /home
```

Now do this in one line by typing:

```
ls / ; ls /home
```

Create a screenshot showing the outcome of the commands.

B. Sometimes you want to clear all the clutter off your screen before running a command. There are two ways to do this. The first is to pick up your computer and shake it like an etch-a-sketch and the second is to use the `clear` command. Of course, I'm just kidding about shaking your computer, but did you think about trying it for just a second? Type the following to clear the screen:

```
clear
```

Next print **Hello World** by typing:

```
echo "Hello World"
```

Now do this in one line by typing:

```
clear; echo "Hello World"
```

Create a screenshot showing your work

C. Assume that you want to clear the screen, print the current date followed by the message "***YourName* is in the House**" where *YourName* is your name. In the space provided enter what you would type to accomplish this on one line.

____

D. Assume that you want to clear the screen, display a list of the files in your home directory followed by the message "**There is no place like home**". In the space provided enter what you would type to accomplish this on one line.

____

5.28 This set of exercises provides practice using **>** to redirect stdout to a file. In these exercises you're going to do the following steps:

1. Ensure you're in your home directory.
2. Check to ensure that a file does not exist, so you can create it.
3. Redirect the output from a command into a file, then verify that the file was created and contains the commands output.
4. Redirect the output from a second command into the same file, then check to see if the original content was overwritten by the output from the second command.

A. Ensure you're in your home directory so that you will be certain to have permission to create new files. You can do this by typing:

```
cd ~
```

B. Display a list of files in the directory by typing:

```
ls
```

Ensure that the file named **testFile** does not exist. If it does, use the `rm` command to delete it.

C. Display the current system date by typing:

```
date
```

In this case, what is stdout? That is, where is the output from the `date` command sent? Circle the correct answer:

- The monitor or display
- There is no stdout for this command

D. Now, use the redirect to send the output from the `date` command to the file named **testFile** by typing:

```
date > testFile
```

- Is the output from the `date` command displayed on the screen? [Yes / No]
- Is the file **testFile** created when you use the redirect? [Yes / No]
- Use the `cat` or `more` command to check the contents of the file **testFile**. Where is the output from the `date` command displayed? Circle the correct answer.
  - The monitor or display
  - The file **testFile**
  - There is no stdout for this command

E. Check to see what happens to the content of an existing file if the output from a command is redirected to the file. First, ensure **testFile**, the file you created in the previous step exists and contains the output from the `date` command by typing:

```
cat testFile
```

Next use the redirect to send the output from the `echo` command to the file by typing:

```
echo "hello world" > testFile
```

Inspect the contents of **testFile**. What does it contain? Circle the correct answer.

- The output from the `date` command. The file was not overwritten.
- The output from the `echo` command. The file was overwritten.
- The output from the `date` command followed by the output from the `echo` command. The file was not overwritten.
- The file does not exist

5.29 This set of exercises provides practice using >> to redirect stdout to a file. In these exercises you're going to do the following steps:

1. Ensure you're in your home directory.
2. Check to ensure that a file does not exist, so you can create it.
3. Redirect the output from a command into a file using >>, then verify that the file was created and contains the commands output.
4. Redirect the output from a second command into the same file, then check to see if the output from the second command is appended to the existing content, or if it overwrites the existing content.

A. Ensure you're in your home directory so that you will be certain to have permission to create new files. You can do this by typing:

```
cd ~
```

B. Display a list of files in the directory by typing:

```
ls
```

Ensure that the file named **testFile** does not exist. If you didn't delete it after the last exercise, then it's probably still there. If so, use the `rm` command to delete it.

C. Display the current system date by typing:

```
date
```

D. Now, use the double redirect >> to send the output from the `date` command to the file named **testFile** by typing:

```
date >> testFile
```

- Is the output from the `date` command displayed on the screen? [Yes / No]
- Is the file testFile created when you use the double redirect? [Yes / No]
- Use the `cat` or `more` command to check the contents of the file **testFile**. Where is the output from the `date` command displayed? Circle the correct answer.
    - The monitor or display
    - The file testFile
    - There is no stdout for this command

E. Check to see what happens to the content of a file if the output from a command is redirected to the file using >>. First, ensure the file you created in the previous step exists and contains the output from the `date` command by typing:

```
cat testFile
```

Next use the redirect to send the output from the `echo` command to the file by typing:

```
echo "I love Linux" >> testFile
```

Inspect the contents of testFile. What does it contain? Circle the correct answer.

- The output from the `date` command. The file was not overwritten or changed.
- The output from the `echo` command. The file was overwritten.
- The output from the `date` command followed by the output from the `echo` command. The file was not overwritten, and the output from the `echo` command was appended to the end of the file.
- The output from the `echo` command followed by the output from the `date` command. The file was not overwritten, and the output from the `echo` command was added to the beginning of the file.
- The file does not exist

5.30 This set of exercises provides practice in applying what you have just learned about redirecting stdout using > and >>. For each of the following write what you type on the command line to accomplish the specified task:

A. Write the command to create a file named `aFiles` that contains a list of all the files in the directory `/usr/lib`

`ls /usr/lib > aFiles  (This answer has been provided as an example.)`

B. Write the command to create a file named `aFiles` that contains a list of all the files in the directory `/usr/lib` that start with the letter `a`

____

C. Write the command to create a file named `libFiles` that contains a list of all the files in the directory `/usr/lib` that start with the letters `lib`

____

D. Write the command to create a file named `soFiles` that contains a list of all the files in the directory `/usr/lib` that end with the letters `so`

____

E. Write the command(s) to create a file named `currentUsers` that contains the current date followed by a list of all the users currently logged into the system. (Hint1 – use the `who` command to display a list of users. Hint 2 – use the ; character to write two commands on the same line.)

____

F. What would be the result of the following?

```
date > files.dat; ls -al >> files.dat
```

That is, when all the commands have been executed what do you expect to see in the file `files.dat`? Circle the correct answer.

- Only the output from the `date` command
- Only the output from the `ls –al` command
- The output from the `date` command followed by the output from the `ls-al` command
- The output from the `ls -al` command followed by the output from the `date` command
- Nothing

G. What would be the result of the following?

```
ls -al >> files.dat; date > files.dat
```

That is, when all the commands have been executed what do you expect to see in the file `files.dat`? Circle the correct answer.

- Only the output from the `date` command
- Only the output from the `ls –al` command
- The output from the `date` command followed by the output from the `ls-al` command
- The output from the `ls -al` command followed by the output from the `date` command
- Nothing

5.31 This exercise demonstrates how the shell can protect you from overwriting existing files using `noclobber`. You will create a screenshot after the last step in the exercise to show your work.

A. Ensure you're in your home directory.

B. Set the `noclobber` variable. This is done by typing:

```
set noclobber
```

C. Ensure that the file **testFile** does not exist. If it does, use the `rm` command to delete it.

D. Execute the `date` command and redirect its output to **`testFile`** by typing the following.

```
date > testFile
```

E. Test `noclobber` by trying to redirect the output from another command to the file **`testFile`**. Type the following:

```
echo "cyber security" > testFile
```

Do you get an error message, or is the content of the file `testFile` overwritten? You should see an error message similar to "`testFile: file exists"` which is the shell's way of telling you that it will not overwrite an existing file.

F. To remove the protection against accidental overwriting you must remove the `noclobber` variable. This is done by typing:

```
unset noclobber
```

Test this by trying to redirect the output from another command to the file **`testFile`**. Type the following:

```
ps -alx > testFile
```

Do you get an error message, or is the content of the file `testFile` overwritten? Use the `cat` command to display testFile, and then create a screenshot.

5.32 This exercise demonstrates how `>!` can be used to override `noclobber`. You will create a screenshot after the last step in the exercise to show your work.

A. Ensure the `noclobber` variable is set.

B. Ensure that the file **testFile** does not exist. If it does, use the `rm` command to delete it.

C. Execute the `date` command, and redirect its output to a file named **`testFile`** by typing:

```
date > testFile
```

D. Next, run the `ls -al` command and send the output to the file **`testFile`** by typing:

```
ls -al > testFile
```

Do you get an error message? If so, what is it and what does it mean?

E. Run the `ls -al` command again, redirecting the output to **testFile** using >! by typing:

```
ls -al >! testFile
```

Inspect the contents of the file. Was the output of the `ls` command written to the file or did the shell prevent the overwriting?

F. Test to check if the `noclobber` variable is still set by running the `date` command again, and redirecting the output to **testFile** using > by typing:

```
date > testFile
```

Inspect the contents of the file. Was the output of the `date` command written to the file or did the shell prevent the overwriting? Create a screenshot showing your work.

COPYING DATA FILES FOR THIS SECTION OF EXERCISES

Before you start this exercise you will need all of the files shown below. If you haven't already you can copy them from the directory **`/home/cutExercises`**. Or, if you have an unlimited amount of time on your hands you can create the files yourself by either using `vi`, or creating the files on the PC and using FTP to move them to your Linux account. To copy the files to your home directory use the following commands:

```
cd  ~
cp –r /home/cutExercises ~/cutExercises
```

After you have a copy of the directory, make sure you move down into it before starting the exercises.

Here are the data files, and what they contain. Note that <tab> represents a single tab character. Tabs are commonly used in Linux/UNIX files as a delimiter.

names.txt

```
1  <tab>  Jesus     <tab>     Rincon    <tab>     Mr.
2  <tab>  Sam       <tab>     Fader     <tab>     Mr.
3  <tab>  Li        <tab>     Yamagura  <tab>     Dr.
4  <tab>  Everett   <tab>     Simpson   <tab>     Mr.
5  <tab>  Renee     <tab>     Vasquez   <tab>     Mrs.
6  <tab>  Franklin  <tab>     Smith     <tab>     Mr.
7  <tab>  Lindell   <tab>     Hall      <tab>     Ms.
8  <tab>  Pat       <tab>     Benson    <tab>     Mrs.
9  <tab>  Lucy      <tab>     Wang      <tab>     Dr.
10 <tab>  Ahmad     <tab>     Eljamal   <tab>     Mr.
```

address.txt

```
1  <tab>  122 Wicker       <tab>  Pasco     <tab> WA    <tab>  99345
2  <tab>  67 Blue Ford St  <tab>  Haley     <tab> ND    <tab>  77547
3  <tab>  888 N 45th       <tab>  Seattle   <tab> WA    <tab>  87888
4  <tab>  1856 Cedar Rd    <tab>  Paris     <tab> TX    <tab>  43444
5  <tab>  997 N Verde      <tab>  Trevor    <tab>  OR   <tab>  97997
6  <tab>  56  Crab Creek   <tab>  Snyder    <tab>  AK   <tab>  68521
7  <tab>  749 Naeflus Ct   <tab>  Perry     <tab>  FL   <tab>  76765
8  <tab>  933 Hendrix Av   <tab>  Havre     <tab>  MI   <tab>  25422
9  <tab>  76 98th S        <tab>  Ashland   <tab>  FL   <tab>  23242
10 <tab>  7444 Hewe Ct     <tab>  Hattrick  <tab>  MI   <tab>  20092
```

phone.txt

```
1  <tab>  (355)     <tab>     344-0909
2  <tab>  (324)     <tab>     399-9654
3  <tab>  (222)     <tab>     665-9987
4  <tab>  (212)     <tab>     878-2176
5  <tab>  (891)     <tab>     231-2187
6  <tab>  (737)     <tab>     865-6664
7  <tab>  (389)     <tab>     893-5543
8  <tab>  (405)     <tab>     267-9876
9  <tab>  (834)     <tab>     854-4432
10 <tab>  (388)     <tab>     329-4422
```

points.txt

```
8. 765
2. 763
1. 755
3. 740
9. 732
5. 730
4. 717
6. 716
7. 702
10. 650
```

Note that the numbers in the first field of this file are not sorted.

guitars.txt

```
Gibson
Fender
Taylor
Gibson
Rickenbacker
Fender
Ovation
Gibson
National
Fender
Taylor
PRS
Epiphone
Gretsch
Fender
```

5.33 This exercise provides practice using the `cut` command to extract a set of specified fields from a file.

A. Use the `cut` command to cut the last names from the file **names.txt**. Create a screenshot showing the result.

B. Use the `cut` command to cut the first and last names from the file **names.txt**. Create a screenshot showing the result.

C. Use the `cut` command to cut the first three characters from the file **guitars.txt**. Create a screenshot showing the result.

5.34 This exercise provides practice using the `paste` command to paste different files together.

A. Use the paste command to paste the fields in the phone.txt file to the fields in the names.txt file by typing:

```
paste names.txt  phone.txt
```

Notice what character is placed between the entries from the different files, that is, which character is used as a delimiter. Create a screenshot showing the result of the command.

B. Use the `paste` command to merge the data in the files names.txt and phone.txt by typing:

```
paste names.txt address.txt > nameAndAddress.txt
```

Use the `more` or `cat` command to display nameAndAddress.txt. Create a screenshot showing the content of the file.

5.35 **Practice with `cut` and `paste`.** Assume that you want to use the data from **names.txt** and **phone.txt** files to create a file with the names (first and last) along with the phone numbers.

A. Use the `cut` command to get the first and last names from the **names.txt** file.

```
cut -f2,3  names.txt > name2
```

B. Use the `cut` command to get the phone numbers from **phones.txt.**

```
cut -f2,3  phone.txt > phone2
```

C. Use the `paste` command to create the file **namePhone.txt** adding the names and phone numbers.

```
paste name2  phone2 > namePhone.txt
```

Use the `more` or `cat` command to display **namePhone.txt**. Create a screenshot showing the content of the file.

D. Clean up the temporary files

```
rm name2 phone2
```

5.36 Practice with `cut` and `paste`. Assume that you want to use the data from **names.txt** and **address.txt** files to create a file with the names (first and last) along with the address.

A. Use the `cut` command to get the first and last names from the **names.txt** file.

```
cut –f2,3  names.txt > name2
```

B. Use the `cut` command to get the address from **address.txt.**

```
cut –f2,3  address.txt > address2
```

C. Use the `paste` command to create the file **nameAddress.txt** adding the names and addresses.

```
paste name2  address2 > nameAddress.txt
```

Use the `more` or `cat` command to display **nameAddress.txt**. Create a screenshot showing the content of the file.

D. Clean up the temporary files

```
rm name2 address2
```

5.37 More practice with `cut` and `paste`. Assume that you want to use the data from **names.txt** but you want it in a different order. You would like to have the title first, followed by the first and last names.

A. Use the cut command to get the first and last names from the **names.txt** file

```
cut –f2,3  names.txt > name2
```

B. Use the cut command to get the title from **names.txt**

```
cut –f4  names.txt > name3
```

C. Use the paste command to create the file **titleNames.txt** adding the titles and names.

```
paste name3  name2 > titleNames.txt
```

Use the `more` or `cat` command to display **titleNames.txt**. Create a screenshot showing the content of the file.

D. Clean up the temporary files

```
rm name2 name3
```

5.38 More practice with `cut` and `paste`. Assume that you want to combine some of the data from **names.txt**, **phone.txt** and **address.txt** into a single file. You would like to have the person's title first, followed by the first and last names; followed by the phone number, followed by the address. Use what you have learned about `cut` and `paste` to accomplish this. Create a screenshot showing the file containing the desired information.

5.39 This exercise provides practice working with the `join` command. Before you start this exercise you will need the three files you used in the `cut` and `paste` exercises. If you haven't already copy them from the directory `/home/cutExercises` to a new subdirectory, and then move into that directory by using the following commands:

```
cd ~
cp –r /home/cutExercises joinPractice
cd joinPractice
```

A. Assume that you want to create a single file named **alumni.contacts** that contains names and addresses. Use what you have learned about `join` to accomplish this. Create a screenshot showing the result.

B. Assume that you want to create a single file named **contactInfo** that contains names and phone numbers. Use what you have learned about `join` to accomplish this. Use the `more` or `cat` command to display contactInfo. Create a screenshot showing the content of the file.

5.40 In this set of exercises you will identify stdin and stdout for commands, and commands on each side of a pipe. These might seem very basic, but to combine commands with pipes it's critical that you are able to easily recognize how stdout from one command becomes stdin for another command.

A. Use the following command to answer the questions:

```
ls –al /etc
```

Where does stdout for `ls` go? Circle the correct answer.

- To the display
- There is no stdout for the `ls` command

B. Use the following commands to answer the questions:

```
ls -al /etc | more
```

Where does stdout for `ls` go? Circle the correct answer.

- To the display
- It becomes the input for the `more` command
- There is no stdout for the `ls` command

What is stdin for the `more` command? Circle the correct answer.

- The keyboard
- The output from the `ls` command
- There is no stdin for the `more` command

C. Use the following command to answer the questions:

```
cat /etc/passwd
```

Where does stdout for the `cat` command go? Circle the correct answer.

- To the display
- There is no stdout for the `cat` command

D. Use the following commands to answer the questions:

```
    cat /etc/passwd | more
```

Where does stdout for the `cat` command go? Circle the correct answer.

- To the display
- It becomes the input for the `more` command
- There is no stdout for the `cat` command

What is stdin for the `more` command? Circle the correct answer.

- The keyboard
- The output from the `cat` command
- There is no stdin for the `more` command

E. Use the following command to answer the questions:

```
ls /etc
```

Where does stdout for `ls` go? Circle the correct answer.

- To the display
- There is no stdout for the `ls` command

F. Use the following commands to answer the questions:

```
ls /etc | tail
```

Where does stdout for `ls` go? Circle the correct answer.

- To the display
- It becomes the input for the `tail` command
- There is no stdout for the `ls` command

What is stdin for the `tail` command? Circle the correct answer.

- The keyboard
- The output from the `ls` command
- There is no stdin for the `tail` command

5.41 In this set of exercises you will identify stdin and stdout for commands, and commands on each side of a pipe. Once again, this might seem very basic. But it's crucial that you have a firm understanding of the basic concepts and that you are able to easily recognize how stdout from one command becomes stdin for another command.

A. Use the following commands to answer the questions:

```
cat guitars.txt | sort | more
```

Where does stdout for `cat` go? Circle the correct answer.

- To the display
- It becomes the input for the `sort` command
- It becomes the input for the `more` command
- There is no stdout for the `cat` command

What is stdin for the `sort` command? Circle the correct answer.

- The keyboard
- The output from the `cat` command
- The output from the `more` command
- There is no stdin for the `sort` command

Where does stdout from the `sort` command go? Circle the correct answer.

- To the display
- It becomes the input for the `cat` command
- It becomes the input for the `more` command
- There is no stdout for the `sort` command

What is stdin for the `more` command? Circle the correct answer.

- The keyboard
- The output from the `cat` command
- The output from the `sort` command
- There is no stdin for the `more` command

Where does stdout for the `more` command go? Circle the correct answer.

- To the display
- It becomes the input for the `cat` command
- It becomes the input for the `sort` command
- There is no stdout for the `more` command

B. Type the following command, look at where the input comes from and where the output goes, and then answer the questions:

```
cut –f2,3 address.txt | sort | tail -5
```

Where does stdout for `cut` go? Circle the correct answer.

- To the display
- It becomes the input for the `sort` command
- It becomes the input for the `tail` command
- There is no stdout for the `cut` command

What is stdin for the `sort` command? Circle the correct answer.

- The keyboard
- The output from the `cut` command
- The output from the `tail` command
- There is no stdin for the `sort` command

Where does stdout from the `sort` command go? Circle the correct answer.

- To the display
- It becomes the input for the `cat` command
- It becomes the input for the `tail` command
- There is no stdout for the `sort` command

What is stdin for the `tail` command? Circle the correct answer.

- The keyboard
- The output from the `cut` command
- The output from the `sort` command
- There is no stdin for the `tail` command

Where does stdout for the `tail` command go? Circle the correct answer.

- To the display
- It becomes the input for the `cut` command
- It becomes the input for the `sort` command
- There is no stdout for the `tail` command

C. Type the following command, look at where the input comes from and where the output goes, and then answer the questions:

```
wc *.txt | sort | head -3
```

Where does stdout for the `wc` command go? Circle the correct answer.

- To the display
- It becomes the input for the `sort` command
- It becomes the input for the `head` command
- There is no stdout for the `wc` command

What is stdin for the `sort` command? Circle the correct answer.

- The keyboard
- The output from the `wc` command
- The output from the `head` command
- There is no stdin for the `sort` command

Where does stdout from the `sort` command go? Circle the correct answer.

- To the display
- It becomes the input for the `wc` command
- It becomes the input for the `head` command
- There is no stdout for the `sort` command

What is stdin for the `head` command? Circle the correct answer.

- The keyboard
- The output from the `wc` command
- The output from the `sort` command
- There is no stdin for the `head` command

Where does stdout for the `head` command go? Circle the correct answer.

- To the display
- It becomes the input for the `wc` command
- It becomes the input for the `sort` command
- There is no stdout for the `head` command

5.42 This exercise provides practice is designed to provide you with some insight into the benefit of using pipes. In this exercise you'll run two series of commands that have the same end outcome. The first set of commands accomplishes a set of tasks without pipes, while the second accomplishes the same thing using pipes.

A. Type the following set of commands. Create a screenshot of the results and answer the questions.

```
cat /etc/passwd > temp1
sort temp1 > temp2
more temp2
rm temp1 temp2
```

Where does stdout for the `cat` command go?

What file does the `sort` command read?

Where does stdout for the `sort` command go?

What file does the `more` command read?

What intermediate files are created during this process?

What happens to these intermediate files in the last command?

B. Type the following set of commands. Create a screenshot of the results and answer the questions.

```
cat /etc/passwd | sort | more
```

Does this accomplish the same thing as the commands in step A.? Yes / No

Are any intermediate files created? Yes / No

Is it easier to use pipes or use the process in Step A?

5.43 This exercise will provide you with practice using redirects, pipes and the `sort` command. Assume that you want to build a list of names from the **names.txt** file. However, you would like to have the last names before the first names, and have the entire list sorted by last name. Use what you have learned about `cut, paste` and `sort` to accomplish this. Create a screenshot showing the result.

- Hint 1 – use the `cut` command to get the fields you want from the names.txt file. To get them in a different order you will have to cut each field and place it in a temporary file. You can then use the `paste` command to combine the fields in the desired order.

- Hint 2 – after you have combined the name fields, run them through `sort`.

5.44 This exercise provides a demonstration of the `uniq` command.

A. Look at the file **guitars.txt** and familiarize yourself with the content.

B. The basic use for the `uniq` command is to only print unique lines, which it does by eliminating duplicate lines. Print the unique lines in guitars.txt by using the command:

```
uniq guitars.txt
```

Did you get the result that you expected? Remember that `uniq` only works when duplicate lines are next to each other in the file which can be accomplished with the command:

```
sort guitars.txt | uniq
```

Create a screenshot showing the results.

C. The uniq command will also count how many times a line occurs if it's given the `-c` option. Find how many times each line in guitars.txt occurs by using:

```
sort guitars.txt | uniq -c
```

Create a screenshot showing the results.

D. The `uniq` command can also be used to print a list of lines that are duplicated. It only prints one copy of each line, but it will allow you to do things like find duplicate records. This is done by using the `-d` option. To do this with guitars.txt use the command:

```
sort guitars.txt | uniq -d
```

Create a screenshot showing the results.

Note – at this point you don't need to remember the details of all the different `uniq` options, you should just remember that `uniq` eliminates duplicates lines.

5.45 This exercise will provide you practice with the `grep` command. Before you start this exercise, you will need the three files in the directory **/home/grepsedPractice**. If you haven't already, copy the directory to your home directory, and then move down into the directory.

A. Assume you want to find all lines that contain the word **Love** in the file **police.lyrics**. Type the following command, then create a screenshot showing the results:

```
grep Love police.lyrics
```

B. Use `grep` to find all lines that contain the word **the** in the file **data**. Create a screenshot showing the results. (Hint – If you want `grep` to match the word **the** and not words containing **the**, like o**the**r, use quotes to include a space on either side of **the**. For example use `" the "`.)

C. Use `grep` to find all of the lines in the three lyric files that contain the word **one**. Create a screenshot showing the results. (Hint – use a filename wildcard to search all the files that end in **lyrics**. For example `*.lyrics`.)

D. Use the `grep` command to find the line in **/etc/passwd** that contains your account information by searching for your username. Create a screenshot showing the results.

5.46 This set of exercises provides practice selecting the correct command(s) to use to start a pipeline. For each of the following situations circle the command(s) that should be used to start the pipeline. Note – there are situations that require the use of more than one command.

A. Assume you want to process the last 8 lines of a file. Which of the following commands should be used to start the pipeline?
- `cat`
- `head`
- `tail`
- `grep`
- `cut`

B. Assume you want to process the 3rd, 4th, and 6th columns of a file. Which of the following commands should be used to start the pipeline?
- `cat`
- `head`
- `tail`
- `grep`
- `cut`

C. Assume you want to process all the lines containing the string **mp3** from a file. Which of the following commands should be used to start the pipeline?
- `cat`
- `head`
- `tail`
- `grep`
- `cut`

D. Assume you want to process the middle 5 lines from a file that contains 70 lines of data. Which of the following commands should be used to start the pipeline?
- `cat`
- `head`
- `tail`
- `grep`
- `cut`

E. Assume you want to process all the data from a file. Which of the following commands should be used to start the pipeline?
- `cat`
- `head`
- `tail`
- `grep`
- `cut`

5.47 This set of exercises provides practice starting and building short command pipelines. To perform these exercises you need to be in your `cutExercises` directory.

A. How many times does the word **Fender** appear in the last 8 lines the file **guitars.txt**? Create a screenshot showing the results. Here are a few hints:
- Start with one command at a time and ensure they do what you expect.
- Start the pipeline by displaying the last 8 lines
- Next, add the command find the lines with the word Fender
- The last command should count these lines for you. Use the `wc -l` command to accomplish this.

B. Sort the first 4 phone numbers the file **phone.txt**, so that they displayed in ascending order of the area code. Create a screenshot showing the results.

C. Sort and display all the fields in the file **address.txt** except the first field. That is, display the street address, city, state and zip code, sorted by the street address. Create a screenshot showing the results. Hint – use the `cut` command to start the pipeline.

D. How many people listed in the first 5 lines the file **names.txt** have the title **Dr.**? Create a screenshot showing the results.

5.48 This exercise provides practice using the `sed` command. Before you start this exercise, you will need the three files in the directory **/home/grepsedPractice**. If you haven't already, copy the directory /home/grepsedPractice to your home directory, and then move into your copy of grepsedPractice.

A. Change all occurrences of the word **one** in the file **threeDog.lyrics** to **three**. Create a screenshot showing the result. Hint – start the pipeline with the `cat` command, then pipe the lines through `sed` to make the changes.

B. Change all occurrences of the word **one** in the file **u2.lyrics** to **geeks on scooters**. Create a screenshot showing the result. Hint – the phrase **geeks on scooters** contains spaces, so it will cause an error if you don't protect the spaces.

C. Change all occurrences of the word **loneliness** in the file **police.lyrics** to **peanutbutter**. Create a screenshot showing the result.

D. Change all occurrences of the word **bottle** in the file **police.lyrics** to **twitter tweet**. Create a screenshot showing the result. Hint – the phrase **twitter tweet** contains spaces, so it will cause an error if you don't protect the spaces.

5.49 This exercise provides experience using a tee in command pipeline.

A. Type the following command(s) and use the result to answer the questions.

```
ls –al /etc | tee ls.out | more
```

Does the output from the `ls` command appear on the screen? Yes / No
Check the contents of the file ls.out. What does it contain?

B. Type the following command(s) and use the result to answer the questions.

```
ps –aux | tee ps.out | more
```

Does the output from the `ps` command appear on the screen? Yes / No
Check the contents of the file ps.out. What does it contain?

5.50 This exercise provides practice with conditional piping. Create a screenshot showing the results after completing the last step in the exercise.

A. Type: `ls –al ~/noSuchDir`

Is what you see on the screen an error message or output from the directory?

B. Type: `ls –al ~/noSuchDir || echo "command failed"`

Is what you see on the screen an error message, output from the directory or the output from the `echo` command?

C. Type: `ls –al ~/noSuchDir && echo "command ran successfully"`

Is what you see on the screen an error message, output from the directory or the output from the `echo` command?

5.51 This exercise provides practice grouping multiple commands using parentheses.

A. Correct the following command so that the date and the output from the `echo` command are directed to the file. Write the corrected command in the space provided.

```
date; echo “Hello World” > hello.out
```

____

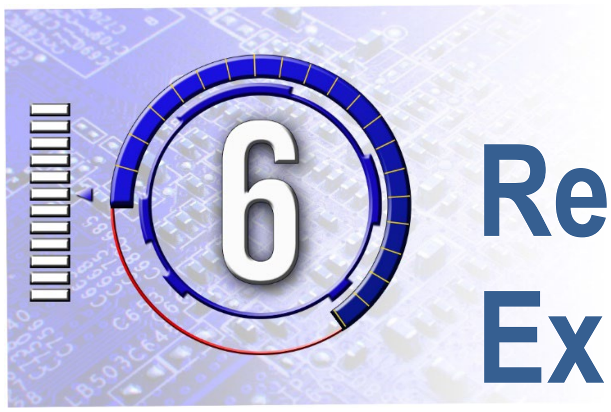
B. Correct the following command so that all the names are displayed on the screen in ascending order. Create a screenshot showing the result.

```
echo “Tony”; echo "Mike"; echo "Beth"; echo "Abe" | sort
```

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. What do the following filename wildcards represent? `*` `?` `[x-y]`
2. How do you print tabs or newlines using the `echo` command?
3. How do you print a character that has special meaning, such as *** ?** \ etc. using the `echo` command?
4. What does following match **file.*** ?
5. What does following match **file.\*** ?
6. State the three standard devices associated with shell procedures.
7. What is stdin?
8. Why would you want to change stdin?
9. What is stdout?
10. What is stderr?
11. Why would you want to change stdout or stderr?
12. How do you tell a command to send its output to a file? If you do is the file automatically overwritten? How do you control this?
13. How do you tell a command to send it's output to a file? What if file does not already exist?
14. How do you tell a command to get input from a previous command?
15. How do you take output from `ls` and view it a page at a time?
16. What commands can be used to start a pipeline by sending the data or portions of data from a file?
17. What command is used to select different columns or records from a file?
18. How are `paste` and `join` different?
19. What is `grep`?
20. What is `sed`?



Regular Expressions

The exercises in this chapter have been designed to reinforce your knowledge of regular expressions and provide hands on experience using regular expressions with Linux/UNIX commands so you can do the following:

1. Describe how regular expressions are used to match strings inside files.
2. Contrast regular expressions and filename wildcards, and describe when each should be used.
3. Build regular expressions to match a given search expression.
4. Locate, download, and modify existing regular expressions.
5. Use regular expressions in command pipelines.

Data Files
To perform the exercises in this section you will need to copy the files from the directory /home/regexpressions to your home directory. To do this type the command:

```
cp -r /home/regexpressions ~
```

When you do the exercises make sure that you are in your copy of the regexpressions directory by using the appropriate `cd` command.

6.1 How many versions of regular expression syntax are there?

A. 1
B. 2
C. 3
D. 4

6.2 What is the oldest and most inconsistent regular expression syntax?

A. PCRE
B. ERE
C. BRE
D. REI

6.3 True or False. The ERE syntax include the Perl shortcuts.

A. True
B. False

6.4 True or False. The PCRE syntax include the ERE updates.

A. True
B. False

6.5 Which of the following will run `grep` using ERE syntax? (Note that there are two correct answers.)

A. `grep`
B. `grep -e`
C. `grep -E`
D. `grep -ERE`
E. `grep -p`
F. `grep -P`

6.6 Which of the following is equivalent to running `grep` with the `-E` option?

A. agrep
B. bgrep
C. cgrep
D. dgrep
E. egrep
F. fgrep
G. None of the above

6.7 These exercises provide practice identifying strings that will be matched by literal expression patterns in regular expressions.

A. Assume that the following text is in a file named **geekPoem**. Circle the text that would be considered a match using the command `grep row geekPoem`

```
make brown storm real dead break
take row boring deal steak
fake crown torn meal reality leak
lake frown or original steal breakfast beaker
```

B. Assume that the following text is in a file named **geekPoem**. Circle the text that would be considered a match using the command `grep dea geekPoem`

```
make brown storm real dead break
take row boring deal steak
fake crown torn meal reality leak
lake frown or original steal breakfast beaker
```

C. Assume that the following text is in a file named **geekPoem**. Circle the text that would be considered a match using the command `grep eak geekPoem`

```
make brown storm real dead break
take row boring deal steak
fake crown torn meal reality leak
lake frown or original steal breakfast beaker
```

D. Assume that the following text is in a file named **geekPoem**. Circle the text that would be considered a match using the command `grep tor geekPoem`

```
make brown storm real dead break
take row boring deal steak
fake crown torn meal reality leak
lake frown or original steal breakfast beaker
```

6.8 This exercise provides practice writing simple literal expression patterns to be used in regular expressions. This requires that you are in the regexpressions directory that you copied to your home directory.

A. Display the file **simpleA1** and familiarize yourself with the content.

B. Write and execute the command to find all strings that match **CBC** in the file **simpleA1**. Create a screenshot showing the result.

C. Write and execute the command to find all strings that match **cbc** in the file **simpleA1**. Create a screenshot showing the result.

D. Write and execute the command to find all strings that match **CS** in the file **simpleA1**. Create a screenshot showing the result.

6.9 This exercise provides practice with simple literal expression patterns with meta-characters. Note – for these exercises you can assume that the regular expression will **not** be processed by the Linux/UNIX shell. But if you want to test any of these you will need to protect the meta-characters from the Linux/UNIX shell. If you test yourself, I also strongly suggest that you use `grep -E` or `egrep` to ensure you use the ERE syntax and avoid the BRE syntax which may drive you insane.You can check your answers by creating your own test file, or by using the file **/home/regexpressions/meta**.

A. What is the regular expression that will match all the occurrences of the string **$500**? Circle the correct answer.
- `$500`
- `\$500`
- `\\$500`
- `\$\500`
- None of the above

B. What is the regular expression that will match all of the occurrences of the string **ticket.prices**? Circle the correct answer.
- `ticket.prices`
- `ticket\.prices`
- `ticket.\prices`
- `ticket..prices`
- None of the above

C. What is the regular expression that will match all of occurrences of the string **(509)**? Circle the correct answer.
- `(509)`
- `\(509)`
- `\(509\)`
- `\\(509)`
- `\\(509\\)`
- None of the above

6.10 This exercise provides practice working with simple literal expression patterns that contain meta-characters Note – for these exercises the regular expression **will** be processed by the Linux/UNIX shell so you will need to protect any special characters from the shell and in the regular expression.

A. Assume that you want to find all the occurrences of the string **$500** in the file **ticketPrices**. Which of the following commands would accomplish this? Circle the correct answer.
   - `egrep $500 ticketPrices`
   - `egrep \$500 ticketPrices`
   - `egrep \\$500 ticketPrices`
   - `egrep '$500' ticketPrices`
   - `egrep '\$500' ticketPrices`
   - None of the above

B. Assume that you want to find any references to the string **chair?** in any file that ends with extension **.txt**. Which of the following commands would accomplish this? Circle the correct answer.
   - `egrep chair? *.txt`
   - `egrep chair\? *.txt`
   - `egrep chair\\? *.txt`
   - `egrep 'chair?' *.txt`
   - `egrep 'chair\?' *.txt`
   - None of the above

C. Assume that you want to find all the occurrences of the string **(509)** in the file **alumni**. Which of the following commands would accomplish this? Circle the correct answer.
   - `egrep (509) alumni`
   - `egrep \(509\) alumni`
   - `egrep \\(509\\) alumni`
   - `egrep '(509)' alumni`
   - `egrep '\(509\)' alumni`
   - None of the above

D. Assume that you want to find any references to DNS names that end in **.edu** in the file **alumni**. Which of the following commands would accomplish this? Circle the correct answer. Hint – there is more than one correct answer.
   - `egrep .edu alumni`
   - `egrep \.edu alumni`
   - `egrep \\.edu alumni`
   - `egrep '.edu' alumni`
   - `egrep '\.edu' alumni`
   - None of the above

E. Assume that you want to find any references to the string **error*** in any file that ends with **.xml** extension. Which of the following commands would accomplish this? Circle the correct answer.
   - `egrep error* *.xml`
   - `egrep error\* *.xml`
   - `egrep error\\* *.xml`
   - `egrep 'error*' *.xml`
   - `egrep 'error\*' *.xml`
   - None of the above

6.11 This exercise provides practice writing simple literal patterns to be used in regular expressions.

A. Write and execute the command to find all strings that match **the** in the file **test**. Create a screenshot showing the result.

B. Write and execute the command to find all strings that match **is** in the file **meta**. Create a screenshot showing the result.

C. Write and execute the command to find all strings that match **Key#** in the file **simpleC**. Create a screenshot showing the result.

D. Write and execute the command to find all strings that match **GPA** in the file **simpleD**. Create a screenshot showing the result.

6.12 This exercise provides practice writing literal patterns containing meta-characters to be used in regular expressions.

A. Write and execute the command to find all strings that match **.org** in the file **simpleA1**. Hint – you will have to use single quotes and a \ to protect the **.** metacharacter. Create a screenshot showing the result.

B. Write and execute the command to find all strings that match **(703)** in the file **occurB**. Hint – you will have to use single quotes and a\ to protect the **(** metacharacter. Create a screenshot showing the result.

6.13 This exercise provides practice interpreting regular expressions using single character wildcards. In each exercise you will be asked to choose the regular expression that matches the given string. For simplicity and clarity, the answers will NOT protect any meta-characters from the shell.

A. Which regular expression(s) will match all of the occurrences of the string **CS** in the followed by *any* 3 characters? Circle the correct answer(s).

- `CS...`
- `CS***`
- `CS???`
- `CS+++`
- `CS[...]`
- `CS(...)`
- None of the above

B. Which regular expression(s) will match all of the occurrences of the string **CS** in the followed by any 3 numbers? Circle the correct answer(s).

- `CS...`
- `CS[1234567890]`
- `CS[0-9]`
- `CS[0-9][0-9][0-9]`
- `CS\d`
- `CS\d\d\d`
- None of the above

C. Which regular expression(s) will match all of the occurrences of the string **CS**, followed by a number, followed by any 2 upper- or lower-case alpha characters? Circle the correct answer(s).

- `CS...`
- `CS[1234567890a-zA-Z]`
- `CS[0-9az-A-Z]`
- `CS[0-9][a-zA-Z][a-zA-Z]`
- `CS\d`
- `CS\d\w\w`
- `CS\d[a-zA-Z][a-zA-Z]`
- None of the above

6.14 This exercise provides practice with creating regular expressions using single character wildcards. In each exercise you will be asked to create the regular expression that matches a given string. To keep things simple, you don't need to worry about protecting any meta-characters from the shell.

A. For each of the following, write the correct regular expression:

The string **CBC** followed by any character:

____

The string **Class** followed by any 2 characters:

____

The string **cbc** followed by any 2 characters, followed by the string **DII:**

____

B. Remember that the regular expression **`'[tT]ony'`** will match any line with either the character **t** or **T** followed by the characters **ony**. For each of the following, write the correct regular expression. For simplicity's sake you don't need to worry about protecting any meta-characters from the shell.

The string **case** or the string **Case**

____

The string **room** or the string **Room**

____

The string **CLASS**: followed by any one of the following characters **aAeEiI**

____

The string **Key#** followed by any one of the following characters **135**

____

6.15 Remember that the regular expression **`Docket[0-9]`** will match any line with the characters **Docket** followed by any single numeral, and **`Window-[A-Z]`** will match any line with the characters **Window-** followed by any single uppercase alpha character. For each of the following, write the correct regular expression. For simplicity's sake you don't need to worry about protecting any meta-characters from the shell.

The string **Key** followed by any single digit:

____

The string **Locker:** followed by any uppercase or lowercase alphabetic character:

____

The string **ID**- followed by any character that is **not** an uppercase alphabetic character:

____

A number that is a valid college grade. This requires a 2 digit number where the first number is in the range 0-4, followed by a **.**, followed by any single number:

____

6.16 This exercise provides practice with creating regular expressions using single character wildcards. Hint – you will have to use \ or single quotes to protect special characters that have meaning to the shell.

A. Write and execute the command to find all strings that contain the string **Key#** followed by any number in the file **simpleC**. Create a screenshot showing the result.

B. Write and execute the command to find all strings that contain the string **Key#** followed by any two numbers in the file **simpleC**. Create a screenshot showing the result.

C. Write and execute the command to find all strings that contain the string **case** followed by a space, followed by any number in the file **simpleC**. Create a screenshot showing the result.

D. Write and execute the command to find all strings that contain the string **case** or the string **Case**, followed by a space, followed by any number in the file **simpleC**. Create a screenshot showing the result.

E. Write and execute the command to find all strings that contain at least 6 numbers in a row in the file **test**. Create a screenshot showing the result.

6.17 This exercise provides practice interpreting regular expressions that use occurrence modifiers to specify how many times an item must repeat, or a range of numbers. In each exercise you will be asked to choose the regular expression that matches a given string. For simplicity and clarity, the answers will NOT use the \ to protect any shell meta-characters, but they will be used to protect any regular expression meta-characters.

A. Which regular expression(s) will match all the occurrences of the string **CS** followed by *any* 3 characters? Circle the correct answer(s).

- `CS.[3]`
- `CS.{3}`
- `CS.(3)`
- `CS???`
- `CS[...]`
- `CS(...)`
- `CS{...}`
- None of the above

B. Which regular expression(s) will match all the occurrences of the string **CS** followed by *any* 3 numbers? Circle the correct answer(s).

- `CS{[0-9]x3}`
- `CS[0-9{3}]`
- `CS{3[0-9]}`
- `CS[0-9]{3}`
- `CS{[0-9]3}`
- None of the above

C. Which regular expression(s) will match all the occurrences of the string **Weight:** followed by at least 1 but no more than 4 numbers? Circle the correct answer(s).

- `Weight:[0-9]{1|4}`
- `Weight:{1,4}[0-9]`
- `Weight:[0-9](1,4)`
- `Weight:[0-9]{1,4}`
- `Weight:{[0-9]1,4}`
- None of the above

D. Which regular expression(s) will match all the occurrences of the string **Password:** followed by at least 10 but no more than 14 alphanumeric characters? Circle the correct answer(s).

- `Password:*14`
- `Password:{10,14}[a-zA-Z0-9]`
- `Password:[a-zA-Z0-9](10|14)`
- `Password:[a-zA-Z0-9]{10,14}`
- `Password:{[a-zA-Z0-9]10,14}`
- None of the above

6.18 This exercise provides practice interpreting regular expressions that use the **? +** and ***** occurrence modifiers to specify how many times an item must repeat. In each exercise you will be asked to choose the regular expression that matches a given string. For simplicity and clarity, the answers will NOT use the \ to protect any shell meta-characters, but they will be used to protect any regular expression meta-characters if necessary.

A. Which regular expression(s) will optionally match the character **Z**? That is, the pattern will match if there's a **Z** or not, or in technical terms zero or one occurrences. Circle the correct answer(s).

- `Z?`
- `Z+`
- `Z*`
- `?Z`
- `+Z`
- `*Z`
- None of the above

B. Which regular expression(s) will match one or more occurrences of any number? Circle the correct answer(s).

- `[0-9]?`
- `[0-9]+`
- `[0-9]*`
- `?[0-9]`
- `+[0-9]`
- `*[0-9]`
- None of the above

C. Which regular expression(s) will match zero or more occurrences of any number? Circle the correct answer(s).

- `[0-9]?`
- `[0-9]+`
- `[0-9]*`
- `?[0-9]`
- `+[0-9]`
- `*[0-9]`
- None of the above

D. Which regular expression(s) will match all occurrences of the string **CS** optionally followed by the letter **I**? Circle the correct answer(s).

- `CSI`
- `CSI?`
- `CSI+`
- `CSI*`
- `CS?I`
- `CS+I`
- `CS*I`
- None of the above

E. Which regular expression(s) will match all occurrences of the string **tempo** optionally followed by the number **1**? Circle the correct answer(s).

- `tempo`
- `tempo1?`
- `tempo1+`
- `tempo1*`
- `tempo?1`
- `tempo+1`
- `tempo*1`
- None of the above

F. Which regular expression(s) will match all occurrences of the string **CS** followed by at least one number, but possibly many numbers? Circle the correct answer(s).

- `CS`
- `CS[0-9]?`
- `CS[0-9]+`
- `CS[0-9]*`
- `CS? [0-9]`
- `CS+[0-9]`
- `CS*[0-9]`
- None of the above

G. Which regular expression(s) will match all occurrences of the string **chorus** followed by at least one character from the set **A**, **B**, **C**, or **D**; but possibly many occurrences of **A**, **B**, **C**, or **D**? Circle the correct answer(s).

- `chorus[A-D]`
- `chorus[A-D]?`
- `chorus[A-D]+`
- `chorus[A-D]*`
- `chorus?[A-D]`
- `chorus+[A-D]`
- `chorus*[A-D]`
- None of the above

H. Which regular expression(s) will match all occurrences of the string **CSIA** followed by zero or more numbers? Circle the correct answer(s).
- `CSIA`
- `CSIA[0-9]?`
- `CSIA[0-9]+`
- `CSIA[0-9]*`
- `CSIA? [0-9]`
- `CSIA+[0-9]`
- `CSIA*[0-9]`
- None of the above

I. Which regular expression(s) will match all occurrences of the string **CS** followed by *any* 3 numbers, optionally followed by the letter **L**? Circle the correct answer(s).
- `CS[0-9]{3}L?`
- `CS[0-9]{3}?L`
- `CS[0-9]{3}L+`
- `CS[0-9]{3}+L`
- `CS[0-9]{3}L*`
- `CS[0-9]{3}*L`
- None of the above

J. Which regular expression(s) will match all occurrences of the string **Weight:** followed by at least 1 but no more than 4 numbers, optionally followed by the letter **G**? Circle the correct answer(s).
- `Weight:[0-9]{1,4}G?`
- `Weight:[0-9]{1,4}?G`
- `Weight:[0-9]{1,4}G+`
- `Weight:[0-9]{1,4}+G`
- `Weight:[0-9]{1,4}G*`
- `Weight:[0-9]{1,4}*G`
- None of the above

K. Which regular expression(s) will match one or more numbers followed by the **:** character? Circle the correct answer(s).
- `[0-9]?:`
- `[0-9]+:`
- `[0-9]*:`
- `?[0-9]:`
- `+[0-9]:`
- `*[0-9]:`
- None of the above

L. Which regular expression(s) will match zero or more occurrences of any character, followed by the **:** character, followed by three numbers? Circle the correct answer(s).
   - `.?:[0-9]{3}`
   - `.+:[0-9]{3}`
   - `.*:[0-9]{3}`
   - `?.:[0-9]{3}`
   - `+.:[0-9]{3}`
   - `*.:[0-9]{3}`
   - None of the above

6.19 This exercise provides practice with regular expressions using occurrence modifiers. Modify each of the following regular expressions to use `{n}` to specify the number of occurrences of a character or set of characters. For example `aaa` would be replaced by `a{3}`. For simplicity's sake, don't worry about protecting any shell meta-characters.

`99999` ______________________

`777-7777` ______________________

`[a-z][a-z]--[a-z][a-z]`

____

`[a-zA-Z][a-zA-Z][a-zA-Z]`

____

`[a-z][a-z]:[0-9][0-9][0-9][0-9]`

____

6.20 This exercise provides practice creating regular expressions using occurrence modifiers. For each of the following, write the correct regular expression. For simplicity's sake, don't worry about protecting any meta-characters from the shell. You must however protect any regular expression meta-characters if necessary.

A. The string **case:** followed by 4 numbers:

____

B. The string **case:** followed by at least 4 but no more than 7 numbers:

____

C. The string **CS** followed by a 1 or 2, followed by 2 numbers. (Note that this will result in numbers in the range 100-299):

____

D. The string **CSIA** followed by numbers in the range 100-499:

____

E. The string **Name:** followed by at least 2 but no more than 10 uppercase letters. For example **Name:JACINDA**:

____

F. The string **Name:** followed by at least 2 but no more than 10 lowercase letters. For example **Name:jacque**:

____

G. The string **Name:** followed by 1 uppercase letter, followed by at least 1 but no more than 9 lowercase letters. For example **Name:Jorge**:

____

H. The string **Station**, followed by three numbers, possibly followed by the letter **A**. For example, **Station123A** or **Station456**. Hint – use the **?** occurrence modifier for the **A**.

____

I. A simple phone number that contains a dash. That is, the numbers will be of the form *xxx-xxxx* where *x* is a number.

____

J. A simple phone number that either contains a dash or not. That is, the numbers will be of the form xxxxxxx or *xxx-xxxx* where *x* is a number. Hint – use the **?** occurrence modifier for the **-**.

____

6.21 This exercise provides practice creating regular expressions using the OR occurrence modifier. For each of the following, write the correct regular expression. For simplicity's sake, don't worry about protecting any meta-characters from the shell. You must however protect any regular expression meta-characters.

A. The string **case OR** the string **CASE**:

____

B. The string **moose OR** the string **meeses**:

____

C. The string **CBC OR** the string **Columbia Basin College**:

____

D. The string **CS** followed by three numbers, **OR** the string **CSIA** with no trailing numbers. For example, **CS332** or **CSIA**. Hint – you'll need to use parentheses.

____

E. The string **case OR** the string **CASE** followed by three numbers. For example, **case332** or **CASE229**.

____

F. The string **CS** followed by three numbers, **OR** the string **CSIA** followed by three numbers. For example, **CS332** or **CSIA429**.

____

6.22 This exercise provides practice with regular expressions using positional modifiers. For each of the following, write the correct regular expression:

A. The string **CBC**" at the beginning of a line:

____

B. The string **CBC** at the end of a line:

____

C. The string **CBC** with whitespace before and after the string. Hint – use quotes to show the whitespace.

____

D. The string **CBC** as a separate word:

____

E. The string **edu** as a separate word:

____

F. A simple phone number at the end of a line. You can assume the phone number is of the form *`xxx-xxxx`* where *`x`* is a number:

____

G. A simple phone number with whitespace before and after the phone number:

____

6.23 This exercise provides practice using regular expressions to select items from structured data.

A. Assume that you are working with a data file that has the following structure.

```
firstName : lastName : title : email : cellPhone :
homePhone
```

Write the regular expression to find lines where the **title** field is **`Padawan`**. For simplicity's sake don't worry about protecting any meta-characters from the shell.

____

B. Assume that you are working with a data file that has the following structure.

```
   firstName : lastName : title : email : cellPhone :
homePhone
```

Write the regular expression to find lines where the **cellphone** field contains **`267`**. For simplicity's sake don't worry about protecting any meta-characters from the shell.

____

6.24 The purpose of this exercise is to provide you with practice building complex regular expressions by starting with simple patterns and extending them one step at a time Note that this exercise is fairly involved and will not be graded. I strongly suggest you try it, but at this point you really just need to have an idea of what regular expressions are and how to match simple patterns. You're not expected to write more complicated regular expressions like the one in this exercise. Assume that you are working with IPv4 addresses. Remember that these addresses have the structure *xxx.xxx.xxx.xxx* where *xxx* is a number between 0-255.

A. Write the regular expression to find any single number between **0** and **255**. You're not writing the full IP address yet, just one of the numbers between 0 – 255. At first glance this might seem simple. You might think to use `\d{3}` but this will match any number between **0** and **999.** To match **0** and **255** is going to require writing a few different patterns and OR'ing them together. That is, you'll need to match 0-199, 200-249 and 250-255. To match these ranges, write the following patterns:

- Start by writing the regular expression for numbers that match the pattern 000-199, or a pattern where the first digit is a 0 or 1 followed by any two numbers.

  ____

- Next, write the regular expression for numbers that match the pattern 200-249. This requires writing a regular expression where the first digit is a 2, the second digit is any number between 0 and 4, followed by any number.

  ____

- Matching numbers in the last range 250-255 requires writing a regular expression where the first digit is a 2, the second digit is 5, and the last digit is any number between 0 and 5.

  ____

B. Now that you have the regular expressions to match numbers in the three ranges, use the `|` which is used for OR operations to connect all three regular expressions. Remember to add parentheses around each of the regular expressions to ensure the OR selects between the entire pattern. Generally speaking you should end up with something like: `(regex1) | (regex2) | (regex3)`

   Write the full regular expression in the space below.

   ____

C. At this point you've built a regular expression to match any number between 000 and 255, which as you can see is a little more complicated than **`\d{3}`**. Now you need to write the regular expression to check for four of these numbers separated by dots. In other words, if we represent the regular expression you write in step B as **B** you would need to match **B.B.B.B**

- Start by writing the regular expression to literally match **B.B.B.B** That is, match a capital **B** followed by a . followed by a capital **B** etc.

  There are two reasons for breaking this out in a separate step. The first is because the . character is a meta character and has special meaning unless you protect it. If you wrote the regular expression as **B.B.B.B** you'll need to add a \ in front of each dot.

  ____

  The second is so that you can see that this can be represented as one pattern, which is B followed by a dot repeated 3 times, ending with a B not followed by a dot. Rewrite the regular expression this way if you didn't already. Note using this form may not seem like it's much simpler at this point, but it will in the next step.

  ____

- Finish by substituting the regular expression you wrote in step B for each occurrence of the letter B in the regular expression you just finished.

  ____

As complicated as this may seem, it's actually not completely accurate. That is, it allows for numbers like 000, 001, etc. where we probably don't want to match 000 and we should match 1 as well as 001, or things like 22 as well as 022. To accomplish this you'd need something like the following:

```
((25[0-5]|2[0-4][0-9]|[1]?[1-9][0-9]?).){3}(25[0-5]|2[0-
4][0-9]|[1]?[1-9]?[0-9])
```

And if you wanted to allow both 001 and 1, you'd have to OR this with what you've built, which would be a real mouthful.

Remember the point of this exercise isn't for you to be able to build super complicated regular expressions, it's to demonstrate the process for building regular expressions one small piece at a time.

6.25 The purpose of this exercise is to demonstrate that it may be simpler to find a regular expression on the Internet than it is to create your own. Search the Internet and find at least one regular expression for matching IPv4 network addresses. Make a screenshot of the web page showing the regular expression.

Compare your experience from the previous exercise, where you wrote your own regular expression, with the amount of effort require for this exercise to answer the following question. Which required less effort:

A. Writing the regular expression to match an IPv4 network address

B. Finding the regular expression to match an IPv4 network address on the Internet

6.26 This exercise has been designed to help you apply what you've learned about the `grep` command and regular expressions. In this exercise you will pretend you're working as a network administrator at a business. All web requests from the internal network are initially passed to a proxy server, which will forward the request to the Internet if it doesn't already have the pages cached. Every web request is logged by the proxy server, which means it's possible to discover all the web pages an employee has visited. The last 30 weeks of logs are saved in files named **proxy1.log**, **proxy2.log** … **proxy30.log**. You're contacted by the company's Chief Security Office (CSO) who needs to determine if any employee has visited a website run by the Bene Gesserit. The Bene Gesserit use the domain name **MissionariaProtectiva.bg.org**. You're tasked with checking the proxy server logs for clues that any user may have been visited the Bene Gesserit web site. If you find that any users have visited this site, you have been asked to check to see what other sites these users have visited. This requires looking at the log files which are located in the directory **/home/proxyLogs**.

A. Use what you've learned about `grep` and regular expressions to search the log files for any requests for pages from the web server at **MissionariaProtectiva.bg.org**. (If you have nothing better to do with your time you could manually search through the files, but let me warn you this will take a lot of time.)

   You should find a single entry in the log file that matches this URL. Each entry in the file logs the following:

   - Date and time of the request
   - The name of the service creating the entry. In these log examples almost all of the entries are created by the W3Svc
   - The IP Address of the device/computer where the request originated
   - The HTTP Command issued, typically GET
   - The name of the page or resource requested

- The request port, typically 80 for HTTP and 443 for HTTPS
- The IP Address of the web server
- The browser identifier
- The full URL of the request
- Additional information regarding the request

Inspect the log entry and find the IP Address of the user that made the request and record it in the space below. This will be the first IP Address in the log entry. For example in the following log entry the IP Address is **111.39.21.66**

```
2005-11-25 08:04:42 W3SVC107714930 111.39.21.66 GET
/home/index.asp page=4 80 - 68.231.144.32 Mozilla/4.0+
(compatible; +MSIE+6.0; +Windows+NT+5.1; +SV1)
HTTP://www.go.org/home/index.asp 200 0 0 45160 446 625
```

B. Next, search the logs for any occurrences of the IP Address you found in the previous step. Hint – you'll have to protect the `.` characters using `\.` and use quotes to include the space character at the end of the IP address. This will return a few entries. Inspect the log entries and find the DNS name in the URLs that the user requested and record them in the spaces below. For example in the following log entry the DNS name in the URL is **www.go.org**

```
2005-11-25 08:04:42 W3SVC107714930 111.39.21.66 GET
/home/index.asp page=4 80 - 68.231.144.32 Mozilla/4.0+
(compatible; +MSIE+6.0; +Windows+NT+5.1; +SV1)
HTTP://www.go.org/home/index.asp 200 0 0 45160 446 625
```

6.27 This exercise has been designed to help you apply what you've learned about the `grep` command and regular expressions. In this exercise you will once again assume you're working as a network administrator at a business. In this case you have been asked to provide the URL for any request made between 8 and 9 AM on 11/25/2021. Note that in the log files the format for the date and time are `yyyy-mm-dd hh:mm:ss` Use what you've learned about `grep` and regular expressions to search the log files for any requests made during this time period and record the URL(s) below.

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. What is a regular expression?

2. Assume you wanted to find all occurrences of the string **CBC** inside files ending in **.htm**. Would you use `grep` and a regular expression, or would you use `ls` and a filename wildcard?

3. True/False. The `*` metacharacter means the same thing in a regular expression as it does in a file name wildcard.

4. What do following metacharacters represent in a regular expression?
   A. .
   B. *
   C. ?
   D. ^
   E. $

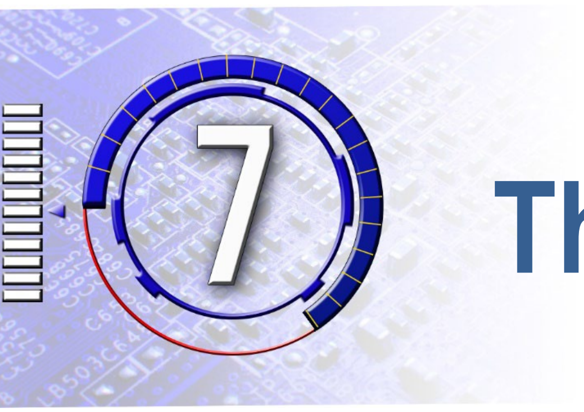
5. What is the pattern that will match any number of any characters in a regular expression?

6. What is the pattern that will match any single number in a regular expression?

7. What is the pattern that will match at least 2 but no more than 5 numbers in a regular expression?

8. What would the command be to find all accounts that start with **cs** in the /etc/passwd file?

9. True or False. It's always better to write your own regular expressions than it is to use one found on the Internet.



The Shell

The exercises in this chapter have been designed to reinforce your knowledge of the Linux/UNIX shell and provide hands on experience using features of the tcsh shell so you can do the following:

1. Define what a shell is and what it does for a user.
2. Describe the major functions provided by any Linux/UNIX shell.
3. Change shells on a temporary or permanent basis.
4. Display and use the shell path, describe what it's used for, and update it after adding new executable files.
5. Change prompts, create aliases, etc.
6. Use shell startup and shut down files to make changes to the shell interface appear permanent.
7. Use commands from the shell history list, and save the list when logging out.

7.1 Find out which shells are available on this computer by looking in the file /etc/shells. List the contents of this file along with the common name for each shell program. For example, the common name for /bin/sh is the Bourne Shell. You may have to do a little Internet research to find out the common names. Use the information you find to complete the following list:

Shell Program	Common Name

7.2 Check each of the shells you listed in the previous exercise and determine whether they are an actual program or a shortcut. If the shell is a shortcut, list which program it points to.

Shell Program	Actual Executable

7.3 Which shell are you currently running? Run the `ps` command and write the results below, or create a screenshot showing the results of the `ps` command.

7.4 This exercise provides practice temporarily running a different shell.

A. Run the `ps` command and verify which shell you are currently using.

B. If you're currently running the tcsh shell, start the bash shell by typing either one of the following:

```
/bin/bash
bash
```

If you're currently running the bash shell, start the tchs shell by typing either one of the following:

```
/bin/tcsh
tcsh
```

C. Use the `ps` command to verify that the new shell is running. Create a screen shot showing the results of the `ps` command.

D. Answer the following questions in the space provided.

- Is your first shell process still running?

  ____

- Which shell is currently handling your commands and input?

  ____

E. Close the shell you just started by typing:

```
exit.
```

Run the `ps` command to verify that the 2nd shell you started has now stopped, and that your first shell process is still running. Create a screenshot showing the results of the `ps` command.

7.5 This exercise provides practice changing your default shell.

A. Use the `grep` command to find your entry in the **/etc/passwd** file by typing:

```
grep username /etc/passwd
```

where *username* is your username. Look at the last field, which holds your default shell. Write the full path to your default shell in the space provided.

____

B. Now run the `chsh` program and change your default shell to one of the shells supported by your system. If you're currently running **/bin/tcsh** set it to **/bin/bash**. Make sure and follow all the warning notes that were provided regarding setting your new default shell. If you do manage to somehow mess this up, and can't login afterwards, make sure to let the instructor know what happened so your account can be fixed.

C. Run `grep` again to verify that your entry in the **/etc/passwd** file has changed. If you don't see anything in the default shell field, then you probably changed the shell to a shortcut. If this is the case do not log out until you have corrected the problem. Create a screenshot showing the results of steps A, B and C.

D. The practical test to verify the change to your default shell is by logging out and logging back in. Do this, and then run the `ps` command to see which shell you are using. Verify that it has been changed, most likely to `bash`.

E. When you are finished make sure that you run `chsh` again to set your default shell back to `/bin/tcsh`. You will need to be in /bin/tcsh for the rest of the exercises in this section. Create a screenshot showing steps D and E.

7.6 This exercise will help you determine which commands are built into the shell and which are external, standalone programs. The main task in this assignment is to complete the last column in the table, indicating whether or not a command is built into the shell or not. Oh, and this may seem a little tricky, but some commands are built into the shell as well as existing outside the shell as a separate program. At this point in your career it's not crucial that you know exactly which Linux/UNIX commands are built-in, but it is a good idea to have some general idea of what's included with the shell code. This will be more important if your career involves working with the Internet of Things (IoT) or programming smaller devices where storage and memory may be limited.

The following table shows most of the commands that are built into the Windows Command Shell. These include Windows commands that are commonly used. The table also shows the Linux/UNIX equivalents to the Windows commands.

DOS	**Linux**	**Function**	**Shell Builtin?**
`DIR`	`ls`	Show directory listing	Y / N
`CD, CHDIR`	`cd`	Changes directory	Y / N
`COPY`	`cp`	Copies file and directories	Y / N
`RENAME`	`mv`	Renames a file or directory	Y / N
`DEL`	`rm`	Deletes files	Y / N
`MKDIR`	`mkdir`	Creates a new directory	Y / N
`RMDIR`	`rmdir`	Removes an empty directory	Y / N
`TYPE`	`cat`	Displays the content of a file	Y / N
`CLS`	`clear`	Clears the screen.	Y / N
`DATE`	`date`	Displays the date	Y / N
`SET`	`set`	Sets or displays the value of an environment variable (DOS) or shell variable (UNIX)	Y / N

There are two or three ways to determine whether a command is built into the Linux/UNIX shell or not. Use any of these methods to determine whether or not the Linux/UNIX commands are built-in to the shell. And use what you discover to complete the table by circling the appropriate response in the last column.

1. Use the `which` command. The syntax for using this is to type:

   ```
   which commandName
   ```

   where *commandName* is the command you're checking on. The `which` command will return something similar to:

`commandName: shell built-in command.`	If the command is built-in.
`/path/commandName`	The path to the command, which means it is not built-in
`commandName aliased to`	The command is an alias. (See below for details on aliases.)

   (Note that the `which` command itself is specific to the tcsh/csh shells. If you're running bash/sh then you should switch to tcsh/csh. Or, you can use `type -a commandName` command which performs a similar function in bash/sh.)

2. Do an Internet search for something like **tcsh built in commands**. There are several web sites that list all of the built-in commands, along with explanations for their use.

3. Use the `man` command to read the manual page for tcsh. If you choose to do this here's a warning. The man page for tcsh, or any shell, is going to be quite long as it contains all the pertinent information about the shell. In other words, the man page will hold a lot more than a concise list of commands, and you'll have to do a lot of reading to find the built-in commands.

7.7 This exercise provides practice creating and removing simple aliases.

A. Create the following alias. Make a screenshot after the last step.

```
alias  showPass  more /etc/passwd
```

Verify that the alias has been created by typing:

```
alias
```

Test the alias by typing:

```
showPass
```

B. Create the following alias. Make a screenshot after the last step.

```
alias  showHome ‘cd ~ ; ls –al’
```

Verify that the alias has been created by typing:

```
alias
```

Test the alias by typing:

```
showHome
```

C. Create aliases for the commands shown in the following table. After creating the last one, run the `alias` command, then create a screenshot.

Alias	**Command**
`Copy`	`cp`
`Del`	`rm –i`
`Print`	`cat`

D. Remove the aliases you created in steps A and B. Then run the `alias` command again to verify that the aliases have been removed. Create a screenshot showing your work.

7.8 This exercise is optional as it covers a relatively advanced topic. This provides practice placing arguments inside an alias comprised of multiple commands.

A. Inspect the following alias. As written the alias will show you a detailed listing for all the processes associated with your username. Or to be more technically correct, it will do this for the username stored in the **$user** variable.

```
alias what  ’ps -aux | grep $user | more’
```

B. Use what you've learned about passing arguments to modify the alias. Modify it so that you can pass any username into the `grep` portion of the alias code. That is, you should be able to type

```
what  userName
```

and have the alias execute

```
ps -aux | grep userName | more
```

Create a screenshot showing the alias in your alias list, and the result of running the command for at least two different users.

7.9 This exercise provides practice overriding an `alias`.

A. Create the following alias:

```
alias cat echo Cat needz nap. Ask dog to showz you
```

B. Test the alias by typing:

```
cat /etc/passwd
```

Does the shell run your alias or the real `cat` command? Circle the correct answer.

C. Add a whack to the start of the alias to tell the shell to override the alias:

```
\cat  /etc/passwd
```

Does the shell run your alias or the default `cat` command? Create a screenshot showing the result.

D. You should probably unalias the cat alias you just made. Or get a cat that isn't so lazy. LOL.

7.10 This exercise provides practice adding commands to your **.login** file and verifying that the commands will execute.

A. Ensure that you're in your home directory.

B. Open the file **.login** in the editor. Don't worry if the file doesn't exist already; you'll be creating it with this process.

```
vi .login
```

As an alternative you can edit the file on the PC and use FTP/SCP to move it to your home directory.

C. Enter the following commands

```
echo 'Hello gorgeous.'
echo 'The best day of your life, so far, is'
echo ' going to be:'
date
```

Note that the quotes should all be ticks or single quotes. The formatting in the Word processor may make the end quote on each line look like a back tick.

D. Save the file.

E. Use the `ls -al` command to ensure that you have read and execute permissions on your .login file. If necessary, use the `chmod` command to give yourself the permissions.

F. Test the changes by logging out, and logging back in. Verify that file executes automatically when you login and that you see the text from the `echo` commands followed by the current date and time. Create a screenshot showing the result.

G. If you don't want or care to see this information every time you login, edit your **.login** file and remove the commands.

7.11 This exercise provides practice adding an `alias` command to your **.login** file, using the source command to execute the .login file, and verifying that the alias is created.

A. Add the following command to your .login file:

```
alias  duplicate  cp
```

B. Execute the .login file by running the `source` command.

```
source .login
```

C. Verify that the alias was created by displaying your list of aliases. Create a screenshot showing this list.

7.12 This exercise provides practice displaying the values for shell variables in the tchs/csh shell.

A. Display the current values for all of the shell variables by typing:

```
set | more
```

There are a few of these variables that you've learned about previously, such as **ignoreeof**. You'll learn about many of the others later in this section. Create a screenshot showing the last screen of output from the `set` command.

B. Display the current value for the **$cwd** shell variable, which holds the value of the current directory. This can be displayed by typing:

```
echo $cwd
```

Run the `pwd` command, which displays the path of the directory you're currently located in. Verify that the path displayed by `pwd` matches the value of the `$cwd` variable?

C. Use the `cd` command to move to the **/var/log/anaconda** directory. Once again display the value of the $cwd variable. Verify that it changed as you expected. Create a screenshot showing the value of the $cwd variable or write the value of the variable in the space provided.

____

7.13 This exercise provides practice displaying the values for shell variables in the tcsh/csh shell.

A. Assume that you want to display the value stored in the **ignoreeof** shell variable. Which of the following commands would accomplish this? Circle the correct answer.

- `echo ignoreeof`
- `echo $ignoreeof`
- `echo \$ignoreeof`
- None of the above

B. Assume that you want to display the value stored in the **prompt** shell variable. Which of the following commands would accomplish this? Circle the correct answer.

- `echo prompt`
- `echo $prompt`
- `echo \$prompt`
- None of the above

7.14 This exercise provides practice creating new custom shell variables in the tcsh/csh shell.

A. Assume that you want to store the value **mullet** in the **haircut** shell variable. Which of the following commands would accomplish this? Circle the correct answer.

- `haircut=mullet`
- `$haircut=mullet`
- `set haircut=mullet`
- `set $haircut=mullet`
- `set haircut = mullet`
- `set $haircut = mullet`
- None of the above

B. Assume that you want to store the value **awkward** in the **emoji** shell variable. Which of the following commands would accomplish this? Circle the correct answer.

- `emoji=awkward`
- `$emoji=awkward`
- `set emoji=awkward`
- `set $emoji=awkward`
- `set emoji = awkward`
- `set $emoji = awkward`
- None of the above

C. Assume that you want to create a shell variable named **contagious** to use as a flag or switch, and you want to set it to on. Which of the following commands would accomplish this? Circle the correct answer.

- `contagious=on`
- `$contagious=on`
- `set contagious`
- `set $contagious`
- `set contagious=on`
- `set $contagious=on`
- None of the above

7.15 This exercise provides practice removing or unsetting shell variables in the tcsh/csh shell.

A. Assume that you want to remove the **haircut** shell variable. Which of the following commands would accomplish this? Circle the correct answer.

- `haircut=””`
- `$haircut=””`
- `set haircut=””`
- `set $haircut=””`
- `unset haircut=mullet`
- `unset $haircut=mullet`
- `unset haircut`
- `unset $haircut`
- None of the above

B. Assume that you want to remove the **emoji** shell variable. Which of the following commands would accomplish this? Circle the correct answer.

- `emoji=””`
- `$emoji=””`
- `set emoji=””`
- `set $emoji=””`
- `unset emoji=awkward`
- `unset $emoji=awkward`
- `unset emoji`
- `unset $emoji`
- None of the above

C. Assume that you want to remove a shell variable named **contagious**, which was being used as a flag or switch. Which of the following commands would accomplish this? Circle the correct answer.

- `contagious=off`
- `$contagious=off`
- `unset contagious`
- `unset $contagious`
- `set contagious=off`
- `set $contagious=off`
- None of the above

7.16 This exercise provides experience finding commands located in the directories specified in the `$path` variable.

A. Start by displaying the directories stored in the `$path` variable by typing:

```
echo $path
```

B. Run the `ls` command on each of the directories in your `$path`. This is just to give you an idea of the different commands available. If I were a good teacher, I'd make you memorize every command and how to use them. LOL. For this portion of the exercise I'd simply like you to look at all of the commands that are available, and maybe appreciate all of the work that has been done over the years to create all these commands.

C. The following table contains a list of commonly used Linux/UNIX commands. For each of the commands in the following table, use the `whereis` command to locate the executable for the command. Complete the second column in the table by filling in the full path to each command. The first one has been done for you as an example.

Command	**Location**	**Command or Full Path**
`ls`	`/bin/ls`	[✓ Command] Full Path
`clear`		Command Full Path
`chfn`		Command Full Path
`ps`		Command Full Path
`echo`		Command Full Path
`ifcfg`		Command Full Path

D. Test running each of the commands in Step C. by typing the command name instead of typing the full path to the executable file. For example, just type **`ls`** instead of typing **`/bin/ls`**. If the command doesn't execute by typing just the command name, try to run it by typing the full path. Determine whether you must provide the full path to the command, or if can you just type the command name. Use your findings to complete the third column of the table by circling **Command** if you can type the command name, or **Full Path** if you have to supply the full path.

7.17 This exercise provides practice changing your `$path` and using rehash to update the table after adding new executables. You will be required to provide answers and create screenshots at different points throughout the exercise.

A. Ensure that you are in your home directory

B. Copy the directory **/home/pathTest** to your home directory by using:

```
cp -r /home/pathTest  .
```

C. Use the `cd` command to move into pathTest. List the files in the new directory using `ls`. Verify that you see the files `hello.sh` and `iq.sh`. If necessary, use the chmod command to ensure that you have execute permission to all the files. You should also `cat` or `more` the files to familiarize yourself with the content, and get an idea of what will happen if the file is executed.

D. Try and execute the file **hello.sh** by just typing the command name. Do **not** add the `./` to specify the relative path. Write the message displayed when you run the command in the space provided.

____

E. Modify your path and add this directory by typing:

```
set path=($path  ~/pathTest)
```

F. Now that you have successfully updated your path, try and execute the file **hello.sh** again by just typing the command name. Do **not** add the `./` to specify the relative path. Write the message displayed when you run the command in the space provided.

____

Use the information you've gathered to answer the following question. Circle the correct answer

True / False. Adding a directory to your `$path` causes the shell to update hash table?

G. Next you will test to see if the hash table is updated when you add a new file to one of the directories in your `$path`. Copy **`newfile.sh`** to **`new2.sh`** by typing

```
cp newfile.sh new2.sh
```

H. Try and run **new2.sh** by typing: `new2.sh`

Write the message displayed when you run the command in the space provided.

____

I. Use the information you've gathered to answer the following question. Circle the correct answer

   True / False. The hash table automatically updates when new commands are added to a directory in `$path`.

J. Now update the hash table by typing `rehash` Try to run the file **new2.sh** again by typing `new2.sh`

   Create a screenshot showing your work.

7.18 This exercise provides practice adding the current directory your search path. Although it's highly discouraged for security reasons, you can add the current directory to the end of your path. You can actually add it to any part of the path, the start the middle or the end, but again for security purposes if you feel you must add it you should add it to the end of the path. Searching the current directory works differently than other directories, since the current directory must be searched every time you type a command instead of adding the commands to the path hash table. To test this, you're going to try and run a command by specifying the command name, both with and without having the current directory in your `$path`. You will be directed to create screenshots are various points in the exercise.

A. Ensure that you are in your home directory. Also ensure that the current directory is not in your `$path` at this time.

B. Copy the file `dotpath.sh` from the directory `/home/pathTest` to your home directory, renaming the copy to **`dotpath2.sh`**, by using:

```
cp pathTest/dotpath.sh ~/dotpath2.sh
```

C. Ensure that you have execute permission on **dotpath2.sh**. Try and execute the file **dotpath2.sh** again by just typing the command name. Do **not** add the `./` to specify the relative path. Create a screenshot or write the message displayed when you run the command in the space provided.

   ____

D. Add the current directory to `$path` and then update the hash table by using the following two commands:

```
set path=($path  .)
rehash
```

E. Try and execute the file **dotpath2.sh** again by just typing the command name. Do **not** add the `./` to specify the relative path. Create a screenshot showing your work.

F. When you're done with the exercise and any testing you might want to do you should reset your $path and remove the . for the current directory. The easiest way to accomplish this is to log out and log back in again.

7.19 [OPTIONAL] This exercise will help you determine the order that the shell uses when searching for commands. Note – this exercise takes you through several steps and forces you to look at the concepts of executing aliases, built-in commands, and commands in your $path. The exercise goes into a level of detail that may prove confusing if you're still learning the basics, so completing this exercise is optional. If you already have some experience with the Linux command line you might want to try the exercise because like learning anything, I think you'll remember the order better if you do a little investigative work yourself. Even if you're new to Linux, you might try a few of the steps and see if it makes sense. In any case the answer for this problem is given at the end of the exercise.

You'll accomplish this by performing a series of tests, where you create commands or aliases using the same name as existing commands like `cd` or `ls`, and then seeing which of those commands executes when you type the command name. To check each of the 4 items against all three other items would require six tests in total. I've listed all six tests below so you can see that each item is checked against the 3 other items.

Alias vs. built-in
Alias vs. $path
Alias vs. specified path
Built-in vs. $path
Built-in vs. specified path
*Specified path vs. $path

There are tests you can do to check the first five items, but it's impossible to check the last one, which is marked with an asterisk *. The reason this can't be checked is that to perform the test you'd need to create a file with a `/` or whack in the file name, which is not possible to do. You can make an alias with a whack in the alias name, but you can't make a file with a name like `/usr/bin/cd` for reasons that I hope are obvious.

Use the information you glean from the tests to complete the following table by adding the 4 places the shell looks for commands in the right column, in the order that the shell looks in each location. For example, if the shell looks at Aliases first, put it in the first row. And don't worry if you don't want to do any or all the steps, as the results are presented after the end of the exercise.

Shell looks here first	
Shell looks here second	
Shell looks here third	
Shell looks here last	

A. **Alias vs. Built-in**. The first test will check to see if the shell runs an alias before it runs a built-in command, or vice-versa. You'll test this by creating an alias named `cd` , which is also a built-in command.

- Ensure that you're in your home directory.

- Create an alias named `cd` and add the following `echo` command by typing:

```
alias cd echo 'this is my seedy cd alias'
```

- Type the following to tell the shell to run the command:

```
cd /usr
```

- When you do this do you see the directory listing or 'this is my alias'? Note – if it runs the alias first you will see `this is my alias /usr` displayed. (Can you grok why you'll see that message from the alias? You don't need to provide an answer, but from perspective as a teacher it would be great if you take a few seconds and think about it.) If the shell runs the built-in `cd` you won't see the message but you will move to the /usr directory, which can be verified by running the `pwd` command.

- Note – when you're done you'll have to use `unalias` to get rid of your **`cd`** alias, or use `c\d` to override the alias, otherwise you won't be able to change directories.

B. **Alias vs $path**. The next test will check an alias versus finding a file in the hash table for the shell search path. You will create an alias named `ls`, and test to see if the alias executes, or `/usr/bin/ls`, which is the file in the hash table executes.

- Ensure that you are in your home directory

- Create an alias file named **`ls`** using the following:

```
alias ls echo 'alas this is my alias ellis'
```

- Type the following command:

```
ls /usr
```

- If you see "`alas this is my alias ellis /usr`" it means that the shell runs the alias before searching the directories in `$path`. If you see the directory listing for the /usr directory it means that the shell runs the command in the hash table before checking for an alias.

- Note – when you're done with this exercise make sure and use `unalias` to get rid of your **`ls`** alias before it drives you crazy.

C. **Alias vs. specified path**. This test will check an alias named `/usr/bin/cd` against the `/usr/bin/cd` in the hash table for the search path. This is of course a ridiculous name for an alias, but an alias name like this needs to be used to perform the test.

- Create an alias named `/usr/bin/cd` by typing the following command:

```
alias /usr/bin/cd echo 'this is my seedy cd alias'
```

- Try to change to the **/etc** directory using the command.

```
/usr/bin/cd /etc
```

If you see message `this is my seedy cd alias /etc` it's proof that the shell has run your alias first. If you actually change directories, then it's proof that the shell runs the `cd` command specified in the path first.

- When you're finished testing make sure and remove the alias.

D. **Built-in vs. $path**. This test will check whether the shell looks for built-in commands before it looks for commands in directories in the search path. The testing will be done using the `cd` command, which exists as both a built-in command, and as a command in the /usr/bin folder which by default is in the shell $path. To tell which command is being used `/usr/bin/cd` will be replaced with a program that echoes some text. Since `/usr/bin` is one of the directories in your $path issuing a `cd` command will either run the built-in command or the fake `cd`. Note that this test can only be performed if you have root or administrative access to your Linux system as it requires making changes to `/usr/bin/cd`. If you don't have root access the results of the test are shown after the last step.

- Use the `cd` command to move to the **`/usr/bin`** directory.

  ```
  cd /usr/bin
  ```

- Use `ls` to check to verify that the file named `cd` exists in the directory.

- Rename **cd** to **cd-orig** using the `mv` command. This way, you can get the original file back at the end of the exercise.

  ```
  mv cd cd-orig
  ```

- Create file named `cd` and add the following `echo` command to the file:

  ```
  echo 'this is my latest cd’
  ```

- Ensure that you have read and execute permission on the cd file you just created. Also ensure that you do not have an alias that uses `cd`.

- Try and use the `cd` command to change to the **/usr** directory.

  ```
  cd /usr
  ```

  If you see the message `this is my latest cd` then it’s proof the shell looks for and executes files in $path before it runs the built-in command. If you actually change directories then it’s proof that the shell runs the built-in `cd` first.

- Normally when this test finished you should make sure and rename **cd-orig** back to **cd**. However, you’ll be using the modified `cd` command again in the next test, so don’t rename it yet.

- If you don’t have root permissions and can’t complete this test, here are the results. The shell finds and runs the built-in commands before it looks for commands in your $path.

E. **Built-in vs. specified path**. This test will check whether the shell looks for built-in commands before it looks for commands that have been specified with a full path. This test is unnecessary because it’s obvious that if you give the shell a path to a command you’re being very specific about the command you want to run, and the shell will run it, assuming the file exists. This isn’t like the other cases being tested, where you give the shell a file name and it has to figure out if you’re referring to an alias or a built-in command or a file in the $path. However, just in case you want to test this for yourself the testing can be done using the modified `cd` command which you built in the previous step against the built-in `cd` command. However in this case you’ll start the command by typing the full path which is `/usr/bin/cd`. This will either run the built-in command or the fake cd. Note that once again this test can only be performed if you have root or administrative access to your Linux system as it requires making changes to `/usr/bin/cd`. If you don't have root access the results of the test are shown after the last step.

- Ensure that the modified cd command you created in the step still exists, and that you have execute permission. If not, recreate it and/or set the permissions.

- Try and use the `cd` command to change to the **/etc** directory using the full path for the `cd` command.

  ```
  /usr/bin/cd /etc
  ```

  If you see the message `this is my latest cd /etc` then it's proof the shell looks for and executes the file specified by the full path before it runs the built-in command. If you actually change directories then it's proof that the shell runs the built-in `cd` first.

- When you're finished testing make sure and rename **cd-orig** back to **cd**.

- If you don't have root permissions and can't complete this test, here are the results. The shell finds and runs the file specified by the full path before it looks for built-in commands. This only makes sense since you're telling the shell exactly which command to run.

F. **Specified path vs. $path**. This is another case that really doesn't have to be tested as once again it's obvious that if you use a path to specify an exact file to run, the shell will run it and not search $path. And, if you want to test this yourself you're going to be out of luck as it would require making a file with a name that looked like a path. That is, you'd need something like a file named **/usr/bin/cd** where the whacks are part of the file name and not part of the path.

G. Answer – The tests, combined with a little bit of common sense and logic, result in the following priority list. And once again, note that this is only important if you make an alias or create a program that has the same name as an existing command. If you avoid using existing command names for any aliases or programs you create you'll avoid any possible confusion and shouldn't have to worry about this.

Shell looks here first	Alias
Shell looks here second	Built-in
Shell looks here third	Absolute or relative path
Shell looks here last	Commands in $path

7.20 This exercise provides practice repeating commands using the tcsh shell's history.

A. Which of the following could be used to repeat the last command you typed?
- -1!
- $$
- !$
- !!
- !1
- None of the above

B. Which of the following could be used to repeat the last command you typed?
- !-1
- -1!
- $-1
- !$
- !1
- None of the above

C. Which of the following would you use to repeat the last command you typed that starts with the letters **st**?
- $st
- st!
- !!st
- !st
- !1st
- None of the above

D. Assume that the last file you edited in `vi` was **strongLemonSmell**. Which of the following would you use to open this file in editor again?
- $vi
- vi!
- !!vi
- !vi
- !!strong
- None of the above

7.21 This exercise provides practice repeating commands using the tcsh shell's history. Assume that you have just run the `history` command and it displays the following list. Also assume that you are always typing command #12. That is, assume you're always typing the command after command #11 which is the `history` command.

```
1  mkdir practiceDir
2  chmod 700  practiceDir
3  cd practiceDir
4  touch junk1 junk2 junk3
5  ls -al
6  date > junk4
7  cat  /etc/passwd  >> junk4
8  rm junk2
9  ls -al
10 cp ~/.login junk2
11 history
```

What command will be executed when you type each of the following? Write in the command number in the space provided.

```
A. !9      _______________
B. !-4     _______________
C. !!      _______________
D. !h      _______________
E. !c      _______________
F. !ca     _______________
G. !?login _______________
```

7.22 This exercise provides practice modifying a command from the history list by adding more to the command line after the expanded command. In this exercise you will build a command pipeline that will display the default shell used by the last 20 student accounts in the **/etc/passwd** file. You will create a screenshot showing your work after the last step.

A. Start the pipeline by using the `grep` command to find all of the student accounts in the **/etc/passwd** file by typing:

```
grep student /etc/passwd
```

B. Repeat the previous command but add the items necessary to pipe the output from the previous command through the `tail` command, displaying the last 20 lines.

C. Repeat the last command and add the items necessary to pipe the output from the previous command through the `cut` command. Use the `cut` command to display fields 1 and 7. Create a screenshot showing the results of this final command pipeline.

7.23 [OPTIONAL] This exercise provides practice retrieving part of a command from the shell command history list. Assume that you have the following commands in your history list.

```
1  mkdir practiceDir
2  chmod 700  practiceDir
3  cd practiceDir
4  touch junk1 junk2 junk3
5  ls -al
6  date > junk4
7  cat  /etc/passwd  >> junk4
8  rm junk2
9  ls -al
10 cp ~/.login junk2
11 history
```

For each of the following, write down what the shell would execute after the history substitution(s) occur:

A. `!6:0` ______________
B. `cat !8:1` ______________
C. `!9:0` ______________
D. `rm !mk:$` ______________
E. `sort !7:$` ______________
F. `ls -al !t:2` ______________

7.24 This exercise provides practice using `!$` to retrieve part of the previous command.

A. Assume that the last command you typed was:

```
mkdir practiceDir
```

Which of the following would you use to change the permissions on **practiceDir** to **700**? Circle the correct answer:

a. `chmod 700 !!`
b. `chmod 700 !*`
c. `chmod 700 !!2`
d. `chmod 700 !$`
e. None of the above

B. Assume that the last command you typed was:

```
mkdir practiceDir/homework
```

Which of the following would you use to move into the **practiceDir/homework** directory? Circle the correct answer:

a. `cd !!`
b. `cd !*`
c. `cd !!2`
d. `cd !$`
e. None of the above

C. Assume that the last command you typed was:

```
chmod 700 downloadAntivirusDefinitions
```

In the space provided write the command you would use to execute the command without typing **downloadAntivirusDefinitions**.

____
____

7.25 This exercise provides practice editing a command from the shell's history list. Assume that you have the following commands in your history list.

```
1  mkdir practiceDir
2  chmod 700  practiceDir
3  cd practiceDir
4  touch junk1 junk2 junk3
5  ls -al
6  date > junk4
7  cat  /etc/passwd  >> junk4
8  rm junk2
9  ls -al
10 cp ~/.login junk2
11 ls junk5
```

For each of the following, write down what the shell would execute after the history substitution(s) occur:

```
^5^2         ______________
^ls^ls -al   ______________
!c:s/2/3/    ______________
!2:s/7/5     ______________
```

7.26 This exercise provides practice saving the shell history list. You will create a screenshot at the end of the exercise to verify your work.

A. Ensure that `$history` has been set. If you're not sure type:

```
echo $history
```

If it has not been set, turn it on and set it to at least 20 commands.

B. Type several commands to populate your command history list. For example, change directories, use the `ls` command, use the `echo` command to write some text to the screen, use the `touch` and `rm` command to create and delete some files, etc. The only command you don't want to type is `exit`.

C. After you've entered at least 20 commands view your history list by typing

```
history
```

D. Tell the shell to save commands when you logout using the following command. You can set the value higher if you wish, just make sure that it's set to 20 as a minimum.

```
set savehist=20
```

E. Logout and log back in. Check the contents of your command history by typing:

```
history
```

It should have several commands in it already, and they should be the commands you entered during your last shell session. Create a screenshot showing your history list.

F. Remember that if you want to make your `$history` and `$savehist` settings permanent you'll have to add the commands to set them to your .login or .cshrc. There's nothing to turn in for this step, it's a just a reminder that you may want to do this.

7.27 Practice setting your prompt using the following prompt variables. Each time after you change your prompt, change directories to at least /, **/usr/lib/**, back to your home directory, and to a sub-directory under your home directory named **c1/c2/c3**. (You will need to create **c1**, then **c2** inside of **c1**, and then create **c3** inside of **c2** the first time around.) Write down what your prompt displays at each of these directories in the space provided or create a screenshot or two showing how your prompt changes as you move around the directory structure.

```
                cd  /    cd /usr/lib   cd or cd~     cd
~/c1/c2/c3
set prompt=%/       .........................
............................
...........................     ......................
set prompt=%~       ..........................
............................
..........................      ......................
set prompt=%C2     .........................
............................
...........................     ......................
```

7.28 In this exercise, you will check to see what order is used for processing your startup files. That is, which is executed first, the system wide login, your **.login**, or your **.cshrc** file.

A. The first test will check whether your **.login** is processed before the system wide login file, or vice versa.

- You will check this by first displaying the aliases created by the system wide login file. You can do this by typing the `alias` command.

- Select one of the aliases such as `ls`. Next you will try and get rid of the alias you selected by adding the `unalias` command to your **.login** file. For example, if you want to get rid of the ls alias add the following to the .login file in your home directory.

```
unalias ls
```

- Save your **.login** file, and then log out of the system. Log back in and recheck your list of aliases. If the `ls` alias is still in the list it means the system wide login file was processed after your .login. That is, your .login removed the alias when it was run, but the system wide login file added it back. If the `ls` alias is not in the list, it's evidence that your .login file runs last. Use the information you've discovered to answer the following question.

  During the login and startup process, which file is processed last?

  a. The system wide login file
  b. Your .login file

B. The next test will check whether your .login is processed before your .cshrc file, or vice versa. You will check this by setting different prompts in each file and then logging in.

- Start by editing your .login file and adding the following command to set your prompt as indicated:

```
        set prompt=”login-last> “
```

- Save your changes. Next, edit your .cshrc and add the following command to set your prompt as indicated:

  ```
  set prompt=”cshrc-last> “
  ```

- Save your changes.

- Log out of the system. Log back in and check your prompt. When you login, one of the files will be processed first and the prompt will be set. Then, the second file will be processed, and the prompt will be changed. When you see the prompt, you will know which file was processed last. Use the information you’ve discovered to answer the following question.

  During the login and startup process, which file is processed last, your .login file or your .cshrc file?

  a. The .login file
  b. The .cshrc file

7.29 This exercise provides practice moving jobs between the background and foreground and stopping commands.

A. Start a simple command that requires more keyboard input, but place it in the background using:

```
cat > deleteme &
```

B. Even though the `cat` command has not finished, the shell will display your prompt signifying it’s ready for you to type another command. Check to see that a job has been created for the `cat` command, and what it's job number is by typing:

```
jobs
```

Create a screenshot showing your work.

C. Move the `cat` command job back to the foreground by typing either:

```
fg
```

or

```
%1
```

The shell will tell you that it's resuming the `cat` command and wait for more input. You can type some text if you wish, or just type `<ctrl-d>` to signal the end of the keyboard input and end the `cat` command.

D. Practice killing a job by starting the `cat` command and placing it in the background again

   ```
   cat > deleteme &
   ```

E. Find the pid for the `cat` command using the `ps` command

   ```
   ps
   ```

   Create a screenshot showing your work.

F. Use the `kill` command to stop the `cat` command. You'll have to use the **pid** associated with the cat process from Step E.

   ```
   kill -9 pid
   ```

G. Run the `ps` command again to verify that the process has been killed.

   ```
   ps
   ```

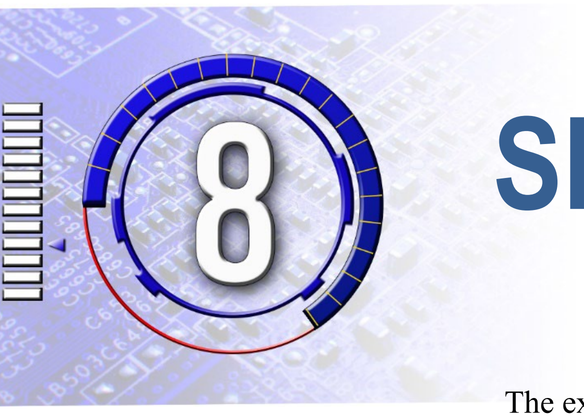
   Create a screenshot showing your work.

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. List some of the functions that the shell provides for you.
2. When you type a command, the shell will look several places for a command with that name. Name these places.
3. In what order will the shell search these places?
4. What does the *rehash* command do for you?
5. When would you use *rehash*?
6. Does everyone on the system have to run the same shell?
7. Why would you want to change your shell?
8. How do you change your shell on a temporary basis? Permanently?
9. Which shells can you change to? Do all UNIX systems have all shells available?
10. How do you find out the built-in commands for a given shell?
11. How can you customize the shell? Can you make these changes so they are always available for you?
12. What are the names of your startup files?
13. What is a shell variable? Why do they exist?
14. Name some shell variables and what they hold?
15. What is an environment variable? How are they different from shell vars?
16. Name some variables that exist in both the shell and the environment.
17. List the steps in the login process? Can the root account set up things to affect all of users?
18. Which will win out, .login or .cshrc?
19. How do you list your currently defined aliases?
20. How do you create an alias?
21. How do you delete an alias?
22. Why would you want to create an alias?
23. How do you turn your history on?
24. How do you see the list of commands that you have already executed?

25. How do you repeat your last command? Name two ways.
26. How do you repeat the last command that starts with the letter `c`?



Shell Scripts

The exercises in this chapter have been designed to reinforce your knowledge of Linux/UNIX shell scripts and provide hands on experience using bash shell scripts so you can do the following:

1. Write and execute simple shell scripts.
2. Add Linux/UNIX commands to shell scripts.
3. Use redirects and pipes in shell scripts.
4. Decipher scripts written by others, and adapt them to meet a set of specified requirements.
5. Utilize basic bash shell programming functions such as writing output, reading data, using variables, performing calculations, making decisions with if statements, and repeating code with loops.

8.1 Assume you are writing a shell script using the bash shell syntax. Which of the following must be the first line in the shell script?

A. The first line must be blank
B. /bin/bash
C. !/bin/bash
D. #/bin/bash
E. !#/bin/bash
F. #!/bin/bash
G. syntax bash
H. None of the above

8.2 Assume you are writing a shell script using the bash shell syntax. Which of the following are true regarding comments?

A. The # character starts a comment.
B. Anything on a line after the # character will be ignored.
C. Comments are started using /* and ended by */
D. Comments are started using # and ended by #
E. Comments must be the only thing on a line
F. None of the above

8.3 Assume you have written a shell script named **wash.sh** and now want to test it. The script is in the directory you are currently working in, and this directory is not in your $path. Which of the following would run the script?

A. wash.sh
B. ~/wash.sh
C. .wash.sh
D. ./wash.sh
E. run wash.sh
F. run ./wash.sh
G. None of the above

8.4 The first program you should write in any language is "hello world". Use `vi` to create a file named hello.sh. Enter the following lines and save the file.

```
#!/bin/sh
echo "Hello World"
```

Change permissions so you can execute the file, then run it by typing:

```
./hello.sh
```

Create a screenshot showing the output from the shell script.

8.5 Each program you write should include comments that describe the program. At a minimum all your programs for this class must include your name and the assignment number in the comments. Add these comments to the program you created in the previous exercise. Use `more` or `cat` to display the file's content and create screenshot.

8.6 The `echo` command can be used to create primitive graphics such as lines or boxes, and to make the output from a script "pretty". Using the `echo` command, write a box containing your name and assignment #. You can use whatever characters you wish to create the box. Remember, if you want to use a character that has special meaning to the shell you will need to protect or escape it using quotes or the "\" character.

```
+--------------------------------+
| Name:  Joe Unix                 |
| Assignment:  x.xx               |
+--------------------------------+
```

Create a screenshot showing the output from the shell script.

8.7 Use the `date` command to add the current date and time to the shell script from the previous exercise. That is, also print the date and time inside the box along with your name and the assignment number. This is a little trickier than it may first appear, as it requires printing the result of the `date` command inside an `echo` command. Remember that the backticks can be used to tell the shell to run a command, and then place the output from that command back on the command line. For example:

```
echo "the current date is: " `date "+%D %T"`
```

Create a screenshot showing the output from the shell script.

8.8 This exercise provides practice using the `echo` command to create ASCII Art. Use the `echo` command and the
`/  \  _` characters to print a triangle that is at least 4 lines in height. Note that when you print the \ character the shell will think you're trying to protect the next character unless you enclose the line of text in single quotes. You can try using double quotes to enclose the characters for your triangle, but it will cause some weird errors. (Yes, this is when all that work with quotes back in Chapter 5 pays off.) Create a screenshot showing the output from the shell script.

8.9 This exercise provides additional practice using the `echo` command to create ASCII Art. Use the `echo` command to print a hawk or eagle. The point of this exercise isn't to have you spend hours trying to print a bird, which would be a super hard script to create unless you're extremely creative. The point of this exercise is to expose you to ASCII art. To accomplish this exercise the suggestion is that you search the Internet for "ASCII Art". You should be able to find something that you can use. Another hint is to find a small bird since you'll probably have to add the `echo` command to the start of every line when you add the art to your script. Create a screenshot showing the script, and another screenshot showing the output from the shell script.

8.10 This exercise provides practice printing tabs. Write a shell script that prints a table of your class schedule for this quarter. Print the class number, the class name, and the time or the word **online** if the class is online. Use tabs to make the columns line up. (Remember that `\t` is the tab character, and that you must use `echo -e` in the Bourne shell to have it expand the tabs.) Create a screenshot showing the output from the shell script.

8.11 This exercise provides additional practice printing tabs. Write a shell script that prints out information from different shell variables. This may be a little tricky as some versions of the bash shell use all capitals for the shell variables. That is, it may be $user or it may be $USER. To check this you can temporarily start the bash shell by typing `bash`, then typing the `set` command. Print two columns, one showing the variable name and the other showing the current value of the variable. Use tabs to separate the two columns. You must print at least the current working directory `$CWD` and the username of the person running the program or `$USER`. Create a screenshot showing the output from the shell script. Hint – use a \ to protect the $ in the variable name. For example:

```
echo –e \$USER \t $USER
```

8.12 This exercise provides practice setting shell variables, and then printing their values. Modify the following shell script, which prints your class schedule for this quarter. Use variables to hold the class names and times. For example:

```
#!/bin/sh

class1=”CS223 Unix”
time1=”8AM - 9AM”
echo -e  “$time1 \t$class1”
```

Create a screenshot showing the output from the shell script.

8.13 This exercise provides practice using the `read` command to gather user input. Modify the following shell script so that it prompts the user for their email address, and then prints out the string they entered. Run the shell script and create a screenshot showing the output. The example shell script can be found in **/home/shellScripts/read/readName.sh**.

```
#!/bin/sh
# Put your comments here

echo –n “Enter your name: ”
 read name
 echo "The name you entered is: $name"
```

8.14 This exercise provides additional practice using the `read` command to gather user input. Create a shell script, which prompts the user for their age and shoe size, and then prints out the values they entered. Run the shell script and create a screenshot showing the output.

8.15 This exercise provides practice combining the input from the `read` command with other commands. Create a shell script, which prompts the user for a username, and then prints out the user's account information from the /etc/passwd file by using the `grep` command. Run the shell script and create a screenshot showing the output.

8.16 This exercise provides experience reading more than one piece of information at a time, and the issues that may occur. Enter the following shell script, which prompts the user for their first and last names. (You can also copy the script from **/home/shellScripts/read/readNames.sh**.) Run the script at least 8 times, giving it two names, one name, three names, and four names, all separated by spaces, and then run the program again with 1-4 names all separated by commas. The point of this is for you to see how the shell handles reading data into multiple variables

```
#!/bin/sh
# Put your comments here

echo -n "Please enter first and last names: "
   read firstName lastName
   echo "Welcome to CBC $firstName"
   echo "Your last name is: $lastName"
```

Create a screenshot showing the output from your last run of the shell script.

8.17 Write a shell script that prompts the user for their name then prints out a customized welcome message. The message must include the name, the current date and time, end with a joke from the fortune file. NOTE: your Linux system will have the `date` program, but it may not have the `fortune` program. You can test this by typing `fortune`. If the program is on your system you will see a random fortune. If not, you will get the error message `command not found`. For example, your program's output should look something like the following:

```
############################################################
###########
#
#
#          Welcome Caterina
#
#
#
#          December 12, 2062
#
#          7:12
#
#
#
############################################################
###########

 Your fortune for today is:
 Trust Me. I'm the Doctor
```

Note that drawing a complete box around the welcome message is difficult as the amount of text on the lines containing the user name, the time, and the date will vary; so it's impossible to know how many spaces to add before the closing `#`. For now you can either omit the closing # on those three lines, or set the spaces so they work with a test name and the current date and time values. That is, your output will probably look like the following:

```
############################################################
###########
#
#
#          Welcome Caterina
#
#
#          December 12, 2062
#          7:12
#
#
############################################################
###########

 Your fortune for today is:
```

```
Trust Me. I'm the Doctor
```

Create a screenshot showing the output from the shell script.

8.18 This exercise provides practice using Linux commands in a shell script. Write a shell script which does the following:

a. Ensures the user is in their home directory
b. Prompts the user for a directory name
c. Creates the directory

Create a screenshot showing the contents of the shell script. (Remember to delete the directory after you're done with this exercise.)

8.19 This exercise provides practice using Linux commands in a shell script. Write a shell script which does the following:

a. Ensures the user is in their home directory
b. Uses the touch command to create or update a file. The file must be named **wLog.*dateStamp***, where ***dateStamp*** is the current date in the mmddyy format. For example, if the current date is July 20 2040 the file name should be **wLog.072040**.
c. Displays the message "Adding w output to the log file".
d. Runs the `w` command and appends the output to the file created in step b.
e. Displays the message "Current Log File Contents"
f. Displays the file created in steps b and d.

Create a screenshot showing the contents of the shell script.

8.20 This exercise provides practice using Linux commands in a shell script. Write a shell script which prints information from the /etc/passwd file to a file. The shell script must do the following:

a. Prompt the user for a user name
b. Finds the line(s) in the file /etc/passwd which contain the user name from step a. (Hint: use the `grep` command.)
c. Takes the lines from step b and removes all the fields except the first and the last using the `cut` command. Remember that the fields are delimited by the "**:**" or colon character. You should end up with the username and default shell.
d. Displays the line(s) from step c.

Create a screenshot showing the contents of the shell script.

8.21 Using the `script` command, record a script that does the following:

a. Use the `echo` command to write “Hello World” to the screen
b. Write the current date to the screen (use the `date` command)
c. Uses `grep` to print your account information from the file `/etc/passwd`
d. When you’re done recording the script type `exit`
e. Start the editor and load the file created by the script command. Create a screenshot showing the content of the file.
f. Use the editor to clean up the script by removing everything except for the commands you want to run.
g. Test out the script to ensure that it works
h. Create a screenshot showing the “cleaned up” version of the file.

8.22 Which of the following is true regarding the `expr` command?

a. It will take integers or floating point numbers as arguments, but always returns an integer result.
b. It will only take integers as arguments, and always returns an integer result.
c. It will take integers or floating point numbers as arguments, and will always return a floating point number.
d. None of the above

8.23 Assume you are using the `expr` command to perform integer math. Which of the following operators must be protected from taking on special meaning to the shell?

a. +
b. –
c. *
d. /
e. All of the above
f. None of the above

8.24 Assume you want to find the sum of two integers, **days1** and **days2**, and store the result in a variable named **totalDays**. Which of the following would accomplish this?

a. `` totalDays=`expr days1 + days2` ``
b. `` totalDays=`expr days1 \+ days2` ``
c. `totalDays=$(expr days1 + days2)`
d. `totalDays=$(expr days1 \+ days2)`
e. None of the above

8.25 Assume you want to multiply two integers, **height** and **width**, and store the result in a variable named **area**. Which of the following would accomplish this?

a. `` area=`expr height * width` ``
b. `` area=`expr height \* width` ``
c. `area=$(expr height * width)`
d. `area=$(expr height \* width)`
e. None of the above

8.26 Write a shell script which prompts the user for three integers, then multiplies them together and prints out the result. You can write the entire script or use the suggested method which is to modify one of the scripts in **/home/shellScripts/calc**. Create a screenshot showing the script, and another screenshot showing the output from the shell script.

8.27 Write a shell script which prompts the user for a three floating point numbers, then multiplies them together and prints out the result. You can write the entire script or use the suggested method which is to modify one of the scripts in **/home/shellScripts/calc**. Create a screenshot showing the script, and another screenshot showing the output from the shell script.

8.28 Write a shell script which calculates gas mileage. You can either write the entire script or use the suggested method which is to modify one of the scripts in **/home/shellScripts/calc**, such as **tickets2.sh**. The script should do the following:

a. Prompts the user and reads the number of miles driven
b. Prompts the user and reads the number of gallons of gas used
c. Uses `bc` to divide the miles driven by the gallons of gas to calculate the miles per gallon
d. Prints the result of the calculation along with some descriptive text.

Create a screenshot showing the script, and another screenshot showing the output from the shell script.

8.29 Write a car rental bill shell script. You can either write the entire script or modify the script **/home/shellScripts/calc/tickets2.sh**. At UNIX car rentals you can rents a car for $25 a day, plus $.05 for each mile you drive, plus $3.99/day insurance, plus 8% tax. The script should do the following

a. Read the users name, number of days the car was rented, and the number of miles driven
b. Calculate the daily, mileage, insurance, and tax payments, and the total payment
c. Print out a custom bill that includes the name, individual charges, and the total payment. (Try and format this so that it looks like a real car rental bill).

Create a screenshot showing the script, and another screenshot showing the output from the shell script.

8.30 Modify the IQ calculator script, which can be found in /home/shellScripts/iq.sh.

a. Read the users age and shoe size
b. If the shoe size is larger than 15 print a message saying that sizes must be in US units (between 1and 15) not European units.
c. If the shoe size is 15 or less calculate the IQ by multiplying the age by the shoe size, then print the result

Create a screenshot showing the script, and another screenshot showing the output from the shell script.

8.31 Create a car rental bill shell script. You can either write the entire script, or modify the script **/home/shellScripts/calc/tickets2.sh**. At UNIX car rentals you can rent a car for $25 a day, plus $.05 for each mile you drive over 100 miles, plus $3.99/day insurance, plus 8% tax. The script should do the following

a. Read the users name, number of days the car was rented, and the number of miles driven
b. Calculate the daily and, insurance payments
c. Calculate the mileage payment by checking to see if the mileage is greater than 100. If it is 100 or less there is no charge. If it is greater than 100 then subtract 100 from the mileage and multiply by 0.05
d. Calculate the tax and the total payment
e. Print out a custom bill that includes the name, individual charges, and the total payment. (Try and format this so that it looks like a real car rental bill).

Create a screenshot showing the script, and another screenshot showing the output from the shell script.

8.32 Write a shell script that uses a loop to build a table of weights on the Earth and the corresponding weight on the Moon. You can either write the entire script or modify **/home/shellScripts/loop/lbs2Kilos.sh**.

a. The table should start at 0 pounds and go to 250 pounds counting by 10. (You can hard code these numbers into the loop)
b. To calculate the weight on the moon, multiply the earth weight by 0.166
c. The table should have the headings “Earth Weight” and “Moon Weight”
d. The left-hand column of the table should contain the Earth weight in pounds, the right column should contain the corresponding weight on the moon.

Create a screenshot showing the script, and another screenshot showing the output from the shell script.

8.33 Write a script, or modify an existing script that calculates what items would weigh on the moon. The script must meet the requirements listed below. Hint – you may want to start with the script you wrote in one of the previous exercises.

a. The script must print a table of all the weights in a range.

b. The script should prompt the user and read the starting weight, ending weight and increment.

c. To calculate the weight on the moon, multiply the earth weight by 0.166

d. The table should have the headings “Earth Weight” and “Moon Weight”

e. The left-hand column of the table should contain the Earth weight in pounds, the right column should contain the corresponding weight on the moon.

f. After the table is built, the script should ask the user if they want to repeat the process and read their response. If the user responds in the affirmative the script should repeat, otherwise the script should exit.

Create a screenshot showing the script, and another screenshot showing the output from the shell script.