DPS918 Lab 1

Available selections:

1 Introduction To Unix Commands

2 Basic Unix Commands

3 Directory Management

4 Practice Commands To Create A Directory Structure

5 Create A Directory Structure

6 Practice Specifying Path Names

7 Permissions

8 Submit DPS918 Lab 1

The highlighted lines above indicate the incomplete parts of the Lab.

There is a total possible mark of 3, with 0.5 marks deducted for each

incomplete part. Late marks will be deducted at the rate of 0.3 marks (10%)

per day for Labs submitted after midnight of May 19 2017.

Unix & Linux Operating Systems

All computer systems require an operating system to control the computer hardware, to schedule

important routine tasks, and to allow the user to interact with the computer system.

The Unix & Linux operating systems are well suited for students at this college due to their many

features. Unix accomodates multiple users, provides for user communication, allows for sharing of

files, and provides easy access to the Internet.

Unix is a family of operating systems, (unofficially) including Linux. You will find slight differences

but many similarities between the various versions of Unix and Linux.

Shells

One method to communicate with the UNIX and LINUX operating systems is to enter commands (or program names) at a 'shell prompt'.

The shell is an interface between the Unix/Linux operating system and the user. When the shell is in

operation, it provides a prompt and waits for the user to type in a command and press the <ENTER> key.

The shell will not run or "execute" a command that the user types until the <ENTER> key is pressed.

This allows the user to edit or make corrections to their commands in case they make a mistake before

they press <ENTER>.

You can backspace at a shell prompt by pressing one of these keys:

<Backspace>

<Ctrl>-<Backspace>

<Ctrl>-h

Try to remember all three key combinations because different ones will work on different computers.

Type the following text: 'I made a mistike' then correct the text using the keys above. Press <ENTER>

when you're ready to continue.

If you notice strange characters such as "]" or "^?", simply try another key combination from the

list above!

Congratulations!

You are now ready to learn some of the basic Unix operating system commands.

This completes the topic 'Introduction to Unix/Linux'.

Press <ENTER> to return to the main menu. Select option 2 from the main menu to learn about basic Unix commands.

2.

UPPER- and lower-case

Let's demonstrate what happens if we enter a Unix or Linux command that is in UPPERCASE.

Did you notice the error message indicating 'PWD: command not found'?

This particular error message indicates that the shell cannot find the command called 'PWD'.

Try issuing the same command, but in lowercase.

This time the command ran correctly. Remember that Unix commands are generally entered in lowercase.

The output of the 'pwd' (Print Working Directory) command showed the 'directory' (or 'folder') which you are currently in. Your current directory

is /home/hhuang66. This happens to be your 'home' directory, which is the directory assigned to you by the system administrator to store your

personal files.

We will discuss directories in more detail later.

Listing Files

You can view a list of files (in your current directory) with the 'ls' (list) command.

(Tip: it is easiest to remember Unix commands, such as 'ls', if you know what they stand for, such as 'list'.)

$ ls

2 BTS530 DPR\_TEST1 Download Pictures Templates addps newfiles1.txt test.cpp

Acceptable-Use-Policy DPR355 Desktop Huiwen Process.h Videos bin public\_html test.txt

BTP305 DPR\_TEST Documents Music Public \ letter2bob sampleletter testlab

Listing Files with Detail

The 'ls' command does list the names of your files, but it doesn't tell you anything about them.

In order to provide more information regarding each file contained in your home directory, issue the command 'ls -l'.

$ cp lab1\_sub/myHistory ~

lab1a

`-- lab1

|-- cars

|-- lab1\_sub

| |-- group

| |-- group.copy

| |-- lab1\_sub.copy

| |-- myHistory

| `-- passwd.copy

`-- passwd

Your current directory is lab1. Copy ~/.bash\_history to lab1\_sub.copy:

$ cp ~/.bash\_history lab1\_sub/lab1\_sub.copy

lab1a

`-- lab1

|-- cars

|-- lab1\_sub

| |-- group

| |-- group.copy

| |-- lab1\_sub.copy

| | `-- .bash\_history

| |-- myHistory

| `-- passwd.copy

`-- passwd

Your current directory is lab1. List detailed info about all the files

in the lab1\_sub.copy directory:

$ ls -al lab1\_sub/lab1\_sub.copy

total 32

drwxr-xr-x 2 hhuang66 users 26 May 13 18:58 .

drwxr-xr-x 3 hhuang66 users 89 May 13 18:47 ..

-rw------- 1 hhuang66 users 29271 May 13 18:58 .bash\_history

lab1a

`-- lab1

|-- cars

|-- lab1\_sub

| |-- group

| |-- group.copy

| |-- lab1\_sub.copy

| | `-- .bash\_history

| |-- myHistory

| `-- passwd.copy

`-- passwd

Your current directory is lab1. Using rmdir, try to delete lab1\_sub.copy,

and note what happens:

$ rmdir lab1\_sub/lab1\_sub.copy

rmdir: failed to remove `lab1\_sub/lab1\_sub.copy': Directory not empty

lab1a

`-- lab1

|-- cars

|-- lab1\_sub

| |-- group

| |-- group.copy

| |-- lab1\_sub.copy

| | `-- .bash\_history

| |-- myHistory

| `-- passwd.copy

`-- passwd

Your current directory is lab1. Delete .bash\_history within lab1\_sub.copy:

$ rm lab1\_sub/lab1\_sub.copy/.bash\_history

lab1a

`-- lab1

|-- cars

|-- lab1\_sub

| |-- group

| |-- group.copy

| |-- lab1\_sub.copy

| |-- myHistory

| `-- passwd.copy

`-- passwd

Your current directory is lab1. Using rmdir, try to delete lab1\_sub.copy

again. It should work this time:

$ rmdir lab1\_sub/lab1\_sub.copy

lab1a

`-- lab1

|-- cars

|-- lab1\_sub

| |-- group

| |-- group.copy

| |-- myHistory

| `-- passwd.copy

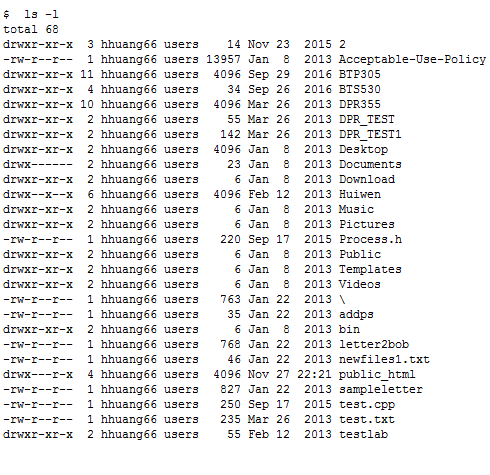
`-- passwd

Change to your home directory again, using the fewest possible keystrokes:

$ cd

Congratulations!

You have successfully completed this section!



We'll discuss the output of the 'ls -l' command in more detail later.

The important thing to notice is that 'ls -l' and 'ls' are the same command, but the addition of the '-l' has changed the way that 'ls' works.

Any additional information that you give to a command is called an argument; an argument that changes the way a command works is a special type

of argument called an option.

Changing Directories

You can change directories to make another directory your current (or 'working') directory. The command to do this is 'cd'. Let's use the 'cd'

command to make the directory '/bin' our current directory.

$ cd /bin

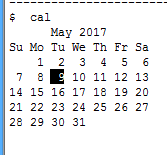
Notice that the cd command doesn't produce any output if it is successful. Most Unix commands are designed not to produce unnecessary output,

so they will not produce a message if it is not required and no error occurred.

Now enter a command to verify that you changed to the 'bin' directory (we used this command a few pages back).

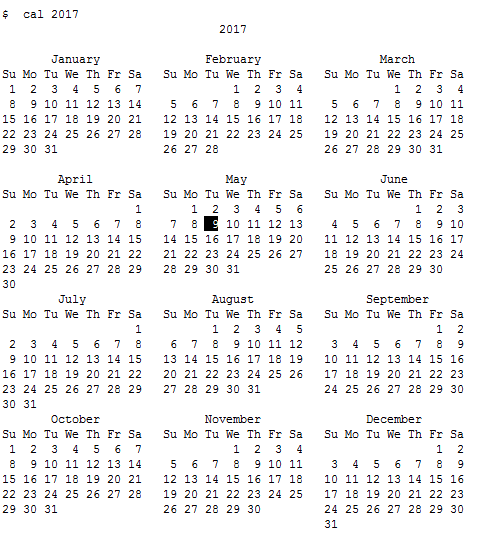


Now let's return to your home directory. If you don't specify the directory when using the 'cd' command, your home directory will be made the current directory.



You should notice just the calendar for the current month - May.

To display an entire calendar for the year 2017, enter 'cal 2017'



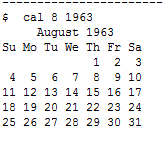
You will notice all months appear for the year 2017. Unfortunately, all of the contents of the calendar may not appear on the terminal at the

same time.

You will learn later in this course to selectively display different parts of the output.

To display the calendar for the month of August for the year 1963, enter the following command:

cal 8 1963



Well Done!

You should practise these Unix commands on your own after you have completed this tutorial. In this way, you get used to using these commands

as a tool when you use Unix/Linux.

This completes the topic 'Basic Unix/Linux Commands'.

Please proceed with the Review Exercise, option 3 on the Main Menu.

Review Exercise

The purpose of this section is to reinforce your skills with issuing UNIX commands covered in the previous sections.

You will be asked to enter a single UNIX command for the following situations. If you are unable to successfully issue the UNIX command after

several attempts, a hint will be provided.

Question 1

Issue a UNIX command to display your Login username.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#2

Unix Command Structure:

command [ argument1 ] [ argument2 ] ... [ argument n ]

NOTE:

An argument (or parameter) is additional information given to a command program, such as command options, filenames, strings of text, numbers

or other elements that the command needs.

One or more spaces or tabs must appear between arguments and between the command and the first argument.

In documentation, square-brackets [ ] indicate that an argument is optional.

Types of Arguments

There are three types of arguments:

- Positional Arguments

- Simple Options

- Options with a Value

Positional Arguments

Positional arguments are expected to appear in a certain sequence. For example, the 'copy' command is written:

cp original destination

The first file listed is the original file (that you are copying FROM) and the second file is the destination (that you are copying TO). If you

change the position of the arguments, you change their meaning.

Positional arguments generally tell the command what it is to operate upon (for example, the file to be managed).

Simple Options

Options are arguments which modify the way in which a command operates. Options usually start with a dash or a double-dash.

For example, the 'ls' command accepts the option '-l'. When that option is entered, 'ls' changes its operation to give a longer output.

Generally, in Linux options may be specified anywhere on a command line, and do not interfere with positional arguments.

In Unix, options must often be specified before positional arguments.

Options with a Value

Some options need an additional value. This lets you specify details of how you want a command's operation to be changed. This is like an option

combined with a positional argument.

As an example, the 'sort' command has an option '-k' which lets you specify the number of the field to sort on. The '-k' option must be immediately

followed by the number of the field.

For example, 'sort -k 5' would sort a file by the fifth field instead of the entire record.

Spaces Between Arguments

To demonstrate the use of commands with arguments, let's refer back to the cal command. As you may recall, the Unix command to display the

month of August 1963 is

cal 8 1963

Try entering this command with more than one SPACE between the command and the first argument and/or between arguments.

Note that having multiple spaces had no effect on the operation of the command.

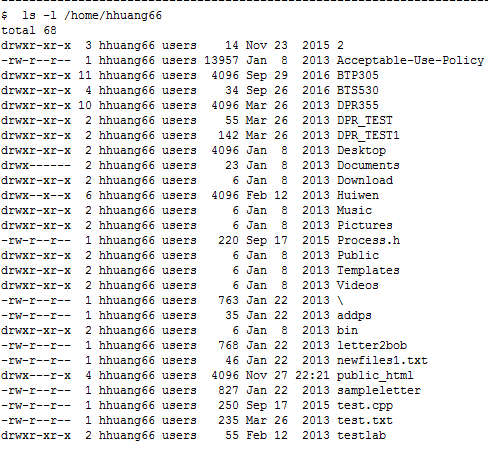
From now on, you can leave as much space as you want between arguments.

Using Options

As you may recall, you can display detailed information about a file by issuing the command 'ls -l'.

We can also add a positional argument specifying the directory to be displayed. Issue the following command to display your home directory:

ls -l /home/hhuang66

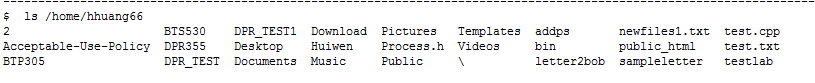


Using Options

Notice the additional information compared to the 'ls' command without the '-l' option.

Issue the same command without '-l' to compare:

ls /home/hhuang66



To demonstrate that an option is not a positional argument, let's re-issue the 'ls -l /home/hhuang66' command with the arguments switched.

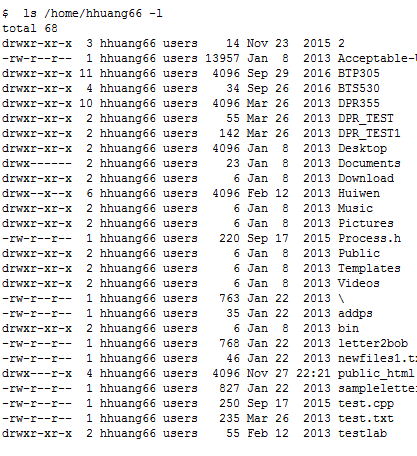
To demonstrate that an option is not a positional argument, let's re-issue the 'ls -l /home/hhuang66' command with the arguments switched.

You entered: ls -l /home/hhuang66

Please try again.

Hint:

ls /home/hhuang66 –l



That concludes the introduction to the Unix command structure.

Please proceed to the section on Entering and Editing Commands.

Entering and Editing Commands

There are a number of keys that may be used to enter and edit commands on Unix servers. Unfortunately, the keys you issue will depend upon your

account configuration and which shell you are currently using. On Linux, using bash:

<Up Arrow> Display previous command

<Down Arrow> Display next command

Additional Keystrokes

<Ctrl>-c Cancel a running command

<Ctrl>-l CLear the screen and continue editing

<Ctrl>-u Undo the command being entered (erase it and start over).

Note: These keystrokes will work at the command prompt, but may not work within these Labs. DO NOT use <Ctrl>-c while this Lab is running,

or the Lab will stop.

Using the Online Manual

There is an online manual command in Linux and Unix called 'man'. You can get help about a command by entering:

man command

Issue the command 'man ls' to display online help regarding the ls command. If you're given a choice of 'man' sections, choose section 1.

Important: When you are in the man command, press <ENTER> to move down one line, <SPACE> to move down one screen, and 'b' to move up one screen.

Press 'q' to quit from the online help and return to this Lab.

Finding Commands

'man' is fine if you know which command you want to use, but since there are 3728 commands on this computer, it's sometimes difficult to remember

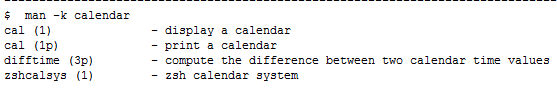
(or find) the right one to use.

'man' can also locate a command if you give it a keyword, using the -k option:

man -k keyword

A brief description of all of the commands with the keyword in their summary will be shown. Let's try getting a list of commands that deal

with calendars...



Once you have used 'man -k' to find possible commands, you can use the regular 'man' command to examine those commands in detail and determine

if they will do what you want.

Do not hesitate to use the man pages -- even experienced Unix users refer to the online documentation for options and commands that they

infrequently use.

That concludes the section on Using the Online Manual. Please proceed to the section on Managing Files.

Available selections:

1 Unix Command Structure - Completed

2 Entering And Editing Commands - Completed

3 Using The Online Manual - Completed

4 Managing Files

5 Accessing Files

6 Review Exercise

Note that the highlighted lines indicate the incomplete sections of

this part of the Lab.

The Review Exercise can only be done when the other sections have

been completed.

Creating Empty Files

Before starting actually working with files, it would be useful to create some files. You are going to use the touch command to create empty files.

This is a great technique to create files to practise with.

$ touch file1 file2 file3

DPS918 Lab 1.1: Introduction To Unix/Linux Page 3/4

---------------------------------------------------------------------------------------------------------

You can backspace at a shell prompt by pressing one of these keys:

<Backspace>

<Ctrl>-<Backspace>

<Ctrl>-h

Try to remember all three key combinations because different ones will work on different computers.

Type the following text: 'I made a mistike' then correct the text using the keys above. Press <ENTER>

when you're ready to continue.

If you notice strange characters such as "]" or "^?", simply try another key combination from the

list above!

Enter the line as requested, then correct it before hitting <ENTER>

DPS918 Lab 1.1: Introduction To Unix/Linux Page 3/4

---------------------------------------------------------------------------------------------------------

You can backspace at a shell prompt by pressing one of these keys:

<Backspace>

<Ctrl>-<Backspace>

<Ctrl>-h

Try to remember all three key combinations because different ones will work on different computers.

Type the following text: 'I made a mistike' then correct the text using the keys above. Press <ENTER>

when you're ready to continue.

If you notice strange characters such as "]" or "^?", simply try another key combination from the

list above!

Available selections:

1 Introduction To Unix/Linux

2 Basic Unix/Linux Commands

3 Review Exercise

Note that the highlighted lines indicate the incomplete sections of

this part of the Lab.

The Review Exercise can only be done when the other sections have

been completed.

Available selections:

1 Introduction To Unix/Linux

2 Basic Unix/Linux Commands

3 Review Exercise

Note that the highlighted lines indicate the incomplete sections of

this part of the Lab.

The Review Exercise can only be done when the other sections have

been completed.

Review Exercise

NOTE: You may NOT do these review questions, YOU MUST COMPLETE SECTION 1 first.

DPS918 Lab 1.1: Introduction To Unix Commands

---------------------------------------------------------------------------------------------------------

Available selections:

1 Introduction To Unix/Linux

2 Basic Unix/Linux Commands

3 Review Exercise

Note that the highlighted lines indicate the incomplete sections of

this part of the Lab.

The Review Exercise can only be done when the other sections have

been completed.

You can backspace at a shell prompt by pressing one of these keys:

<Backspace>

<Ctrl>-<Backspace>

<Ctrl>-h

Try to remember all three key combinations because different ones will work on different computers.

Type the following text: 'I made a mistike' then correct the text using the keys above. Press <ENTER>

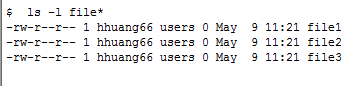
when you're ready to continue.

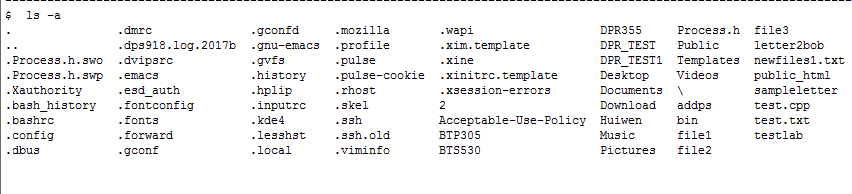
If you notice strange characters such as "]" or "^?", simply try another key combination from the

list above!

$ ls file\*

file1 file2 file3





Notice that a compact file list is displayed. Dot '.' represents the current directory, and double-dot '..' represents the

parent directory (we'll talk more about this later).

Files that appear with a period as the first character in the filename are called hidden files.

Copying Files

The cp command is used to copy files. Command structure:

cp source-file destination-file

Use the 'cp' command to copy file1 to file4 in the current directory

$ cp file1 file4

Renaming (Moving) Files

The 'mv' ('move' to a new name) command is used to rename files. Command Structure:

mv [source-file] [destination-file]

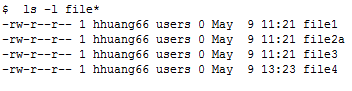
or

mv [old file-name] [new file-name]

Enter the mv command below to change the name of file2 to file2a

$ mv file2 file2a

Good. Now verify that the 'mv' command was successful



Good. That completes the section on managing files. We will be discussing this topic in

greater depth.

Proceed to the section on Accessing Files.

Accessing Files

You will be required to display the contents of a file from time to time. Four popular

Linux commands to view file contents are:

· cat

· more

· head

· tail

Let's use a file for experimenting.

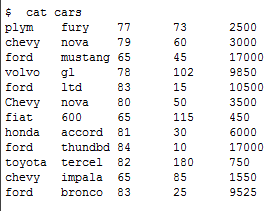
Copy a file called 'cars' from the directory ~dps918/2017b into your current directory,

using the command:

cp ~dps918/2017b/cars .

Well done, the file was copied.

The 'cat' command will dump the contents of the file on to the display. Let's try it...



In order to demonstrate some other commands, the file 'file1' has been created containing

a calendar of the years 2010-2016 to make it large.

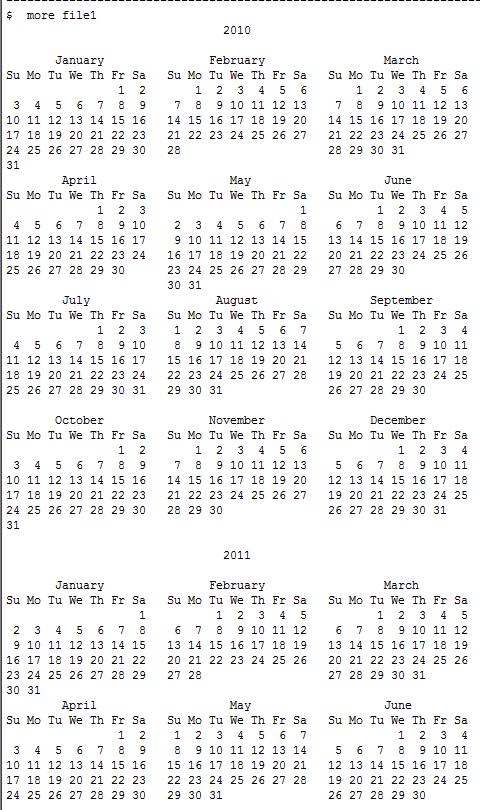
To view files larger than your terminal window, you can issue the 'more' command. This

command allows the user to move throughout the file using movement keys that are similar

to the man command.

When using 'more', the spacebar will advance one page and <Enter> will advance one

line. Press 'q' to quit.

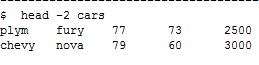


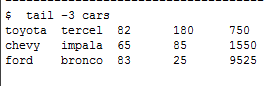
You can also use the 'head' and 'tail' commands to view a number of lines at the

beginning and end of files respectively.

You can specify the number of lines that will be displayed from the top or bottom of

the file by using an option -# where # represents the number of lines to be displayed.





Displaying File Contents

There are many commands that allow the user not only to display the contents of a file

(such as cat, head, or tail), but to modify the display.

Some of these commands are:

· sort

· uniq

· diff

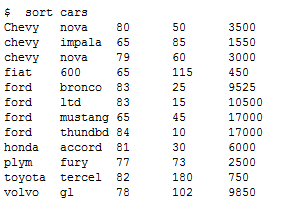
· grep

sort

The sort command is used to sort contents of a file. Command Structure:

sort [options] filename

Try this command to sort the contents of 'cars':



uniq

The uniq command is used to display only one occurence of identical adjacent lines. If

you want to prevent duplication of identical lines that are NOT adjacent, the file

should be sorted prior to using this command (or use sort -u). Command Structure:

uniq [options] [filename]

In order to demonstrate the 'uniq' command, the file 'file2' has been created with

the following contents:

This is a line

This is a line

This is NOT the same line

This is a line

This is a line

Now, issue the uniq command on file2 and notice what happens:

$ uniq file2

This is a line

This is NOT the same line

This is a line

Notice that the uniq command only displays identical adjacent lines once. In order

to avoid duplication in the entire file, you should sort the file prior to using

the uniq command.

Later we will examine how commands such as 'sort' and 'uniq' can be used together.

The diff command is used to display differences between two files.

Let's try it:

$ diff cars file2

1,12c1,5

< plym fury 77 73 2500

< chevy nova 79 60 3000

< ford mustang 65 45 17000

< volvo gl 78 102 9850

< ford ltd 83 15 10500

< Chevy nova 80 50 3500

< fiat 600 65 115 450

< honda accord 81 30 6000

< ford thundbd 84 10 17000

< toyota tercel 82 180 750

< chevy impala 65 85 1550

< ford bronco 83 25 9525

---

> This is a line

> This is a line

> This is NOT the same line

> This is a line

> This is a line

Did you notice greater-than signs '>' and less than signs '<' before text? These

signs are used to indicate:

· '>' - Text in the second file that is not in the first file

· '<' - Text in the first file that is not in the second file

The grep command is used to display matching patterns within a file. Command structure:

grep [options] [pattern] [filename]

Use the grep commmand to search for and display all lines containing the pattern

'chevy' in cars.

$ grep 'chevy' cars

chevy nova 79 60 3000

chevy impala 65 85 1550

Review Exercise

The purpose of this section is to reinforce your skills with issuing UNIX

commands covered in the previous sections.

You will be asked to enter a single UNIX command for the following

situations. If you are unable to successfully issue the UNIX command after

several attempts, a hint will be provided.

Data File

A data file has been created named 'poem'. It contains these lines:

This is my file

It is not your file

So hands-off!

So hands-off!

Question 1

Enter a command to display the contents of 'poem' all-at-once on the screen.

$ cat poem

This is my file

It is not your file

So hands-off!

So hands-off!

Question 2

Enter a command to display the first 3 lines of 'poem' on the screen.

$ head -3 poem

This is my file

It is not your file

So hands-off!

Question 3

Enter a command to display the last 2 lines of 'poem' on the screen.

$ tail -2 poem

So hands-off!

So hands-off!

Question 4

Enter a command that will display the lines from 'poem' in sorted order.

$ sort poem

It is not your file

So hands-off!

So hands-off!

This is my file

Question 5

Enter a command that will display the unique adjacent lines from 'poem'. You are

not required to sort the file.

$ uniq poem

This is my file

It is not your file

So hands-off!

Question 6

Issue a Linux command to create the following empty files in your current

directory:

a\_text

a\_memos

a\_reports

$ touch a\_text a\_memos a\_reports

Question 7

Issue a Linux command to copy the file 'a\_text' to 'a\_text.bak' in your current

directory.

$ cp a\_text a\_text.bak

Question 8

Issue a Linux command to rename the file 'a\_reports' to 'a\_announcements'.

$ mv a\_reports a\_announcements

Question 9

Issue a Linux command to compare differences between the files 'poem' and 'a\_text'.

$ diff poem a\_text

1,4d0

< This is my file

< It is not your file

< So hands-off!

< So hands-off!

DPS918 Lab 1.3: Unix Filesystem Hierarchy

Unix Filesystem Hierarchy

The Linux OS has a 'hierarchical' file system, which means that directories can contain other directories or files. This is also known as a 'tree diagram', which is usually shown with the root directory at the top:

/ (root)

|

--------------------------------

| | | |

bin dev usr etc

|

---------------

| | |

A B C

In the Linux and Unix OS, the storage device such as the hard disk contains directories. Directories are actually files that are used to store other

files. This allows the storage device to be organized and prevent confusion.

The root directory (denoted by a slash '/') is the beginning (or master) directory.

All other directories are related to the root directory -- they are contained, directly or indirectly, within the root directory.

Directory Terminology:

· Directory - A 'directory file' used to store other files.

· Sub-directory - A directory file DIRECTLY contained within a

'parent' directory. Also called a 'child directory'.

In the diagram, 'usr' is a child of '/', the root.

· Parent Directory - A directory that contains one or more children.

In the diagram, '/' is the parent of 'bin', 'dev',

'usr', and 'etc'.

/ (root)

|

--------------------------------

| | | |

bin dev usr etc

Important things to note about Unix file hierarchies:

- There are no drive letters. All of the disk drives are 'mounted' into a single hierarchy.

- In Unix, file names are used for many types of resources that you would not usually think of as files, such as directories, devices, communication

between programs, status information, and network connections (remember that 'everything is a file').

- Files do not have to have an 'extension', though you may use one to indicate the type of file content (for example, '.html' to indicate HTML web files).

Naming Files (and Directories)

The maximum length of a file name, whether ordinary files or directories, varies from system to system. The sizes can range from 14 to 255 characters.

Most characters can be used in filenames, but for simplicity, you should stick to:

· Uppercase or lowercase letters

· Numbers

· Underscore '\_'

· Period '.'

· Comma ','

Note:

· A period BEFORE a filename 'hides' the file.

· All file names are case sensitive.

Great!

You've finished the 'Unix Filesystem Hierarchy' section. Please proceed to the section on 'Pathnames'.

DPS918 Lab 1.3: Pathnames

Pathnames

A pathname is used to specify the location of a file or directory. Pathnames are used when issuing Linux commands when working with directories and files.

There are three categories of pathnames:

· Absolute Pathname - location reference starting from root '/'

· Relative Pathname - location reference starting from the 'current directory'

· Relative-to-Home Pathname - location reference starting from your 'home directory'

Absolute Pathnames

An absolute pathname specifies how to find a file, starting at the root directory and working your way down to a specific file. Absolute pathnames always

start with a slash '/' character.

An absolute pathname consists of a list of directories separated by slashes '/' and ending with the particular file or directory to which you are referring.

For example, the absolute pathname:

/etc/system/config/ejb.conf

specifies the file 'ejb.conf' stored within the 'config' directory, which is within the 'system' directory, which is within the 'etc' directory, which is

within the root directory.

Relative Pathnames

Unix systems employ the concept of a current directory, which you can set to any directory in the hierarchy using 'cd'.

A relative pathname specifies how to find a particular file or directory starting from the current directory.

If the current directory is set to '/etc/system', then these pathnames are equivalent:

Relative Pathname Absolute Pathname

foo.txt /etc/system/foo.txt

conf/current /etc/system/conf/current

old/red/blue /etc/system/old/red/blue

Relative Pathnames

When using relative pathnames, these symbols are available:

. THIS (current) directory

.. Parent directory

If the current directory is set to '/etc/system', then these pathnames are equivalent:

Relative Pathname Absolute Pathname

. /etc/system

foo.txt /etc/system/foo.txt

./foo.txt /etc/system/foo.txt

../foo.txt /etc/foo.txt

../../home/joe/text /home/joe/text

Relative-to-Home Pathnames

Each user of a Unix system is assigned a home directory by the system administrator. Home directories may be located anywhere in the hierarchy.

The home directory is provided for you to store your personal files and directories. On this computer, you have been assigned the home directory '/home/hhuang66'

Note that your home directory is NOT necessarily the same as your current directory. Your current directory changes as you move around the file hierarchy,

but your home directory stays the same.

Relative-to-Home pathnames start with a tilde '~'.

If you specify a tilde followed by a slash, the pathname is relative to your home directory.

You can also specify a pathname relative to another user's home directory by starting your pathname with '~userid/'.

Your home directory is /home/hhuang66, so these pathnames are equivalent:

Relative-to-Home Absolute Pathname

~/lastyear.txt /home/hhuang66/lastyear.txt

~/unix/notes /home/hhuang66/unix/notes

If your friend's userid was 'jstudent' and her home directory was '/home/jstudent', these pathnames would be equivalent:

Relative-to-Home Absolute Pathname

~jstudent/test /home/jstudent/test

~jstudent/2017/diagram /home/jstudent/2017/diagram

That's the end of the 'Pathnames' section.

Now go ahead to the section 'Ambiguous Pathnames'.

Ambiguous Pathnames

Unix provides the 'ambiguous pathnames' feature to permit you to specify a number of files at once.

There are three 'wildcard' symbols used in ambiguous pathnames:

\* matches zero or more of any character

? matches exactly one of any character

[class] matches exactly one character from the class

The Asterisk

The asterisk matches zero or more characters, like this:

Pattern Matches Doesn't Match

\* Matches all non-hidden filenames

a\*b ab A12

a2314234b able

all\_the\_tab resub

\*txt file.txt txtfile

file\_txt bastxtyot

texttxt .file.txt

txt

The Question-Mark

The question-mark matches exactly one of any character, like this:

Pattern Matches Doesn't Match

? Matches all one-character filenames except . (dot)

a aa

b

a?b a2b ab

axb acesdb

a\_b

?txt atxt notes.txt

3txt txt

qtxt .txt

The Character-Class

A character-class is enclosed in square brackets [ ] and may include a list of characters or a range of characters separated by a dash. The character

class matches any one character in the list or ranges.

Ambiguous Pathname Matches Does not match

[abc] a A

b aligator

c aaaaa

notes[127-9] notes1 notes3

notes2 notes

notes7 notes.txt

notes8 notes127

notes9

DPS918 Lab 1.3: Ambiguous Pathnames Page 6/16

-------------------------------------------------------------------------------------------------------------------------------------

Globbing

The process of converting an amiguous filename into a list of matching file names is called 'globbing' and is performed by

the shell. On other operating systems, the equivalent operation is performed by the command and not by the shell.

Be aware that because globbing is performed by the shell, globbing is applied to all arguments, whether they are supposed to

represent files or some other information.

For example, the 'echo' command displays information on the screen. If you type 'echo \*', the asterisk will be replaced by

a list of all of the files in the current directory. To avoid this, use quotes around the argument.

In order to demonstrate the use of 'wildcard' characters, issue a command to create the following empty files in your current

directory:

a1 a11 a123 a1234 a23 a22 a3 1 11 123

$ touch a1 a11 a123 a1234 a23 a22 a3 1 11 123

The files have been created.

Now issue an 'echo' command to display all filenames that begin with the letter 'a'.

$ echo a\*

a1 a11 a123 a1234 a22 a23 a3 addps

Now, issue an 'echo' command to display filenames in your current directory that begin with 'a' and end with '3'

$ echo a\*3

a123 a23 a3

You just saw the output from 'echo' with the pattern 'a\*3'.

To demonstrate, the difference between \* and ? used as wildcards, issue the command 'echo a?3'

$ echo a?3

a23

Notice that using the '\*' symbol matches zero or more characters, as opposed to matching exactly one character with the '?' symbol.

You can also use multiple '?' symbols to match a certain number of characters.

$ echo a??3

a123

Character classes behave like the '?' wildcard, except you can specify what characters '?' can or cannot represent.

First, let's display filenames that begin with either the lowercase letter 'a' (case matters) or begin with the number '1'.

$ echo [a1]\*

1 11 123 a1 a11 a123 a1234 a22 a23 a3 addps

Good.

Now let's display filenames that end with the number '1' or '2' or '3', using the range '[1-3]'

$ echo \*[1-3]

1 11 123 2 DPR\_TEST1 a1 a11 a123 a22 a23 a3

You can also combine ranges and/or lists.

For example, to list all filenames that begin with a letter (whether upper or lowercase), use the ambiguous filename '[a-zA-Z]\*'

Note that globbing is used with any command and any argument, whether meant to be a filename or not.

$ echo [a-zA-Z]\*

Acceptable-Use-Policy BTP305 BTS530 DPR355 DPR\_TEST DPR\_TEST1 Desktop Documents Download Huiwen Music Pictures Process.h Public Templates Videos a1 a11 a123 a1234 a22 a23 a3 addps bin letter2bob newfiles1.txt public\_html sampleletter test.cpp test.txt testlab

Now let's invert a character class using '!'.

Let's display all filenames that do not start with 'a'.

$ echo [!a]\*

1 11 123 2 Acceptable-Use-Policy BTP305 BTS530 DPR355 DPR\_TEST DPR\_TEST1 Desktop Documents Download Huiwen Music Pictures Process.h Public Templates Videos bin letter2bob newfiles1.txt public\_html sampleletter test.cpp test.txt testlab

DPS918 Lab 1.3: Ambiguous Pathnames Page 16/16

-------------------------------------------------------------------------------------------------------------------------------------

Well done!

Now proceed to the next section to try out some directory management commands.

DPS918 Lab 1.3: Directory Management Commands

Let's work with pathnames and directories.

You create directories by issuing the linux command:

mkdir directory-name

The command above is using a relative pathname; therefore, the directory will be created as a "child" directory. Below is an

example of using an absolute pathname to create a directory:

mkdir /students/username/directory-name

To save time, you can create more than one child directory under the current directory by placing multiple directory names

within the same Linux command:

eg. mkdir d1 d2 d3

Let's issue this command to create some directories in which you can practice.

The directories have been made.

Now use the 'ls' command with an ambiguous pathname to verify that the directories have been made.

$ ls -ld d?

drwxr-xr-x 2 hhuang66 users 6 May 11 10:53 d1

drwxr-xr-x 2 hhuang66 users 6 May 11 10:53 d2

drwxr-xr-x 2 hhuang66 users 6 May 11 10:53 d3

You can also issue a Linux command to create directories even if their 'parent' directories do NOT exist. To do this, add the

'-p' (parent) argument to the mkdir command.

Let's create a child directory with its parents.

You can also issue a Linux command to create directories even if their 'parent' directories do NOT exist. To do this, add the

'-p' (parent) argument to the mkdir command.

Let's create a child directory with its parents.

$ mkdir -p testing/1/2/3

Now we will issue a command to verify that the directories have been made, using a relative pathname.

$ ls -ld testing/1/2/3

drwxr-xr-x 2 hhuang66 users 6 May 11 10:56 testing/1/2/3

Now issue the same 'ls -ld testing/1/2/3' command with an absolute pathname.

(Note that your current directory is your home directory: /home/hhuang66 ).

You entered: ls -ld testing/1/2/3

Please try again.

Hint:

Try 'ls -ld /home/hhuang66/testing/1/2/3'

$ ls -ld /home/hhuang66/testing/1/2/3

drwxr-xr-x 2 hhuang66 users 6 May 11 10:56 /home/hhuang66/testing/1/2/3

You can change your current directory using the 'cd' command.

Change to the '/tmp' directory.

$ cd /tmp

We can check our current directory with the 'pwd' command (print working directory).

$ pwd

/tmp

Now use a relative-to-home pathname to list info about the directory that you created in your home directory a few minutes ago.

$ ls -ld ~/testing/1/2/3

drwxr-xr-x 2 hhuang66 users 6 May 11 10:56 /home/hhuang66/testing/1/2/3

If you do not specify any arguments for the 'cd' command, you will set your current directory back to your home directory.

Let's try it...

$ cd

Now check which directory is your current (working) directory.

$ pwd

/home/hhuang66

Create the empty file called 'file1' in the directory called '3' that you just created. You may use any of the types of pathnames.

$ touch testing/1/2/3/file1

Now issue a command to verify that you created the file.

$ ls testing/1/2/3

file1

Use the 'mv' command to move directory '3' from under directory '2' to under directory '1'.

Use the 'mv' command to move directory '3' from under directory '2' to under directory '1'.

You entered: mv 3 .2

Please try again.

Hint:

Try 'mv testing/1/2/3 testing/1'

Now verify that this directory has been moved using the '-R' argument to the 'ls' command.

$ ls -R testing/1

testing/1:

2 3

testing/1/2:

testing/1/3:

file1

Directories may be removed with the command 'rmdir', but:

- You cannot be in the directory that you're trying to remove (or one of its child directories).

- The directory must be empty.

Try removing directory 3 using the 'rmdir' command.

$ rmdir testing/1/3

rmdir: failed to remove `testing/1/3': Directory not empty

You probably noted an error message.

The directory '3' contains a file, so it is not empty. This means that you cannot remove this file using just the 'rmdir' command.

There is a more powerful command available to remove entire directory paths. The command 'rm -r' will remove non-empty

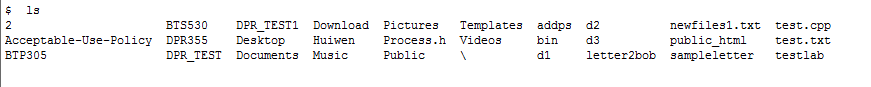
directories which contain files or child directories, grandchild directories, etc.

Use the option -i to prompt whether each directory or file is to be removed.

In this case, verify that each directory or file should be removed by typing 'y' at each prompt.

$ rm -ir testing

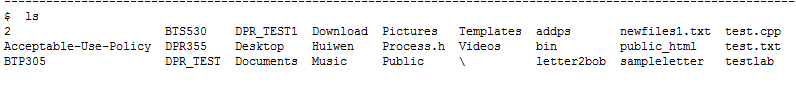
rm: descend into directory `testing'?



Let's remove the empty directories 'd1', 'd2', and 'd3' using the 'rmdir' command, using the question-mark wildcard.

$ rmdir d?

Verify that the directories have been removed.



Good work!

Now proceed to the Review Exercise.

DPS918 Lab 1.3: Review Exercise

Question 1

Enter the Linux command to create both a parent directory called 'systems' and it's child directory called 'part3' at the same

time. Assume that directory 'systems' will branch-off your home directory and that you are in your home directory to start. Use

a relative pathname.

-------- My practice ------

systems

part3

home directory : /home/hhuang66

/home/hhuang66/systems/parts

mkdir –p systems/parts3

Question 2

Enter a Linux command to create the empty file called 'test1' in the directory

'systems' (you are still in your home directory).

touch systems/test1

Question 3

Assume that your current directory contains the files 'labtest', 'labtest1',

'labtest2', 'labtest2a', 'labtest3', and 'labtest4'.

Issue a command to delete only the files 'labtest1' and 'labtest2' using one

ambiguous pathname.

rm labtest[1-2]

Question 4

Assume that you are NOT currently in your home directory.

Enter a command to copy all files in your home directory beginning with the letter

'a' to the current directory.

cp ~/a\* .

Question 5

Enter a command to delete all files that have filenames starting with 'labtest',

except 'labtest' itself, from the current directory (Delete all files starting

with 'labtest' followed by one or more characters).

rm labtest?\*

Question 6

Enter a Linux command to remove a directory called

'assignments' and all of it's contents, and allow

the system to prompt the user to remove each file or

directory.

Assume you are in the home directory, which happens to

be the parent directory of the 'assignments' directory.

rm -ir assignments

Question 7

Here are two inverted-tree diagrams. Issue a command to

change the left tree structure to be the same as the

tree structure on the right. Assume that you are in

your home directory and use relative pathnames. [home]

is your home directory:

[home] [home]

| |

+----------+----------+ +-------------+

| | | | |

systems ideas courses ideas courses

| |

notes systems

|

notes

mv system courses

Question 8

Issue a command to delete all ordinary files in your current directory with

2-character names.

rm ??

Question 9

Issue a command to delete the directory 'courses' (in your home directory)

and all of its children.

Use an absolute pathname (remember that your home directory is /home/hhuang66

).

rm -r /home/hhuang66/courses

Question 10

Enter a command to make the root directory your current directory.

cd /

Congratulations!

You have successfully completed the Review Exercise!

Available selections:

1 Unix Filesystem Hierarchy - Completed

2 Pathnames (Absolute, Relative, Relative-to-Home) - Completed

3 Ambiguous Pathnames - Completed

4 Directory Management Commands - Completed

5 Review Exercise - Completed

Note that the highlighted lines indicate the incomplete sections of

this part of the Lab.

The Review Exercise can only be done when the other sections have

been completed.

DPS918 Lab 1.4: Practice Commands To Create A Directory Structure Page 1/26

Practice Commands to Create a Directory Structure

The purpose of this section is to reinforce your skills with issuing UNIX

commands covered previously.

You will be asked to enter a single UNIX command for the following

situations. If you are unable to successfully issue the UNIX command after

several attempts, a hint will be provided.

\*\*\* Note that in most cases, only the shortest possible command line will

be accepted.\*\*\*

You are currently in your home directory. Within your home directory, create

a directory called lab1 using the shortest possible path name:

$ mkdir lab1

Copy the file called cars that resides in the directory ~dps918/2017b to your

lab1 directory:

$ cp ~dps918/2017b/cars lab1

Make lab1 your present working directory:

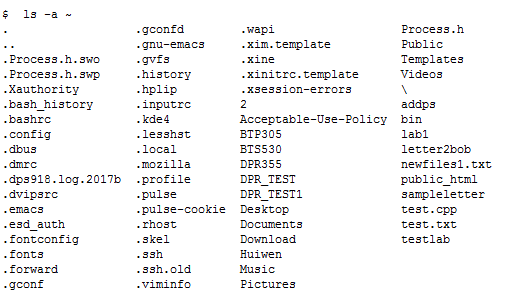
$ cd lab1

Without changing working directories, list only the names of files in your home

directory. Use the shortest possible pathname, and be sure that hidden files

are also listed:

$ ls -a ~



Change to your home directory using the fewest possible keystrokes:

$ cd

While in your home directory, create another directory called lab1a:

$ mkdir lab1a

Move your lab1 directory (including its contents) into lab1a:

$ mv lab1 lab1a

lab1a

`-- lab1

`-- cars

Make lab1 your present working directory:

$ cd lab1a/lab1

lab1a

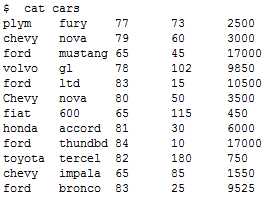
`-- lab1

`-- cars

Within your current directory (lab1), view the complete contents of the file

called cars:

$ cat cars



lab1a

`-- lab1

`-- cars

Copy the file /etc/passwd to your current directory (lab1):

$ cp /etc/passwd .

lab1a

`-- lab1

|-- cars

`-- passwd

Create a subdirectory under your current directory (lab1) called lab1\_sub:

$ mkdir lab1\_sub

lab1a

`-- lab1

|-- cars

|-- lab1\_sub

`-- passwd

Copy the file /etc/group to your lab1\_sub. Your current directory is lab1:

$ cp /etc/group lab1\_sub

lab1a

`-- lab1

|-- cars

|-- lab1\_sub

| `-- group

`-- passwd

Copy your local copy of passwd (from lab1, your current directory) to

lab1\_sub, but make sure this newest copy is named passwd.copy:

$ cp passwd lab1\_sub/passwd.copy

lab1a

`-- lab1

|-- cars

|-- lab1\_sub

| |-- group

| `-- passwd.copy

`-- passwd

Your current directory is lab1. Copy /etc/group to lab1\_sub, naming the

copy group.copy:

$ cp /etc/group lab1\_sub/group.copy

lab1a

`-- lab1

|-- cars

|-- lab1\_sub

| |-- group

| |-- group.copy

| `-- passwd.copy

`-- passwd

Your current directory is lab1. Create a subdirectory under lab1\_sub

called lab1\_sub.copy:

$ mkdir lab1\_sub/lab1\_sub.copy

lab1a

`-- lab1

|-- cars

|-- lab1\_sub

| |-- group

| |-- group.copy

| |-- lab1\_sub.copy

| `-- passwd.copy

`-- passwd

Your current directory is lab1. List detailed info about the non-hidden

files in the lab1\_sub directory:

-----------------------------------------------------------------------------

$ ls -l lab1\_sub

total 12

-rw-r--r-- 1 hhuang66 users 637 May 13 18:21 group

-rw-r--r-- 1 hhuang66 users 637 May 13 18:35 group.copy

drwxr-xr-x 2 hhuang66 users 6 May 13 18:37 lab1\_sub.copy

-rw-r--r-- 1 hhuang66 users 1643 May 13 18:29 passwd.copy

lab1a

`-- lab1

|-- cars

|-- lab1\_sub

| |-- group

| |-- group.copy

| |-- lab1\_sub.copy

| `-- passwd.copy

`-- passwd

Your current directory is lab1. Copy ~/.bash\_history to lab1\_sub,

calling it myHistory:

$ cp ~/.bash\_history lab1\_sub/myHistory

DPS918 Lab 1.5: Create A Directory Structure

Create a Directory Structure

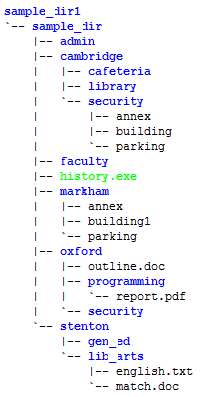
The purpose of this section is to reinforce your skills with issuing UNIX commands covered previously.

You will be asked to create a reasonably complex directory structure. This directory structure will be used in the next section.

WARNING: Use a large screen with a fairly small font so that you can see the entire directory tree, plus the instructions. A size of at least 130 characters by 34 lines

is required.

DPS918 Lab 1.5: Create A Directory Structure



OPEN A NEW SSH WINDOW, and within your home directory, create a directory called sample\_dir1 with the structure as shown above. Directories are in blue. As you

go along, check your progress using this command: ~dps918/2017b/showtree ~/sample\_dir1

You have not created the /home/hhuang66/sample\_dir1/sample\_dir directory, please do so.

Press <Enter> to continue:

Congratulations!

You have successfully completed this section!

DPS918 Lab 1.6: Practice Specifying Path Names

Practice Specifying Path Names

The purpose of this section is to reinforce your skills with issuing UNIX commands covered previously.

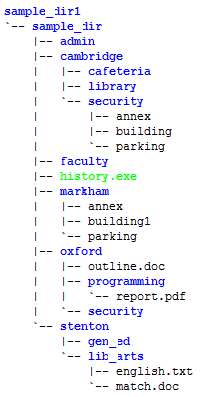
You will be asked to enter a single UNIX command for the following situations, using the directory

structure created in the previous section.

\*\*\* Note that in most cases, only the shortest possible command line will be accepted.\*\*\*

Change to your home directory.

$ cd



sample\_dir1 is a subdirectory of your current (home) directory. Move to sample\_dir using a

relative pathname:

$ cd sample\_dir1/sample\_dir

Your current directory is sample\_dir. Move to security (within cambridge)

using a relative pathname:

$ cd cambridge/security

Your current directory is security. Move to cambridge using a relative pathname:

$ cd ..

Your current directory is cambridge. Move to your home directory using a

relative pathname (don't use cd by itself for this move):

$ cd ../../..

Your current directory is your home directory. Move to stenton using a

relative pathname:

$ cd sample\_dir1/sample\_dir/stenton

Your current directory is stenton. Make a copy of the file named parking

(from the security directory), name it parking2, and place it in your current

directory, using relative pathnames:

$ cp ../cambridge/security/parking parking2

Your current directory is stenton. Remove the security directory, including

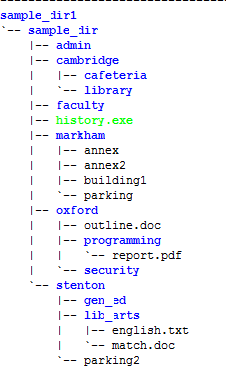
it's contents (no prompting), using a relative pathname:

$ rm -r ../cambridge/security

Your current directory is stenton. Make a copy of the file annex (in markham),

called annex2, and place it in markham:

$ cp ../markham/annex ../markham/annex2



Your current directory is stenton. Move to oxford using a relative-to-home

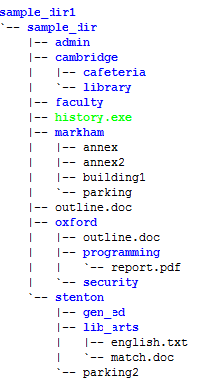
pathname:

$ cd ~/sample\_dir1/sample\_dir/oxford

Your current directory is oxford. Make a copy of the file named outline.doc

and place it into the sample\_dir directory, using relative pathnames:

$ cp outline.doc ..



Your current directory is oxford. Display an 'ls -l' listing of the stenton

directory, using an absolute pathname. Note that your home directory is

/home/hhuang66:

$ ls -l /home/hhuang66/sample\_dir1/sample\_dir/stenton



Your current directory is oxford. Create a new empty regular file named

dir\_practice inside the library directory, using a relative pathname:

$ touch ../cambridge/library/dir\_practice

Congratulations!

You have successfully completed this section!

DPS918 Lab 1.7: Permissions

---------------------------------------------------------------------------------

Available selections:

1 File Permissions

2 Setting Permissions

3 Using umask To Limit Permissions On New Files

4 Review Exercise

Note that the highlighted lines indicate the incomplete sections of

this part of the Lab.

The Review Exercise can only be done when the other sections have

been completed.

File Permissions

Unix systems use a binary value to keep track of the file permissions (security

settings) for each file.

There are three communities of users, and three file permissions which may

be turned ON or OFF for each community.

The communities are:

User (or 'u') - the file's owner.

Group (or 'g') - the file's group.

Other (or 'o') - all other users of the system.

Memorize these communities and the order (u, g, o). Notice that they go from

smallest (1 user) to medium-size (a group of users) to largest (all other

users on the system).

In order to make the 'User' and 'Group' communities mean something, each file

has to be owned by a particular user, and be placed into a particular group.

By default, a file is owned by the person that created it, but this can be

changed (using the 'chown' command).

Every user of a Unix system is in a group. Files created by a user are in

the same group as the user, but that can also be changed (using the 'chgrp'

command).

You can see your current user ID and group ID using the 'id' command. You can

see a file's owner and group using the 'ls -l' command.

Take a look at your user and group IDs.

$ id

uid=36329(hhuang66) gid=500(users) groups=500(users)

Now take a look at the owner and group of your home directory. This information

is in the third and fourth column of detailed listings produced by 'ls -ld'.

Note that you are currently in your home directory.

$ ls -ld

drwx--x--x 40 hhuang66 users 4096 May 15 10:54 .

Each of the communities has three separate permissions which may be turned on

(granted) or turned off (denied).

These permissions are:

Permission: Read Write Execute

Letter: r w x

Value: 4 2 1

Memorize the sequence (r, w, x) and values of these permissions.

Read permission means that a community may read a file. For directories,

this means permission to list the filenames in the directory.

Write permission means that a particular community may place information

in a file or change information that is already in a file (delete, modify,

overwrite). In the case of a directory, this is permission to add, rename,

or delete files listed in that directory.

Execute permission permits a file to be run, if it is a program or script.

For directories, this permission is sometimes called Search or Passthrough,

and it allows files in the directory to be accessed.

Symbolic Representation shown by "ls -l"

Symbolic representation uses three sets of symbols, one per community, with

three symbols in each set, one per permission. A symbolic representation of

a permission takes 9 characters.

The communities are given in u, g, o order, and the permissions are written

in r, w, x order. If a permission is turned on, then the letter for that

permission is shown, otherwise, a dash is used.

For example, the symbolic permission:

---------

...represents all permissions being turned OFF.

This pattern:

rwxrwxrwx

...means that all permissions are turned ON.

This pattern:

rw-r-----

Shows that the User (first three characters) has read and write permission;

users in the Group (middle three characters) have only read permission;

and Others (last three characters) have no permission.

The detailed listings produced by 'ls -l' include the permissions of each

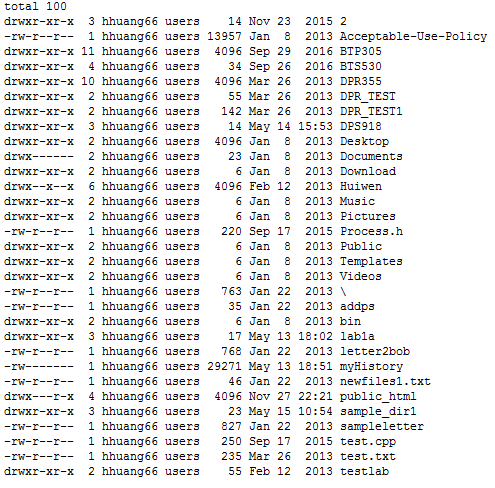
file in the first field, with an extra character at the start which is used

to represent the file type: '-' for a regular file, 'd' for a directory,

and other symbols for other special file types.

Take a look at the permissions of the files in your current directory.

$ ls –l



If the permissions on a file were set so that everyone could read from or

write to the file, how would that be represented symbolically (within the

output of an 'ls -l' command)?

rw-rw-rw-

Good. Try another one:

Write the symbolic representation of a permission setting where the user could

read, write, or execute the file; group members could read or execute only;

and others could only execute the file.

rwxr-x—x

Octal Notation

The other way of representing permissions is to use Octal notation.

Octal uses one digit per community, or three digits to represent the entire

permissions setting.

Remember that in Octal, each digit represents 3 bits. These 3 bits are the

permissions (r, w, x) for a community. Therefore each digit is the sum of the

permission values (4 for r, 2 for w, and 1 for x) for all of the permissions

that are granted to each community.

For example, in the permission:

rwxr-x--x

The first community, User, has been granted r (4) plus w (2) plus x (1)

permission, so the first octal digit is 4+2+1=7.

The second community, Group, has been granted r (4) plus x (1) permission,

so the second octal digit is 4+1=5.

The last community, Other, has been granted x (1) permission, so the last

octal digit is 1.

Putting this together, the octal representation of the permission above would

be written as 751.

Here's another example: a file has '644' permission. This works out to:

User: 6 = 4 (r) + 2 (w) = rw-

Group: 4 = 4 (r) = r--

Other: 4 = 4 (r) = r--

Therefore, 644 in octal notation is equivalent to 'rw-r--r--' in symbolic

representation.

Your turn!

What is the octal equivalent to this symbolic permission:

rwxrw-r—

Now try writing the symbolic representation for this octal permission:

0774

(Note that the extra 0 in the front can be ignored. This octal digit is used

for advanced permissions, including "set user ID", "set group ID",and the

"sticky bit", which we will not cover in this Lab).

Great!

That completes the section on Representing Permissions.

You will now be taken to the main menu. Proceed with Setting Permissions.

DPS918 Lab 1.7: Setting Permissions

Setting Permissions

Permissions are set using the 'chmod' (change mode) command. The syntax for

this command is:

chmod permission file-list

You can list the permission as an octal number, or by using a modified form

of symbolic notation.

For example, to set file permission of 744 (rwxr--r--) on the file 'cows',

you would use the command 'chmod 744 cows'.

Let's experiment. Create an empty file called 'file1'.

$ touch file1

Now set permissions of 761 on 'file1'.

$ chmod 761 file1

Now check the permissions on 'file1'.

$ ls -l file1

-rwxrw---x 1 hhuang66 users 0 May 16 07:28 file1

Good. As you saw, the permissions were set to rwxrw---x (=761).

Now try setting the permissions on 'file1' to 640 using octal notation.

$ chmod 640 file1

Now check the permissions on 'file1' again.

$ ls -l file1

-rw-r----- 1 hhuang66 users 0 May 16 07:28 file1

Using chmod's Symbolic Mode

chmod can also use a variation of symbolic representation. When you are using chmod in this way, the permissions are

specified as:

- communities (u, g, o, or a for all)

- an operation (- to remove, + to add, = to set)

- permissions (r, w, or x)

For example, to add read permission for Other for the file 'file1', you would write:

chmod o+r file1

Here are some more examples:

To add read and write permission for everyone:

chmod ugo+rw file1

or chmod a+rw file1

To remove read permission for the Group:

chmod g-r file1

To set the User permission to rw:

chmod u=rw file1

You can also combine several operations using commas:

To add read permission for User and remove read for Group and Other:

chmod u+r,go-r file1

To add read and execute permission for everyone and remove write permission for Other:

chmod a+rx,o-w file1

Notice that using this technique with + or - does not affect permissions that you do not mention. For example, if you use

'chmod g+r file1', then the permissions for User and Other are not affected, and the write and execute permissions for

Group are not affected.

Use chmod's symbolic representation to add read permission for everyone to 'file1'.

$ chmod a+r file1

Now check the permissions on 'file1'.

$ ls -l file1

-rw-r--r-- 1 hhuang66 users 0 May 16 07:28 file1

Good. Now use chmod's symbolic representation to remove write permission for User from 'file1'.

$ chmod u-w file1

Once again, check the permissions on 'file1'.

$ ls -l file1

-r--r--r-- 1 hhuang66 users 0 May 16 07:28 file1

Remember that 'x' means Passthrough permission on directories? If passthrough permission is turned off, then the contents of the directory and subdirectories is effectively cut off.

This provides a simple way to prevent other people from accessing the contents of your files: simply remove 'x' permission for Group and Other on your home directory (use '~' to represent your home directory).

Enter a command to do this using symbolic notation (it won't be run).

chmod go-x ~

Well done.

Proceed to section 3, Using umask to Limit Permissions on New Files.

Available selections:

1 File Permissions - Completed

2 Setting Permissions - Completed

3 Using umask To Limit Permissions On New Files

4 Review Exercise

Note that the highlighted lines indicate the incomplete sections of this part of the Lab.

The Review Exercise can only be done when the other sections have been completed.

DPS918 Lab 1.7: Using umask To Limit Permissions On New Files

Using umask

When a Unix program creates a file, it tells the kernel what permissions it wants on the new file. These permissions may be limited (restricted) by using a value called 'umask'.

The umask value is set with the 'umask' command.

The umask should be set to the permissions that you do NOT want to grant.

For example, to prevent the system from setting write permission for group, and prevent it from setting ANY permission for other, you would use a umask of 027. Also note that umask only affects NEW files created, not existing files.

Try the example now.

$ umask 027

Good.

Now create the empty file 'testing'.

$ touch testing

Now check the permissions on that file 'testing'.

$ ls -l testing

-rw-r----- 1 hhuang66 users 0 May 16 07:53 testing

Notice that the new file did not have write permission for Group, and had no permission for Other.

When using an FTP client program, the files that you copy to your system are affected by the umask. When you copy from your FTP client to a server, the files are affected by the server program's umask, so it doesn't matter what umask setting you try to use.

That concludes this section.

Proceed to the Review!

Review Exercise

The purpose of this section is to reinforce your skills with issuing UNIX commands covered in the previous sections.

You will be asked to enter a single UNIX command for the following situations. If you are unable to successfully issue the UNIX command after several attempts, a hint will be provided.

Question 1

What is the symbolic representation, as displayed by the 'ls -l' command, for a permission setting for a file where the User can read, write, and execute; the Group can read; and the Other users can't do anything?

rwxr-----

Question 2

What is the octal permission number that represents 'rwxr-----'?

740

Question 3

What is the command to grant all permissions for everyone for the file 'foo' (using octal representation)?

chmod 777 foo

Question 4

What is the command to remove read permission for Group and Other on the file 'green'?

(Hint: Since you don't want to change other permissions, use a symbolic permission instead of an octal permission).

chmod go-r green

Question 5

What is the command to add passthrough permission for everyone for your home directory (~)?

(Hint: Since you don't want to change other permissions, use a symbolic permission again instead of an octal permission).

chmod ugo+x ~

Congratulations!

You have successfully completed the Review Exercise!

Available selections:

1 File Permissions - Completed

2 Setting Permissions - Completed

3 Using umask To Limit Permissions On New Files - Completed

4 Review Exercise - Completed

Note that the highlighted lines indicate the incomplete sections of

this part of the Lab.

The Review Exercise can only be done when the other sections have

been completed.

Total marks obtained is 3 out of 3.

Lab 1 Section 1 was completed

Lab 1 Section 2 was completed

Lab 1 Section 3 was completed

Lab 1 Section 4 was completed

Lab 1 Section 5 was completed

Lab 1 Section 6 was completed

Lab 1 Section 7 was completed

Submission Confirmation

You may now submit Lab 1 by typing 'Yes'. If you do NOT want to submit at this time, type 'm' or 'M'.

Congratulations!

You have successfully submitted Lab 1!

Available selections:

1 Introduction To Unix Commands - Completed

2 Basic Unix Commands - Completed

3 Directory Management - Completed

4 Practice Commands To Create A Directory Structure - Completed

5 Create A Directory Structure - Completed

6 Practice Specifying Path Names - Completed

7 Permissions - Completed

8 Submit DPS918 Lab 1 - Completed - 3 out of 3

The highlighted lines above indicate the incomplete parts of the Lab.

There is a total possible mark of 3, with 0.5 marks deducted for each

incomplete part. Late marks will be deducted at the rate of 0.3 marks (10%)

per day for Labs submitted after midnight of May 19 2017.