

Using UNIX/LINUX – a BCMB/CHEM 8190 Tutorial

Updated (1/1/2020)

Objective: Learn some basic aspects of the UNIX operating system and how to use it.

What is UNIX? Computer operating systems control and manage hardware and software operations of the computer. Most personal computers run Microsoft Windows (for instance, ‘Windows 10’), which is built on an operating system known as Windows NT. Most other computers use some variation of the UNIX operating system. Apple computers (MacOS) run a version known as BSD UNIX, most others run some variation of LINUX (Red Hat, CentOS, Debian, Ubuntu, etcetera). Fortunately, these varieties are very similar, and most of what you need to learn to be able to use any of these is common to them all. Some knowledge of UNIX/LINUX and related subject matter is not only useful to scientists but obligatory. In this tutorial, we will get to know some simple UNIX commands, and otherwise become familiar with UNIX based operating systems. For this tutorial, you’ll need a computer running some version of UNIX. This could be most any Apple computer, or any PC running some version of LINUX (computers running any version of Microsoft Windows will not work).

UNIX Tutorials There are hundreds of on-line UNIX help/tutorial sites, and hundreds of UNIX books for you to purchase (or check out of the library!). Here are links to a few:

<http://www.ee.surrey.ac.uk/Teaching/Unix/>

<https://www.cs.sfu.ca/~ggbaker/reference/unix/>

<https://www.tutorialspoint.com/unix/index.htm>

<https://www.guru99.com/unix-linux-tutorial.html>

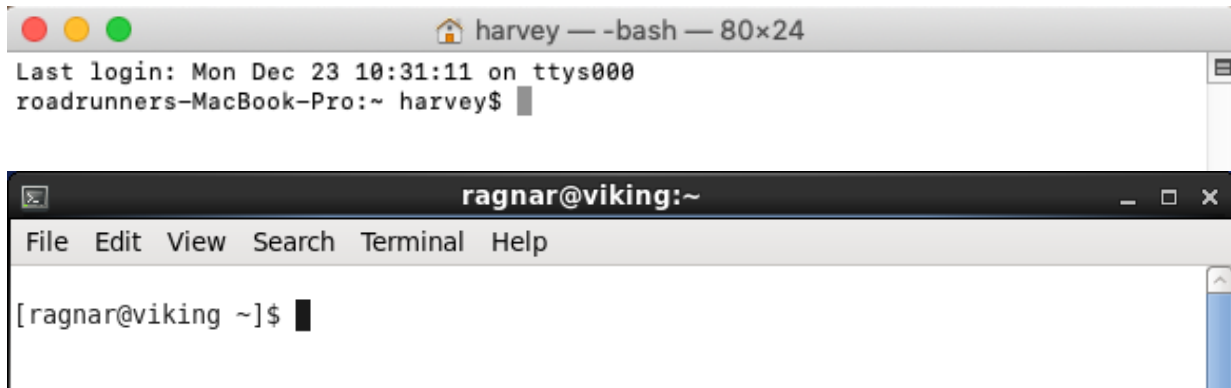
<https://www.softwaretestinghelp.com/unix-tutorials/>

<https://www.softlab.ece.ntua.gr/facilities/documentation/unix/unixintro/>

UNIX Shells Unix “shells” are interfaces for interacting with the UNIX “kernel”. Using typed commands, users can direct the computer to perform specific tasks or functions via the shell and the shell interface. There are many different shells. Probably the most popular these days is called ‘bash’ (“Bourne again shell”), which is a free version (GNU Bash) based on the Bourne shell, but there are many others including C shell (csh / tcsh), Korn shell (ksh), etcetera. The newest version of the Apple operating system, MacOS Catalina, uses the Z shell (zsh). For our tutorial today we will use the **C shell**. Most of the websites linked above will have some explanations about UNIX shells. You’ll find that some programs you might use may require you to use a particular shell, so, it often is important to know what shell you are using. Also, although the most often used commands tend to be common to all shells, some are not.

Terminal We’ll assume that you have access to a computer running some version of UNIX or LINUX, and that there is an account set up on that computer for you, and you are logged into your account. The program (‘application’) that you’ll need to access the shell and enter commands is usually called ‘terminal’. If you are using an Apple (MacOS) computer, you’ll find the ‘terminal’ application in the ‘Utilities’ subfolder in the ‘Applications’ folder. If you are using a computer running LINUX, typically on the main toolbar there will be a ‘Applications’ option,

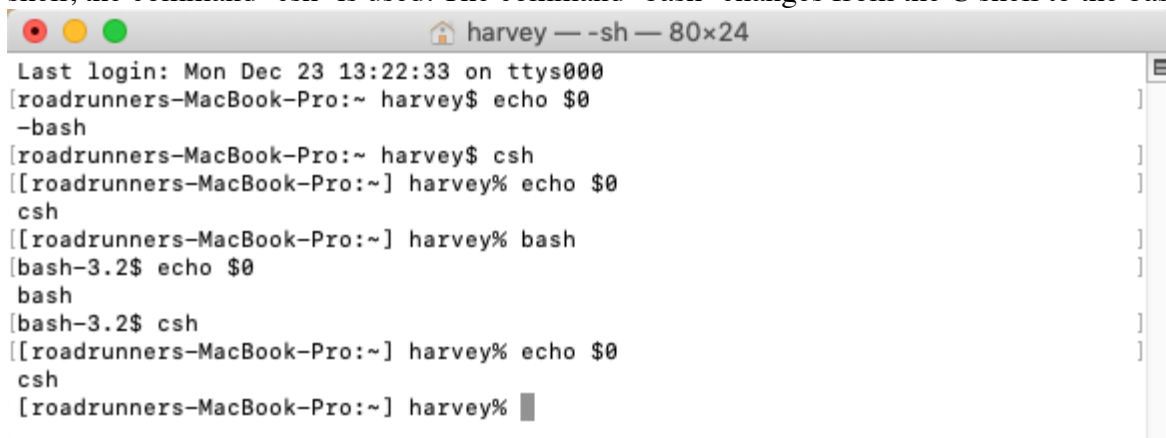
then ‘System Tools’ then ‘Terminal’, although different versions of LINUX will be different (sometimes, ‘Open Terminal’ is found under the ‘File’ menu, sometimes a right-click of the mouse will give an “open in terminal” option). In any case, select the ‘Terminal’ option and a terminal window should appear on the screen:



The top terminal window is from an Apple computer. The user, ‘harvey’, is on a computer/filesystem named ‘roadrunners-MacBook-Pro’. The banner at the top of the window indicates the ‘bash’ shell is being used. The lower window is from a computer running CentOS. The user ‘ragnar’ is on a computer/filesystem named ‘viking’.

Entering Commands: Is UNIX Case Sensitive? Some versions of UNIX are case sensitive, others are not. For instance, in CentOS (and most LINUX operating systems), the command ‘**pwd**’ is NOT the same as pWd, PWd, PWD, and so on. The command ‘**pwd**’ will only work if entered in all lower case. Two files, in the same folder or directory (see below), can have the same name with different capitalization (junk and juNk are both allowed). On most MacOS filesystems, all of these commands (pwd, pWd, PWd, PWD, and so on) are equivalent. However, files junk and juNk cannot both be present in the same folder (the filesystem recognizes them as equivalent). The best practice is to never use any capital letters, any time, with any version of LINUX/UNIX.

Identifying and Changing Shells. For our tutorial, we will use the C shell (‘csh’, ‘tcsh’). So, we need to be able to identify the current shell and to change from one shell to another. To identify the current shell, the command ‘echo \$0’ is used. To change from the bash to the C shell, the command ‘csh’ is used. The command ‘bash’ changes from the C shell to the bash:

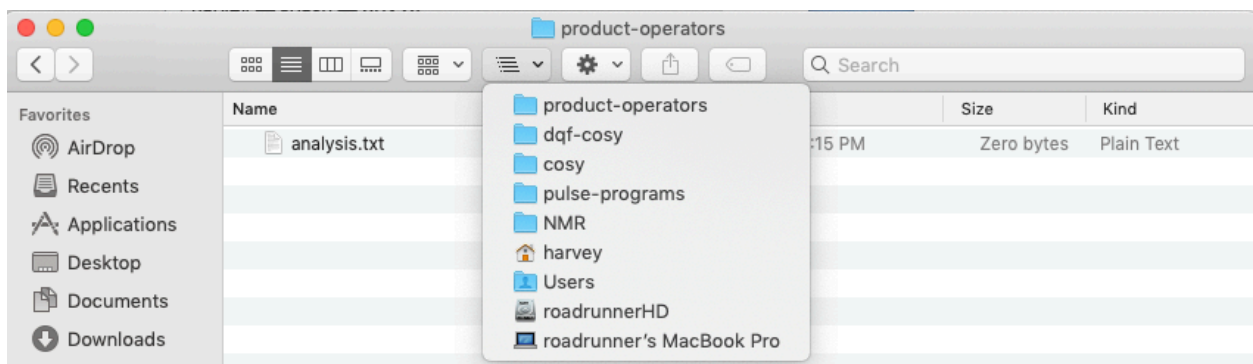


```
ragnar@viking:~  
File Edit View Search Terminal Help  
[ragnar@viking ~]$ echo $0  
/bin/bash  
[ragnar@viking ~]$ csh  
[ragnar@viking ~]$ echo $0  
csh  
[ragnar@viking ~]$ bash  
[ragnar@viking ~]$ echo $0  
bash  
[ragnar@viking ~]$ csh  
[ragnar@viking ~]$ echo $0  
csh  
[ragnar@viking ~]$
```

The top window shows commands entered in a MacOS UNIX window, and the bottom window shows the commands entered in a LINUX (CentOS) window. The output of the ‘echo \$0’ command tells us we either are using a bash shell (output ‘-bash’, ‘bash’, ‘/bin/bash’, etcetera) or a C shell (output ‘csh’). Typing either ‘csh’ or ‘bash’ changes to a C shell or a bash shell, respectively.

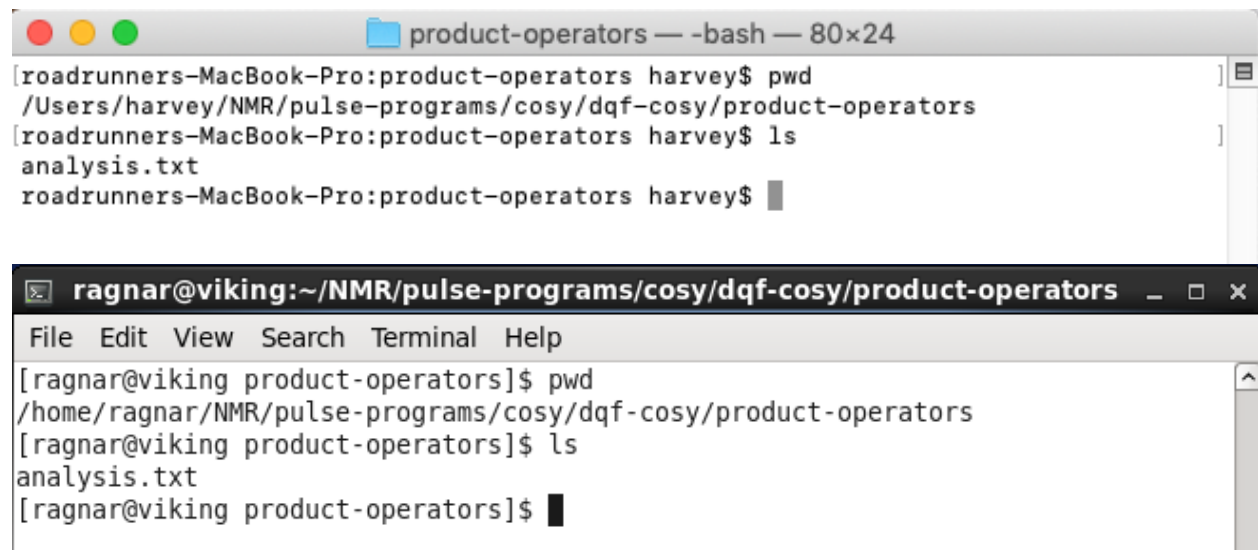
For the remainder of this tutorial, we’ll use a generic prompt (‘[UserName@.....]\$’). This is where commands are entered, as shown in the examples above. For the most part, we won’t distinguish between MacOS UNIX and LINUX. Commands that you (users) will type will appear in **bold**. We’ll note here that, in order to use UNIX/LINUX effectively, you have to remember some commands, however, in order to do most of the important things that you’ll need to do, the number of commands to remember is small.

Directories (Folders) and Contents The modern graphical user interfaces for Apple computers and for modern LINUX computers uses ‘folders’ to store other files and other folders (‘subfolders’). In UNIX/LINUX, the term ‘directory’ is used instead of folder. So, a directory may contain files, or other directories (‘subdirectories’), and so on.



In the example here (MacOS), the current folder is called ‘product-operators’. It contains one file, named ‘analysis.txt’. The ‘product-operators’ folder is in a folder called ‘dqf-cosy’, which is in a folder ‘cosy’, which is in a folder named ‘pulse-programs’, which is in a folder called ‘NMR’, which is in a folder called ‘harvey’ (the home folder for the user ‘harvey’), which is in a folder called ‘Users’ (which includes home folders for all users of the computer). These are on a disk drive called ‘roadrunnerHD’ on the computer ‘roadrunner’s MacBook Pro’. All of this information is called the ‘*path*’ of the file ‘analysis.txt’.

In UNIX/LINUX, identifying the current directory (folder), the contents of the directory, and the path of the contents, are accomplished with the command **pwd** (‘print working directory’) and with the command **ls** (‘list’ the contents of a directory).



The first screenshot shows a macOS terminal window titled 'product-operators — -bash — 80x24'. The prompt is '[roadrunners-MacBook-Pro:product-operators harvey\$]'. The user enters 'pwd', and the output is '/Users/harvey/NMR/pulse-programs/cosy/dqf-cosy/product-operators'. Then the user enters 'ls', and the output is 'analysis.txt'. The second screenshot shows a Linux terminal window titled 'ragnar@vikings:~/NMR/pulse-programs/cosy/dqf-cosy/product-operators'. The prompt is '[ragnar@vikings product-operators]\$'. The user enters 'pwd', and the output is '/home/ragnar/NMR/pulse-programs/cosy/dqf-cosy/product-operators'. Then the user enters 'ls', and the output is 'analysis.txt'.

In the example here, the current directory is ‘product-operators’. Typing the command ‘**pwd**’ returns the path of this directory (all the directories, or folders, and their subfolders or subdirectories). Typing the command ‘**ls**’ shows the contents of the ‘product-operators’ directory.

The commands ‘**pwd**’ and ‘**ls**’ are two of the most often used UNIX/LINUX commands. They will work with any shell. It is good practice to use them generously (as we will do below).

Options/Arguments and Man (manual) Pages. UNIX has a built-in help feature called the **man** pages (for “manual” pages) that helps to explain the commands and the options/arguments available for each of them. Type

```
[UserName@.....]$ man ls
```

The man page(s) for the “**ls**” command will be displayed, one page at a time. You can access each page sequentially by pressing the space bar. You can quit any time by typing **q** (for quit). The options/arguments described for many UNIX commands are accessed by appending a dash (-) followed by (usually) single letter arguments that can be concatenated. Here is an example with four arguments concatenated.

```
[UserName@.....]$ ls -ltFr
```

Please remember there is always a space between the command and the argument or dash. In this case, this command will list the directory contents in the “long” format (**-l**), sorted by when the files/directories were last modified (**-t**), oldest first (**-r**), and with a slash following all directories (**-F**) so that you can distinguish directories from other files. If you don’t have any files or folders in your current directory, then there will be no output. Below we’ll examine how to create new directories/folders and text files.

Files, Directories, and Moving Around. Now we’ll learn how to make new directories and learn how to navigate between directories. First type

```
[UserName@.....]$ pwd
/home/UserName
```

The response should be something like /home/UserName on a LINUX system, or something like /Users/UserName using a MacOS system: this is the current directory (or the *path* to the current directory). Now, we will make a new directory called ‘junk’. The command is “**mkdir**” for “**make directory**”.

```
[UserName@.....]$ mkdir junk
[UserName@.....]$ ls
junk
```

The output should contain an entry “junk”. It may not be clear, just by looking at the output, if “junk” is a file or a directory. The option ‘**-F**’ for the command **ls** causes a backslash (“/”) to be placed after each subdirectory in a directory, but not after files, so that directories/subdirectories can be distinguished from files. Type

```
[UserName@.....]$ ls -F
junk/
```

The entry “junk” should be followed by a “/”, indicating that it is a directory.

In order to change from one directory to another, the UNIX/LINUX command ‘**cd**’ (**c**hange **d**irectory) is used. Now we’ll change the current directory to the “junk” directory (we’ll “go into” the “junk” directory).

```
[UserName@.....]$ cd junk
```

Now, type

```
[UserName@.....]$ pwd
/home/UserName/junk
```

The output should be something like /home/UserName/junk using LINUX and /Users/UserName/junk using MacOS UNIX.

```
[UserName@.....]$ ls
```

Because there are no files or directories in the “junk” directory, there should be no output.

So, how do we get “back” to the previous directory? There are two ways, both using the “cd” command. First, in UNIX you can change to ANY directory simply by using the “cd” command followed by the *complete path* of the directory (something like “/home/UserName” using LINUX, or “/Users/UserName” using MacOS UNIX) So, type

```
[UserName@.....]$ cd /home/UserName or /Users/UserName
[UserName@.....]$ pwd
/home/UserName
```

The output should be ‘/home/UserName’ using LINUX, ‘/Users/UserName’ using MacOS UNIX.

From here to the end of the tutorial, we’ll assume that you are using a LINUX system and your home directory is “/home/UserName”. If you are using a MacOS UNIX system, your home directory will be something like “/Users/UserName”. So, for the rest of the tutorial, if you are using MacOS UNIX, simple substitute “/Users/UserName” for “/home/UserName”.

What you will find is that, under normal circumstances, it requires a lot of typing to always enter the complete path to any directory for simple directory changes. In our case, we simply want to change from a subdirectory to the directory that contains it (in the hierarchical directory structure). In this case, we can simply type

```
[UserName@.....]$ cd .. or [UserName@.....]$ cd ../
```

(these two commands do the same thing). This allows us to move “up” or “back” one directory (*from /home/UserName/junk to /home/UserName*).

Now, you should be back in the original directory (/home/UserName). Type

```
[UserName@.....]$ cd junk
```

Now, you should be in the “junk” directory again. Type

```
[UserName@.....]$ pwd
```

The output should be “/home/UserName/junk”. Now, make a new directory in the “junk” directory called “test”

```
[UserName@.....]$ mkdir test
[UserName@.....]$ cd test
[UserName@.....]$ pwd
/home/UserName/junk/test
```

The output should be “/home/UserName/junk/test”. Now, go back to the original directory (“/home/UserName”). Type

```
[UserName@.....]$ cd ../../
[UserName@.....]$ pwd
/home/UserName
```

The output should be “/home/UserName”. So, from “/home/UserName/junk/test”, to return to the original directory, one can use the command “**cd ../**” to move to “/home/UserName/junk”, and then “**cd ../**” again to move to /home/UserName”, or go directly using “**cd ../../**”.

Getting Tired of Typing? There are several useful shortcuts. One is that the up arrow will retrieve the previous command and place it on the current command line. You can then move along the text with the right and left arrows, delete with the backspace key, and type in new text – very convenient if you make a mistake in typing a command line. You can also use the mouse to highlight text anywhere on the screen. Hitting the center mouse button then pastes that text to the point where your command line cursor sits.

Creating and Editing Files We will now learn how to make text files and edit them. Modern graphical interfaces include many very nice text editors. UNIX, however, has some built in text editors that are uniformly available in (most) all types of UNIX, so whatever type of workstation you sit down at, these should be available. One of them is called **vi**. This is a really good editor for you to learn to use, but it is not easy to learn, and takes a lot of time to do so.

If you are using a LINUX operating system, it will likely include a built-in, graphical text editor that is named **gedit**, which is quite easy to use, and can be accessed either as a command (‘gedit’) typed in a terminal window, or from the graphical interface. Likewise MacOS has a built-in graphical text editor called **TextEdit** that can be accessed from the ‘Applications’ folder, and is often found on the Dock.

Typically, both LINUX and MacOS will have a bare-bones, command-line text editor called ‘**nano**’, which is an enhanced, free version of an editor called ‘pico’. In this tutorial, our goal is not to learn all about how to use this editor, but simply to use it to create some files.

So, first, go to the “junk” directory.

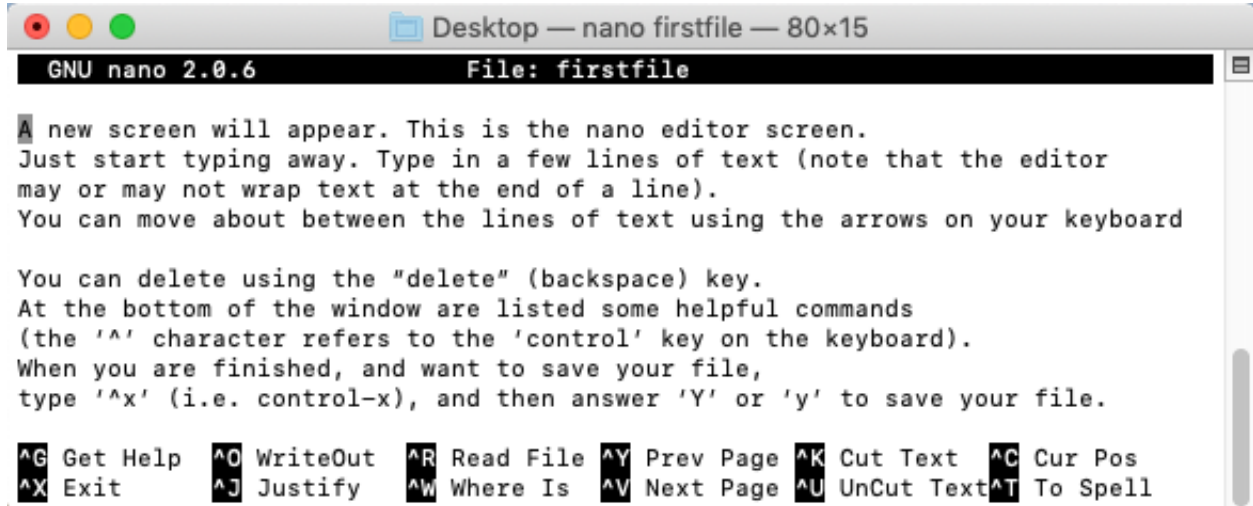
```
[UserName@.....]$ cd /home/UserName/junk
[UserName@.....]$ pwd
/home/UserName/junk
```

Now we will create a file called “firstfile” type

```
[UserName@.....]$ nano firstfile
```

A new screen will appear (see below). This is the **nano** editor screen. Just start typing away. Type in a few lines of text (note that the editor may or may not wrap text at the end of a line).

You can move about between the lines of text using the arrows on your keyboard (←, ↑, →, ↓). You can delete using the “delete” (backspace) key. At the bottom of the window are listed some helpful commands (the ‘^’ character refers to the ‘control’ key on the keyboard). When you are finished, and want to save your file, type ‘^x’ (i.e. control-x), and then answer ‘Y’ or ‘y’ or ‘yes’ to save your file.



```
GNU nano 2.0.6 File: firstfile

A new screen will appear. This is the nano editor screen.
Just start typing away. Type in a few lines of text (note that the editor
may or may not wrap text at the end of a line).
You can move about between the lines of text using the arrows on your keyboard

You can delete using the "delete" (backspace) key.
At the bottom of the window are listed some helpful commands
(the ^ character refers to the ^control key on the keyboard).
When you are finished, and want to save your file,
type ^x (i.e. control-x), and then answer Y or y to save your file.

^G Get Help  ^O WriteOut  ^R Read File  ^Y Prev Page  ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is   ^V Next Page  ^U UnCut Text ^T To Spell
```

Now type

```
[UserName@.....]$ ls -F
firstfile  test/
```

Your “junk” directory should now contain one file called “firstfile” and a subdirectory called “test”. The ‘test’ subdirectory should be followed by a backslash (‘/’) to indicate it is a directory.

Now edit “firstfile” using the ‘nano’ editor. Your file should open in the nano editor window.

```
[UserName@.....]$ nano firstfile
```

You should see the file that you created earlier. Make some changes in your file. Then save your changes by typing ‘^x’ as before. A prompt will ask you what you would like the filename to be; the default will be ‘firstfile’, but, if you want to make a new file, you can type in a new name. So, enter the name “**secondfile**” when nano asks you for the file name. Now go to the still active command window and type

```
[UserName@.....]$ ls -F
firstfile  secondfile  test/
```

You should see 2 files, “firstfile” and “secondfile” (and one subdirectory, “test”), and possibly a file named ‘~firstfile’. This latter file is a temporary save in case you inadvertently overwrite a file.

Copying, Moving, and Renaming Files and Directories Now we will learn some basics of copying and deleting files and directories. The current directory should be /home/UserName/junk. If not, go to that directory by typing

```
[UserName@.....]$ cd /home/UserName/junk
```

List the directory contents, discriminating between files and directories, by typing

```
[UserName@.....]$ ls -F  
firstfile          secondfile        test/
```

There should be two files (“firstfile” and “secondfile”) and one directory (“test”). Now, lets make a copy of “firstfile”. We’ll call it “thirdfile”. LINUX/UNIX has a command ‘**cp**’ for ‘copy’, that can make a copy of a file, and give the copy a new name.

```
[UserName@.....]$ cp firstfile thirdfile
```

This command makes an identical copy of “firstfile” and gives it the name “thirdfile”. You can verify this if you like by using the **nano** editor to look at thirdfile. Now, make a copy of “secondfile”, and call it “fourthfile”.

```
[UserName@.....]$ cp secondfile fourthfile  
[UserName@.....]$ ls -F  
firstfile          fourthfile        secondfile        test/    thirdfile
```

UNIX/LINUX has a very nice command called ‘**mv**’, presumably for ‘move’. This command can be used to move a file from one directory to another, it can be used to move a subdirectory to another directory, it can be used to rename a file or directory, etcetera. It is one of the most useful and powerful commands. First, we’ll use the ‘**mv**’ command to move the file called “fourthfile” out of the current directory and into the “test” directory. We can do this with either of the following commands

```
[UserName@.....]$ mv fourthfile /home/UserName/junk/test
```

or

```
[UserName@.....]$ mv fourthfile ./test
```

The first of these defines the destination directory path fully and explicitly, whereas the second defines it relative to the current directory (“./” indicates that the directory that follows is a subdirectory of the current directory). While we’re at it, let’s move the file called “thirdfile” into the test directory also

```
[UserName@.....]$ mv thirdfile ./test
```

Now, it is important to discuss good practices. As you’ve guessed by now, it is critical always to know what the current directory is, and what the contents are. Once you use a command like

‘mv’ or ‘cp’ or ‘cd’, you will want to make sure that the result is what you expected. It is VERY easy to make mistakes typing, so you’ll want to always check yourself. If you do, you will avoid many types of common problems. One of the best ways to avoid common mistakes is to use the commands ‘pwd’ and ‘ls’ VERY liberally. We’ll illustrate this below.

The current directory should still be ‘/home/UserName/junk’. There should be two files (‘firstfile’ and ‘secondfile’) and one subdirectory (‘test/’) in the ‘junk’ directory. You moved the files ‘thirdfile’ and ‘fourthfile’ to the subdirectory ‘test’. So, let’s check to ensure that our expectations are correct, and, once we are finished, let’s make sure the ‘junk’ subdirectory is the current directory and let’s confirm this.

```
[UserName@.....]$ pwd
/home/UserName/junk

[UserName@.....]$ ls -F
firstfile      secondfile    test/

[UserName@.....]$ cd ./test
[UserName@.....]$ pwd
home/UserName/junk/test

[UserName@.....]$ ls -F
thirdfile      fourthfile

[UserName@.....]$ cd ../
[UserName@.....]$ pwd
/home/UserName/junk

[UserName@.....]$ ls -F
firstfile      secondfile    test/
```

Now, let’s move “thirdfile” back to /home/UserName/junk

```
[UserName@.....]$ pwd
/home/UserName/junk

[UserName@.....]$ ls -F
firstfile      secondfile    test/

[UserName@.....]$ cd test
[UserName@.....]$ pwd
home/UserName/junk/test

[UserName@.....]$ ls -F
thirdfile      fourthfile

[UserName@.....]$ mv thirdfile ../
[UserName@.....]$ ls -F
fourthfile
```

```

[UserName@.....]$ cd ../
[UserName@.....]$ pwd
home/UserName/junk

[UserName@.....]$ ls -F
firstfile      secondfile    test/        thirdfile

```

So far, we've used commands like 'ls' and 'mv' to operate on files in the current directory. However, 'ls', for instance, can be used to list the contents of any directory, and 'mv' can be used to move any file from any directory to any other directory.

For instance, if the current directory is '/home/UserName/junk', the contents of the subdirectory 'test' can be listed.

```

[UserName@.....]$ pwd
home/UserName/junk

[UserName@.....]$ ls -F
firstfile      secondfile    test/        thirdfile

[UserName@.....]$ ls -F ./test
fourthfile

[UserName@.....]$ ls -F /home/UserName/junk/test
fourthfile

```

Now, let's use the 'mv' command to rename the directory called "test". We will give it the new name "bob".

```

[UserName@.....]$ mv test bob    or    mv ./test ./bob

[UserName@.....]$ ls -F
firstfile      secondfile    bob/         thirdfile

[UserName@.....]$ ls ./bob
fourthfile

```

The last commands will show that the directory "test" is now called "bob", and that the directory "bob" contains the single file "fourthfile".

```

[UserName@.....]$ ls -F
bob/           firstfile      secondfile    thirdfile
[UserName@.....]$ ls ./bob
fourthfile
[UserName@.....]$

```

The ‘**cp**’ command can be used to make copies of directories and their contents. To copy directories, the option ‘**-r**’ must be used. Here we’ll make a copy of the “bob” directory. We’ll call it “obo”.

```
[UserName@.....]$ cp -r bob obo or cp -r ./bob ./obo
[UserName@.....]$ ls -F
bob/                firstfile          obo/                secondfile          thirdfile
```

For the next exercise make one more copy of “bob”.

```
[UserName@.....]$ cp -r bob boo
[UserName@.....]$ pwd
home/UserName/junk

[UserName@.....]$ ls -F
bob/                firstfile          secondfile
boo/                obo/              thirdfile
```

Removing / Deleting Files and Directories In Windows, and in other nice, friendly interfaces like the MacOS graphical interface, and some LINUX graphical interfaces, there is a trash can where you can dispose of the files and directories that you don’t want any more. If you decide later that you really don’t want to throw them away, you can go and retrieve them from the trash. This is a nice and safe way to deal with deleting files. In Unix, there is no nice safe way. *There is no trash can, there is no “undo”, there is no safety net. Once you remove a file, it is gone forever, never to return. You cannot recover it. Don’t forget this. Be careful.*

Make sure you are in the directory /home/UserName/junk.

```
[UserName@.....]$ pwd
home/UserName/junk

[UserName@.....]$ ls -F
bob/                firstfile          secondfile
boo/                obo/              thirdfile
```

You should see the three directories “bob”, “obo”, and “boo”. We are going to delete two of these. There are two commands for removing/deleting files and directories, “**rm**” (i.e. “**remove**”), and “**rmdir**”. The first, “**rm**” will remove files or directories (appropriate options/arguments are necessary to delete directories). The second, “**rmdir**”, will remove directories that are empty (contain no files or subdirectories/folders). Let’s test this latter fact.

```
[UserName@.....]$ rmdir obo
rmdir: failed to remove 'obo': Directory not empty

[UserName@.....]$ ls -F
bob/                firstfile          secondfile
boo/                obo/              thirdfile
```

Because ‘rmdir’ can only remove directories/folders that are empty (contain no files or subdirectories/folders), you get an error message saying “Directory not empty”.

So, first we’ll remove the files in the ‘obo’ directory, then we’ll remove the directory. We could do this by first using ‘cd’ to go into the ‘obo’ directory, then remove the file that’s there, and then use ‘cd’ to go back to the ‘junk’ directory. However, this time we’ll simply perform all operations from the ‘junk’ directory (for practice).

```
[UserName@.....]$ pwd
home/UserName/junk

[UserName@.....]$ ls -F
bob/          firstfile      secondfile
boo/          obo/          thirdfile

[UserName@.....]$ ls -F ./obo
fourthfile

[UserName@.....]$ rm ./obo/fourthfile

[UserName@.....]$ ls -F ./obo
```

So, the last ‘ls -F ./obo’ command returns a blank line, indicating that there are no files remaining in the ‘obo’ directory. Now, the folder ‘obo’ can be removed with ‘rmdir’

```
[UserName@.....]$ rmdir obo
[UserName@.....]$ ls -F
bob/          boo/          firstfile      secondfile      thirdfile
```

You can remove directories that are *not* empty using “rm” with the “-r” option. Like the “cp” command, the “-r” option indicates that a recursive operation is to be performed on a directory (and its contents). We will remove the directory “boo” and its contents with this command.

```
[UserName@.....]$ pwd
home/UserName/junk

[UserName@.....]$ ls -F
bob/          boo/          firstfile      secondfile      thirdfile

[UserName@.....]$ ls -F ./boo
fourthfile

[UserName@.....]$ rm -r boo    or    rm -r ./boo

[UserName@.....]$ ls -F
bob/          firstfile      secondfile      thirdfile
```

You’ll see that the “boo” directory (and its contents) is gone.

Examining File Contents You have seen that text editors like nano can be used not only to create new files and to view and edit the contents of files that have already been created. UNIX provides other means also to quickly view or examine the contents of files, but not edit them. One of these, ‘**cat**’, simply lists the entire contents of a file to the screen. Another commands, ‘**more**’, lists the contents to the screen one page at a time.

Make sure you are in the directory /home/UserName/junk. We’ll use the ‘**cat**’ command to list the contents of the file ‘firstfile’.

```
[UserName@.....]$ pwd
home/UserName/junk
```

```
[UserName@.....]$ ls -F
bob/          boo/          firstfile     secondfile    thirdfile
```

```
[UserName@.....]$ cat firstfile
```

A new screen will appear. This is the nano editor screen. Just start typing away. Type in a few lines of text (note that the editor may or may not wrap text at the end of a line). You can move about between the lines of text using the arrows on your keyboard

You can delete using the “delete” (backspace) key. At the bottom of the window are listed some helpful commands (the ‘^’ character refers to the ‘control’ key on the keyboard). When you are finished, and want to save your file, type ‘^x’ (i.e. control-x), and then answer ‘Y’ or ‘y’ to save your file.

Note that the output of the command is just a listing of the contents of ‘firstfile’. The ‘**cat**’ command is good for quickly dumping the contents of (short) files to the screen to get a quick look at what is in them. It is not so good, usually, for long files. The ‘**more**’ command is generally better for longer files that you would like to examine in detail. It allows you to examine the contents of large text files one page at a time, like the manner in which the “man” pages are displayed. If the file is more than one page, pressing the **space bar** will scroll to the next page. Typing ‘**q**’ (quit) will exit the command. You can try this command with the ‘firstfile’ file. You’ll see the entire contents of the file on the first page (because the file is short).

UNIX has a really nice, useful command called “**grep**” that allows you to search through files in a directory for words or strings. The command replies with a list of all the lines in a file containing the requested word or string. For instance, “firstfile” contains the word “delete” (it contains this word twice). If we wish to know if a file contains a particular word, or part of a word, or a phrase, grep will find it.

```
[UserName@.....]$ pwd
home/UserName/junk
```

```
[UserName@.....]$ ls -F
bob/          boo/          firstfile     secondfile    thirdfile
```

```
[UserName@.....]$ grep delete firstfile
```

You can delete using the "delete" (backspace) key.

```
[UserName@.....]$ grep 'rap te' firstfile
```

may or may not wrap text at the end of a line).

```
[UserName@.....]$ grep 'save your file' firstfile
```

When you are finished, and want to save your file,

type '^x' (i.e. control-x), and then answer 'Y' or 'y' to save your file.

Finding Files Graphical interfaces such as Microsoft Windows, MacOS, and some LINUX graphical interfaces have nice utilities for finding files on disks. For UNIX/LINUX based operating systems, these are often based on the UNIX "**find**" command. We will demonstrate how to use the UNIX "**find**" command to find files.

First, navigate to your home directory using the '**cd**' command, and then confirm it with the '**pwd**' command.

```
[UserName@.....]$ cd /home/UserName
```

```
[UserName@.....]$ pwd
```

home/UserName

Now we'll use the '**find**' command. We know that in the directory '/home/UserName/junk' is a file named 'firstfile'. We'll use the 'find' command to confirm that. We want to search all subdirectories of our current directory ('./'), we will search for a file that has a particular name ('-name'), which is '**firstfile**', and we will ask that the command print the result (location of the file) to the screen ('-print').

```
[UserName@.....]$ find ./ -name firstfile -print
```

./junk/firstfile

So, in this case, we confirm that the file 'firstfile' is located in the directory 'junk', which is a subdirectory of /home/UserName/.

When using the 'find' command, it is often the case that one is searching for groups of files with similar names (parts of the name in common), and it is often the case that the full name of the file is not known, but part of it is. In these cases, it is useful to use "wildcard" characters in the search. Later (below), we'll discuss in more detail the general use of "wildcards" in UNIX/LINUX. Here we'll introduce the concept here using the '**find**' command. Let's say we want to find all files in any subdirectories of our home directory that have '**fil**' in the name. Here, then, the name that we'll search for is '***fil***', where the asterisks ('*') are the 'wildcards', each representing any string of characters.

```
[UserName@.....]$ cd /home/UserName
```

```
[UserName@.....]$ pwd
```

home/UserName

```
[UserName@.....]$ find ./ -name '*fil*' -print
./junk/bob/fourthfile
./junk/thirdfile
./junk/firstfile
./junk/boo/fourthfile
./junk/secondfile
```

So, in this case, any files in the path that begins “/home/UserName” (i.e. “./”) that have the string “**fil**” in the name, will be found, and the result printed to the screen. This is what we observe in the output shown above.

The “**find**” command is a very powerful one. Consult the man pages (“**man find**”) for lots of options and also examples.

Wildcards In UNIX/LINUX, the wildcard character (the asterisk, “*”) can substitute for any character or string of characters. It is extremely useful, and also can be very dangerous.

Once you finish this tutorial, you might want to experiment with using some wildcards with certain commands. For instance, “**ls ***” will list the names of all files in a directory, the names of all the subdirectories, and the contents of all subdirectories. The command “**mv * ../**” will move the contents of the current directory to the directory above it in the hierarchy.

Arguably one of the most dangerous commands in UNIX/LINUX is “**rm -r ***”). This command will erase (remember, permanently, there is no safety net) all files and all subdirectories and their contents, in the current directory. This command is probably the cause of more loss of more important information (experimental results, critical files) than any other. Hopefully you will never have the experience of deleting a week’s worth (or year’s worth) of data or results because you either used this command incorrectly or inadvertently. Be very careful using wildcards with the “**rm**” command.

Directing Input and Output In UNIX, the output from one command can be directed in particular ways. Above, using the “**find**” command, we demonstrated that we can specifically direct the output to the screen. In addition, output can be directed into a file, or it can be directed to serve as the input for another command. These are usually referred to as “redirection” and “piping” or “pipes”, respectively. These are very powerful aspects of UNIX and are often used to reduce human intervention (and workload).

First, we’ll direct the output (referred to in UNIX as STDOUT, i.e. “standard output”) from a simple command into a file. For instance, we’ll list the contents of our directory and put it into a file, rather than display the output on the screen. This is accomplished using the redirector symbol “>”. We will list the contents of our directory using “**ls -F**”, and, rather than have it printed to the screen, we’ll redirect it into a file, which we’ll call (arbitrarily) “junklist”. Then we’ll use the ‘cat’ command to display the contents of ‘junklist’ to confirm the contents.

```
[UserName@.....]$ cd /home/UserName/junk
[UserName@.....]$ pwd
home/UserName/junk
```



```

[UserName@.....]$ ls -F > junklist
[UserName@.....]$ cat junklist
bob/
boo/
firstfile
junklist
secondfile
thirdfile

[UserName@.....]$ ls -F
bob/          firstfile      secondfile
boo/          junklist       thirdfile

```

So, the command created a file called ‘junklist’ that lists the contents of the subdirectory ‘junk’, which now includes the file ‘junklist’.

The redirector “>>” allows you to append output to an existing file. Below we use this redirector to append the output of the “ls -F” command to the existing file ‘junklist’. We then use the ‘cat’ command to confirm that everything in the ‘junklist’ file is listed twice, the result of appending the output to what already existed in the file.

```

[UserName@.....]$ ls -F >> junklist
[UserName@.....]$ cat junklist
bob/
boo/
firstfile
junklist
secondfile
thirdfile
bob/
boo/
firstfile
junklist
secondfile
thirdfile

```

Now we’ll use a UNIX “pipe” to use the output of one command as the input for the next, and then we’ll direct the result of the second command into a file. We’ll make a list of all the files in the ‘junk’ subdirectory that have the letter ‘t’ in the filename (“ls *t*”), then we’ll “pipe” (i.e. forward) that information using the “pipe” director (the verticle line character, “|”) to the “grep” command. Using “grep”, we’ll find any/all instances of the string “answer” (in the files with ‘t’ in their filenames). Then, as above, we’ll redirect (“>”) that result into a file called ‘result’.

```

[UserName@.....]$ cd /home/UserName/junk
[UserName@.....]$ pwd
home/UserName/junk

[UserName@.....]$ ls -F
bob/          firstfile      secondfile
boo/          junklist       thirdfile

```

```
[UserName@.....]$ ls *t* | grep answer * > result
```

```
[UserName@.....]$ cat result
```

```
firstfile:type '^x' (i.e. control-x), and then answer 'Y' or 'y' to save  
your file.
```

```
secondfile:type '^x' (i.e. control-x), and then answer 'Y' or 'y' to save  
your file.
```

```
thirdfile:type '^x' (i.e. control-x), and then answer 'Y' or 'y' to save  
your file.
```

```
[UserName@.....]$ ls -F
```

```
bob/      firstfile      result          thirdfile  
boo/      junklist       secondfile
```

The file “result” will contain all instances of the occurrence of the string “answer” in files in your directory that have a ‘t’ in their name (for instance, “answer” should appear in the files “firstfile”, “secondfile”, and “thirdfile”).

Aliases Are you tired of typing “ls -F” all the time? Maybe you would like to minimize keystrokes, or maybe you are tired of trying to remember certain commands, etcetera. Well, UNIX allows you to make up your own commands with the “alias” command. For instance, maybe every time you want to list your directory contents you want the output to be consistent with “ls -F” but you don’t want to type “ls -F” all the time, maybe you would just like to type “l” (lower case “L”). Here we set up the alias and demonstrate its use. Please note that the syntax for the alias command shown is for the C shell (csh). If you are using the bash shell, the syntax is different (alias l=“ls -F”).

```
[UserName@.....]$ cd /home/UserName/junk
```

```
[UserName@.....]$ pwd
```

```
home/UserName/junk
```

```
[UserName@.....]$ ls -F
```

```
bob/      firstfile      result          thirdfile  
boo/      junklist       secondfile
```

```
[UserName@.....]$ alias l 'ls -F' (“l” is lower case L, not “one”)
```

```
[UserName@.....]$ l
```

```
bob/      firstfile      result          thirdfile  
boo/      junklist       secondfile
```

Now, simply typing “l” will execute “ls -F”. You can alias any commands you like. These alias definitions will disappear when you log off and you will have to re-enter them next time. There is a way to keep the aliases so that they are ‘permanently’ defined. You can add your alias definitions to a file that is executed each time you log in. For the C shell (csh) it is called the “.cshrc” file or the “.tcshrc” file. For the bash shell, it is usually called the “.bash” file. In any case, beginners should consult more advanced users for modifying these files.

Monitoring and Terminating Processes When you open a terminal (a Bourne shell, C shell, or a K shell), you begin a “process”. Likewise, if you start a program running from the shell, a new process is created. You can monitor the processes that are running on your computer. You can stop the ones that belong to you. Processes are monitored using the “ps” command.

```
[UserName@.....]$ ps -u UserName
```

This command displays the processes of the current user (you). If you look at the output, on the right side is the “command”, i.e. in this case it will tell you that you are using something like a “csh” or a “bash” (C shell or bash). On the left the output tells who you are and the PID (“process ID”). The PID is important if you want to end a process. If you need to end a process, you would use the “kill” command. For instance you would type (don’t actually type this)

```
[UserName@.....]$ kill -9 PID
```

where PID is the “process identification number”. The “-9” option guarantees a “sure kill”.

If you want to see what *everyone* is doing, you can type

```
[UserName@.....]$ ps -ef.
```

Periodically, a UNIX/LINUX process will get “stuck”, and the only way to stop it is to kill it. In order to do so, you will have to use the ‘ps’ command to identify it, and then the ‘kill’ command to stop it. These are somewhat advanced topics, so if you intend to use UNIX/LINUX a lot you will have to learn more about them.

Some Miscellaneous Commands There are many, many UNIX commands. Some are quite useful and powerful.....some are there to add certain conveniences.....some are there for no apparent reason. Here are some miscellaneous commands:

```
[UserName@.....]$ passwd (change your password)
[UserName@.....]$ clear (clears the terminal window)
[UserName@.....]$ whoami (tells you who you are logged in as)
[UserName@.....]$ who (tells you everyone who is logged in)
[UserName@.....]$ date (returns the day, date, time and year)
[UserName@.....]$ cal (displays a calendar for the current month)
```

Quitting a Terminal Window Typically, typing “exit” will end the terminal session and, depending on the operating system, may also close the window, or at least render it inactive.