

INFRASTRUCTURE TECHNOLOGIES
CIS-350
Unix/Linux – Simple Command Line Interface (CLI)
Lab 3

This Lab is worth 50 points. The questions in the Lab 3 Report that you need to answer are worth 25 points. You **must** log in to Ubuntu Linux and actually work the lab commands on the Ubuntu Linux system. The hands-on work is worth remaining 25 points.

Due date: **See Blackboard**

UNIX/Linux Commands and Utilities

WHEN YOU FINISH WORKING THIS LAB YOU SHOULD BE ABLE TO:

- Know basic UNIX/Linux commands.
- Describe the general form of a UNIX/Linux command.
- Briefly describe a hierarchical directory structure.
- Distinguish between a path name and a file name.
- Distinguish between the root directory, your home directory, and your working directory.
- Explain the significance of the (.) and (..) file names.
- Distinguish between creating a directory and creating a file.
- Change working directories.
- Explain how wild-card characters are used.
- Briefly explain redirection.
- Explain filters and pipes.
- Briefly explain what a shell script is.
- Explain the advantage of running selected programs in the background.

UNIX

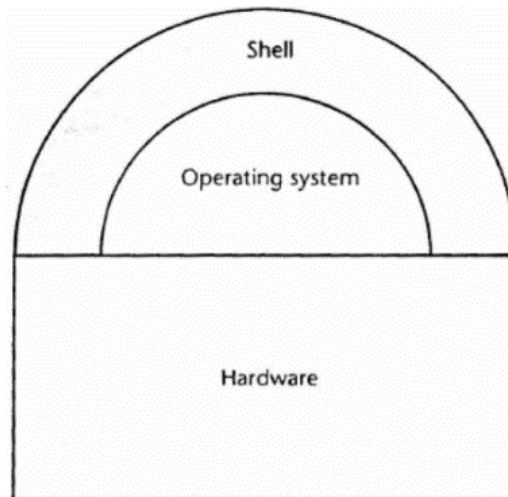
UNIX was developed at Bell Laboratories, a division of AT&T, in the 1970s. Largely the work of three individuals, Ken Thompson, Dennis Ritchie, and Brian Kernighan, the system's main thrust was providing a convenient working environment for programming. Today, UNIX is an important standard that has influenced the design of many modern operating systems. For example, MS-DOS, the predecessor of the Windows operating system, and then Windows incorporated numerous UNIX features, such as piping, input/output (I/O) redirection, and upside-down hierarchical tree directory structure. Experienced programmers consider UNIX simple, elegant, and easy to learn. Beginners, on the other hand, sometimes find it terse, cryptic, and not very friendly.

Linux

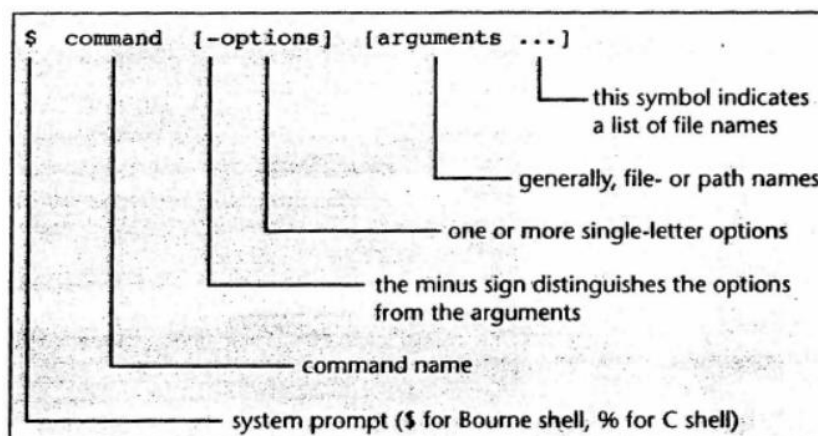
In 1991, Linus Torvalds, then a graduate student at the University at Helsinki, created a version of UNIX to run on the Intel 386 chip. He released the source code on the Internet as Linux. Since then it has been refined and modified and today incorporates contributions from hundreds of software developers around the world. Linux is open-source software. Openness implies that the complete source code is freely available and the operating system can be extended through common interfaces. Today Linux has been ported to run on most platforms including the personal computer. The initial stable version of the Linux, kernel 1.0, was released in 1994. Kernel refers to the core of the operating system and not the applications such as the shell, compilers, and other programs that run on the kernel. Commercial vendors such as RedHat, Caldera, Suse, and Mandrake have taken these stable versions and packaged the software for easy installation on a wide variety of personal computers and added other enhancements that make Linux useful. There are also open source distributors such as **Ubuntu**, Debian, and Gentoo that have packaged the software for distribution. In all Linux labs we will be using **Ubuntu Linux**.

The UNIX/Linux Shell

UNIX/Linux commands are processed by a **shell** that lies between the user and the resident operating system. (See the figure on the next page.) The shell is not really part of the operating system, so it can be changed. Professional programmers might choose a technical shell. Beginners might prefer selecting commands from a menu or pointing at pictures (icons). The idea of a command processor that is independent from the operating system was an important UNIX/Linux innovation. Four shells are in common use. The standard shell, sometimes called the **Bourne shell**, was developed at Bell Laboratories. A second, the **C shell**, is related to the C programming language. The **Korn shell** is a combination of the Bourne and C shells. When you login to the Ubuntu Linux you will be in the **Bourne Again shell (Bash shell)**. The four shells are similar. In fact, you can complete all UNIX/Linux labs using any of these shells. As you become more experienced, you will want to write shell programs, and at that point, the differences between shells become significant.



The figure below shows the general form of a UNIX/Linux command. The system prompt (often, a dollar sign for the Bourne shell and Bash shell or a percent sign for the C shell) is displayed by UNIX/Linux. Command names are generally terse (*ed* for editor, *man* for manual on-line help, *cp* for copy a file), but meaningful. One or more spaces separate the command name from the options. If they are included, the options are usually preceded by a minus sign (a hyphen) to distinguish them from the arguments. Most options are designated by a single lower-case letter, and multiple options can be coded. One or more spaces separate the options from the arguments, which generally consist of one or more file names.



General Introduction to ALL UNIX/Linux Labs.

You will work the Linux labs in the VMWare environment of the College of Business, University of Louisville. We will use a free version of Ubuntu Linux - 18.04.3 LTS (GNU/Linux 4.15.0-65-generic x86_64). The 18.04.1 LTS version was released in August 2019. The DNS hostname for the computer with Ubuntu Linux is mercury.cbpa.louisville.edu and its IP address that subject to change is 136.165.108.23.

The information about Ubuntu Linux is available at https://en.wikipedia.org/wiki/Ubuntu_%28operating_system%29 and other web sites. Also, three useful links to the websites covering Ubuntu Linux documentation, management and support are below.

Documentation: <https://help.ubuntu.com>
Management: <https://landscape.canonical.com>
Support: <https://ubuntu.com/advantage>

To access (log in to) Linux off the University campus you must have a VMWare Horizon client downloaded to your own desktop or your own laptop. The client is available at the following link:

<https://business.louisville.edu/current-student-resources/facilities-technology/virtual-desktop/>.

To access (log in to) Linux on the University campus, you will need a PuTTY Secure Shell client. The name of the Linux server is *mercury.cbpa.louisville.edu*. The exact log in procedures are described in a file named *Instructions to Log in to Ubuntu Linux.pdf* posted in the Assignments/Labs folder on Blackboard (BB).

For file transfer between the Linux system and your local computer you will use PSFTP which is available on the same web site as PuTTY. The web site is <http://www.putty.org/> and <http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html>. If you would like to download PuTTY to your own laptop, remember to choose the right version of PuTTY and PSFTP, for 32- or 64-bit Windows OS. The same refers to a VMWare Horizon client as there are two versions of the client for 32- and 64-bit Windows OS.

NOTES/HINTS: (Read them before you start the tutorial)

1. When you login to Linux, you will see the Linux prompt consisting of *youruserid*, @, and *mercury:~\$*: linked together. For example, *jmzura01@mercury:~\$*. Ubuntu Linux users use the **Bourne-Again shell (Bash shell)** when they first login.
2. The screen shots shown in this tutorial will not exactly match yours. I ran this lab from my Linux account that I had had for several years. As a result, the names of files and the number of files in your Linux directories and mine will be different.
3. You may change your initial password from *hello123* to any other password by issuing the *passwd* command from the prompt and then following the dialog. Memorize the new password and remember that Linux password is case sensitive.
4. To get to the C shell, type the command **cs** and press Enter. The default prompt for the C shell on this installation of Linux is %. To return to the Bash shell type **exit** from the prompt and press Enter. To go to the Korn shell type **ks** and press Enter. The default prompt for the Korn shell on this installation of Linux is \$. To return to the Bash shell type **exit** and press Enter. To determine the shell you are currently using, type **echo \$0** and press Enter. The tutorial you will follow can be completed using any of the available shells.
5. To delete the entire command line you are entering from any of the four shells, hit **CTRL-U**.
6. You may use the *nano* editor or *pico* editor instead of the *vi* editor to create the required files **tom**, **dick**, and **harriet**, etc. on page 12 of this tutorial. Be sure to use filenames as suggested in the tutorial.
7. The *nano* and *pico* editors are easier than *vi* editor and are more self-explanatory. For example, to access these editors you type **nano tom** or **pico tom**, respectively, and press Enter, where **tom** is a file name. To get help on *nano* and *pico* editors commands, type **Ctrl-G** (hold down the Control key and press G) within the editor.
8. A Web link to the *vi* editor commands would be useful. For example, <https://www.cs.colostate.edu/helpdocs/vi.html>.
9. The *vi* editor is awkward, so be patient since it will take you some time to get used to it. The *vi* editor has two modes of operation: **Command Mode** and **Input Mode**. While *vi* is in

Command Mode, you can give *vi* commands. For example, in Command Mode you can delete text or exit from *vi*. You can use the **Enter** key, the **Spacebar**, and the **arrow keys** to move the cursor. If your terminal does not have arrow keys, you can use **h**, **j**, **k**, and **l** keys to move the cursor left, down, up, and right, respectively. To enter insert mode of the *vi* editor, type **i**. You can type anything you want - a short couple of lines is fine. You will need to exit insert mode with **Esc**. You may hear a beep (you are in the Command Mode). Then try typing the **:**. The cursor should go to the bottom to wait for a command. **:wq** (for write and quit) will save the file and get you out. To quit without saving your work, type **:q!**.

10. Continue the same practice to make files **dick** and **harriet** on page 12. Unix recognizes upper and lower case letters as being different, so be consistent. After you watch **cat tom dick harriet**, try **more tom dick harriet** on page 13. This will be interesting to watch.
11. After you create a link from one directory to the other on page 14, be sure you change directories and use the **cat** to view the contents of the file. Try **ls -l** to see the protection level in both directories. Are they the same?
12. You may use a **PSFTP** client at the link:
(<http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html>) to download/upload a file between the Linux platform and your local computer.
 - a. Start the PSFTP client
 - b. Type the command: *open mercury.cbpa.louisville.edu*.
 - c. From the *login as:* prompt, type *your ULink user id* and *Enter*.
 - d. Key in your *password*. (Note that it may be the new ULink password, for example.
 - e. You should see the *psftp>* prompt.
 - f. From the *psftp>* prompt, type *get filename*. For example, *get eval* and *Enter*, where *eval* is the file name. This should transfer file *eval* from the Linux server directory to a default directory on your local machine. If the Desktop is your local directory, you should see file *eval* on your desktop.
 - g. If you receive any errors, type *help* from the *psftp>* prompt and press *Enter*. You will see a list of useful commands that will allow you to access your Linux machine (called the server) and your local machine. For example, command *lcd* will allow you to change the local working directory on your local computer, and command *lpwd* will allow you to display the local working directory on your local computer. If you get errors during the file transfer, try to resolve the problem by yourself. If you still keep getting errors, please contact the instructor.
 - h. Type *exit* and press *Enter* to close PSFTP program.

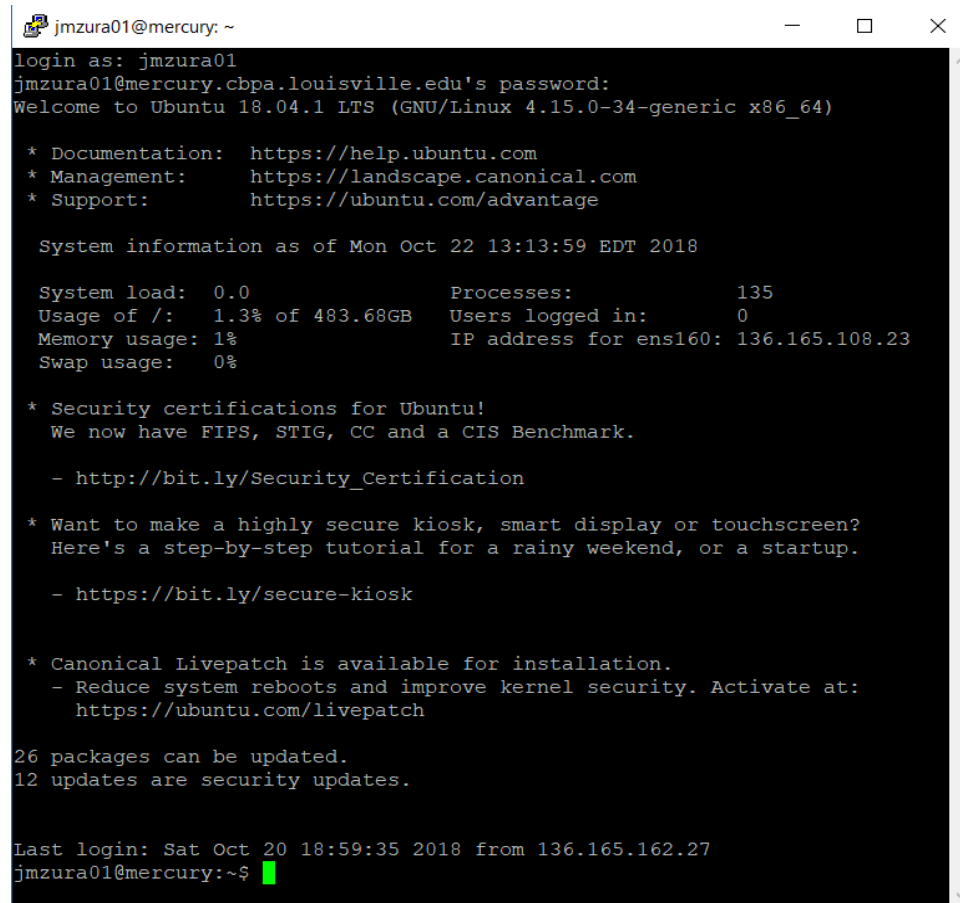
THE LAB

This introductory lab to UNIX/Linux simple commands and utilities is presented as a tutorial. The examples were run on a Dell computer under Ubuntu Linux and the Bourne Again (Bash) shell. Do not just read it. Instead, log in to the Ubuntu Linux system and, as you read about a command, enter it and see for yourself how the computer responds.

Usually, a system administrator or super user is responsible for such start-up procedures as booting the system and setting the date and time, so the Linux user can ignore these tasks. When you sit down at a terminal, the system should be up and running. Every Linux session begins with a request for a **login** name (**your userid**) which is the same as your ULink user id and a **password** (your initial password is **hello123**). In response to the first prompt, type your login name and press return. Next, you'll be asked for your password. Type it and press return. For security reasons, passwords are never displayed. As mentioned, the precise steps to access Ubuntu Linux are described in the file named *Instructions to Log in to Ubuntu Linux.pdf* posted in the Assignments/Labs folder on BB.

On some systems, you will be expected to select your own password the first time you log on. Use any combination of up to eight keyboard characters. UNIX prefers relatively lengthy passwords and may ask you to try again if you suggest less than six characters. You may change your initial password (**hello123**) to another password using the *passwd* command/utility with no options. If you decide to do that, you will have to follow a simple dialog concerning the password's change displayed on the screen. Memorize the new

password! Next time you log in use the new password. The following figure shows a normal log on sequence to Ubuntu Linux.



```
jmzura01@mercury: ~  
login as: jmzura01  
jmzura01@mercury.cbpa.louisville.edu's password:  
Welcome to Ubuntu 18.04.1 LTS (GNU/Linux 4.15.0-34-generic x86_64)  
  
* Documentation:  https://help.ubuntu.com  
* Management:    https://landscape.canonical.com  
* Support:       https://ubuntu.com/advantage  
  
System information as of Mon Oct 22 13:13:59 EDT 2018  
  
System load:  0.0          Processes:            135  
Usage of /:   1.3% of 483.68GB Users logged in:      0  
Memory usage: 1%          IP address for ens160: 136.165.108.23  
Swap usage:   0%  
  
* Security certifications for Ubuntu!  
We now have FIPS, STIG, CC and a CIS Benchmark.  
  
- http://bit.ly/Security_Certification  
  
* Want to make a highly secure kiosk, smart display or touchscreen?  
Here's a step-by-step tutorial for a rainy weekend, or a startup.  
  
- https://bit.ly/secure-kiosk  
  
* Canonical Livepatch is available for installation.  
- Reduce system reboots and improve kernel security. Activate at:  
  https://ubuntu.com/livepatch  
  
26 packages can be updated.  
12 updates are security updates.  
  
Last login: Sat Oct 20 18:59:35 2018 from 136.165.162.27  
jmzura01@mercury:~$
```

Logging on

After each command press the <Enter> or <Return> key. The majority of Linux commands are written in lower case.

The *date* utility displays the system date and time. Type

date

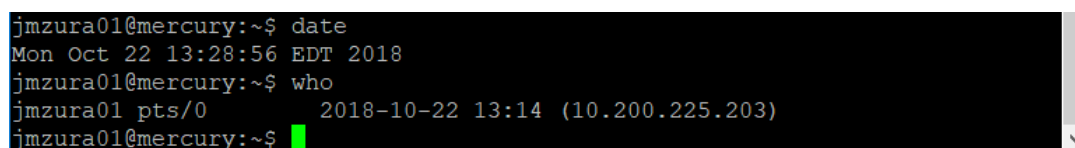
To identify users currently logged on your system, type

who

A user working on more than one project may have two or more login names, and that can be confusing. The command

who am i

displays the user's current login name.



```
jmzura01@mercury:~$ date  
Mon Oct 22 13:28:56 EDT 2018  
jmzura01@mercury:~$ who  
jmzura01 pts/0      2018-10-22 13:14 (10.200.225.203)  
jmzura01@mercury:~$
```

The *write* utility (on some systems, it is *talk*) allows two users to exchange real-time messages, while *mail* sends and receives electronic mail. Many larger UNIX/Linux systems feature an on-line reference manual. To obtain a description of any utility, code *man* followed by the utility name. For example,

man who

displays a description of the *who* utility.

Try out by coding

who am i

and then

man who

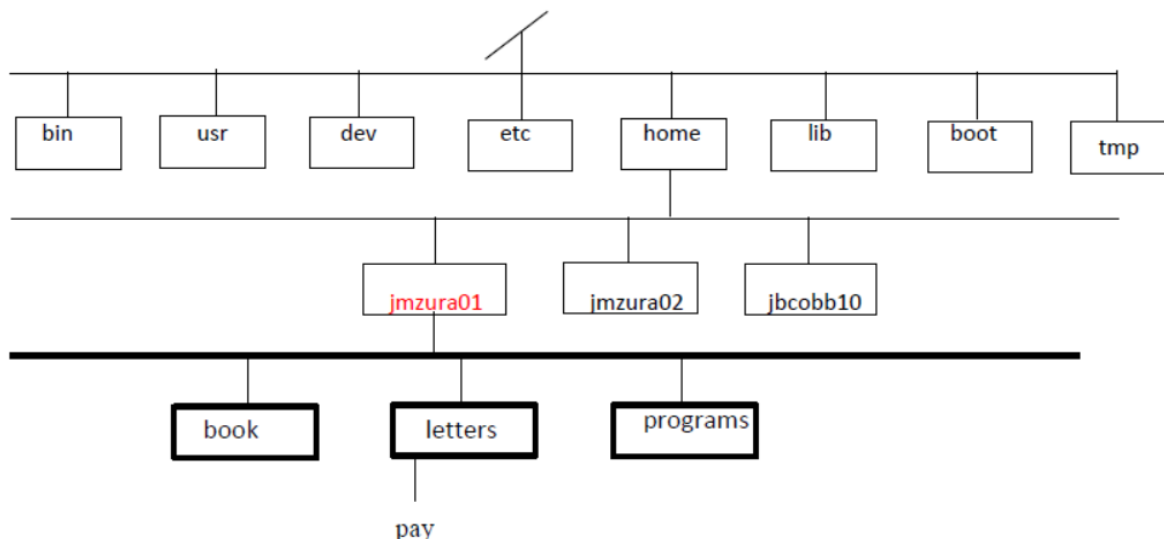
Four shells, the Bourne Again shell (bash), the Bourne shell, the C shell, and the Korn shell are considered UNIX/Linux standards. On most systems, including Ubuntu Linux, the bash is started after login. While the examples in this lab and two other labs on Linux will work under either shell, you can select one. To activate the Bourne shell, type *sh*. To return to the bash shell type *exit*. To switch to the C shell, type *csh*. To return to the bash shell type *exit*. To go to the Korn shell, type *ksh*. To return to the bash shell type *exit*. The Bourne Again shell (bash), which is based on the Bourne shell, is the standard on UNIX/Linux systems and is used in the examples in this lab and two other labs.

The File System

The UNIX file system allows a user to identify, save, and retrieve files by name. (A program is a type of file.)

File Names

A UNIX/Linux **file name** consists of from 1 to 255 characters. (Earlier versions of UNIX had a 14-character limit.) Do not use slashes (/) and avoid starting a file name with a minus sign or hyphen; otherwise, virtually any combination of characters you can type is legal. Note that UNIX/Linux distinguishes between upper case and lower case, so "A" and "a" are different. If you include a period, the characters following it are considered the file name extension. The extension is significant to some compilers and to the linkage editor; otherwise, it is simply part of the file name. An invisible file's name starts with a period. Invisible file names are not normally displayed when a directory is listed.



Directory

Like Windows, UNIX/Linux uses upside-down hierarchical tree directory structure. The structure begins with a **root directory** (see the "/" in the figure above). "Growing" from the root are several "children". Some hold references to utilities and other system routines. One, *home*, holds all the user directory names. (Recall the class discussion and lecture notes.) Note that *jmzura01* (it will be your user id) is *home*'s child, a grandchild of the *root* directory. Under *jmzura01* come subdirectories to hold *book*, *letters*, *programs*, and *chapters* (not shown). Incidentally, a directory is a special type of file, so the rules for naming directories and files are the same. The *usr* directory is for files that are shareable across a whole site. For the meaning of other directories (*bin*, *dev*, *lib*, etc.) please refer to the lecture notes posted in Course Documents folder on BB.

Path Names

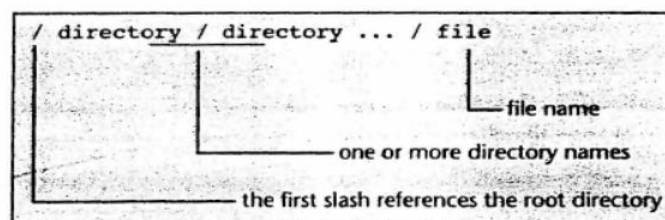
With all these directories, however, you need more than a simple name to find a file. For example, it is possible to have files named *pay* recorded under two different directories. A reference to *pay* would thus be ambiguous - which *pay* do you mean? To uniquely identify a file, you need a complete/absolute **path name**, for example,

/home/jmzura01/letters/pay

Look at the figure on page 6 and follow the path name. The first slash (/) indicates the root directory. Move down to *home*, then *jmzura01*, then *letters* directories, and finally to file *pay*. At first glance, subdirectories seem to complicate rather than simplify accessing files. In practice, however, people rarely use such lengthy pathnames. Instead, when you log on, UNIX/Linux selects your **home directory** (its name usually matches your login name, or example *jmzura01*) as your initial **working directory**. Unless it is told otherwise, the operating system searches for files starting with your working directory; thus,

letters/pay

is all you need to find file *pay*. Above you can see a relative path as you are already in *jmzura01* subdirectory. The operating system, however, builds internally an absolute path starting at the root to access file *pay*. Later in the chapter, you will see how to change working directories. A general form of the path concept is shown below.



Viewing a Directory

Before you begin creating and manipulating directories, look through some existing ones. Start by displaying your working directory. Just type

pwd

and then press Return. The results are shown below (your working directory name will be different).

```
jmzura01@mercury: ~  
jmzura01@mercury:~$ pwd  
/home/jmzura01  
jmzura01@mercury:~$
```

You may write down your working directory, so you will not forget it, but it may be unnecessary because the working directory may be displayed on the screen anyway.

Even if this is the first time you have logged on, your home directory should contain a few files. To view their names, type an *ls -a* (list directory) command:

ls -a

The output is shown below (your output will obviously differ).

```

jhzura01@mercury:~$ ls -a
.          eval50      prog1       savenow1
..         .eval.swp   prog100.c   savewho
a.out      .gnupg             prog10.c    script1
arguments  joe             prog1.c     script2
.bash_history joe1           prog2.c     script3
.bash_logout joe111         prog3.c     script4
.bashrc     joe121        prog4       .sh_history
book        joe5           prog4.c     simple1
.cache      lab4           program1.c  simple2
checkuser   Lab4          programs    .sudo_as_admin_successful
designmenu   letters       random.dat  test1
designmenu10 log           randsor.dat typescript
eval         names        sample10    users_on
eval100      namessorted   sample11    .vim
eval101      .nano         savels      .viminfo
eval5        .profile      savenow
jhzura01@mercury:~$

```

The command lists all files including invisible files, such as `.profile` and `.bashrc`, in a short form. The file names are displayed only. The file names that begin with a period are usually invisible; had the `-a` option not been coded, they would not have been listed.

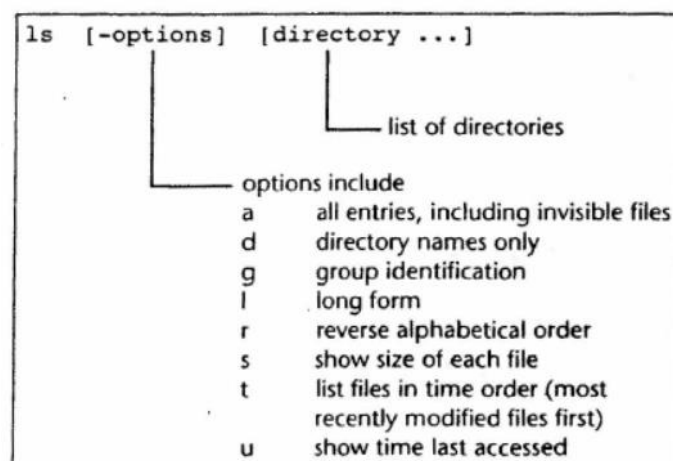
Two files, `(.)` and `(..)`, visible in the left top corner are particularly interesting. The single period stands for the working directory; the double period is a synonym for its parent. They are useful shorthand for writing path names.

For example, code

`ls`

with no options. File names beginning with a period should no longer appear. In fact, on many systems, absolutely nothing is displayed because initially there are no regular files. If you ask for a list of files, and there are none, UNIX/Linux displays nothing, not even a "directory empty" message. To experienced programmers, that makes sense. A beginner might find such terseness a bit intimidating, however.

A general form of the `ls` command is shown below. The command displays, normally in alphabetical order, the names of the files in the specified directories. If no directories are coded, the contents of the current working directory is listed.



To list only directories, code a `d` option

`ls -d`

When you type `ls -d`, access to some directories may be disabled and you will only see the period `.` or the command will not work. You may try to type `ls -al` or `ls` instead.

Next, take a look at the long form:

`ls -l`

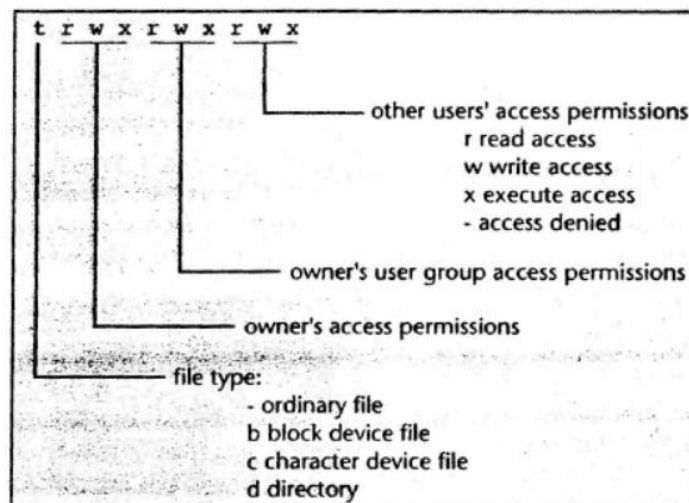
Finally, list everything, including invisible files, in long form:

`ls -la`

```
jnzura01@mercury:~$ ls -la
total 41392
drwxr-xr-x 11 jnzura01 jnzura01 4096 Oct 18 13:46 .
drwxr-xr-x 213 root root 4096 Aug 23 16:55 ..
-rwxrwxr-x 1 jnzura01 jnzura01 8488 Oct 17 21:50 a.out
-rwxrw-r-- 1 jnzura01 jnzura01 58 Oct 26 2017 arguments
-rw----- 1 jnzura01 jnzura01 13626 Oct 18 14:16 .bash_history
-rw-r--r-- 1 jnzura01 jnzura01 220 Aug 8 2017 .bash_logout
-rw-r--r-- 1 jnzura01 jnzura01 3771 Aug 8 2017 .bashrc
drwxrwxr-x 2 jnzura01 jnzura01 4096 Mar 27 2018 book
drwx----- 2 jnzura01 jnzura01 4096 Aug 9 2017 .cache
-rwxr-xr-x 1 jnzura01 jnzura01 156 Oct 26 2017 checkuser
-rwxrw-r-- 1 jnzura01 jnzura01 725 Mar 30 2018 designmenu
-rwxrw-r-- 1 jnzura01 jnzura01 725 Mar 21 2018 designmenu10
-rw-rw-r-- 1 jnzura01 jnzura01 61 Mar 25 2018 eval
-rw-rw-r-- 1 jnzura01 jnzura01 17 Mar 26 2018 eval100
-rw-rw-r-- 1 jnzura01 jnzura01 22 Mar 26 2018 eval101
-rw-rw-r-- 1 jnzura01 jnzura01 18 Mar 28 2018 eval5
-rw-rw-r-- 1 jnzura01 jnzura01 26 Mar 25 2018 eval50
-rw-rw-r-- 1 jnzura01 jnzura01 1024 Oct 26 2017 .eval.swp
drwx----- 3 jnzura01 jnzura01 4096 Aug 22 13:49 .gnupg
-rw-rw-r-- 1 jnzura01 jnzura01 0 Aug 19 2017 joe
drwxrwxr-x 2 jnzura01 jnzura01 4096 Oct 17 22:38 joe1
-rw-rw-r-- 1 jnzura01 jnzura01 31 Oct 17 22:40 joe111
-rw-rw-r-- 1 jnzura01 jnzura01 250 Mar 5 2018 joe121
```

The screen above represents about 1/3 only of the directory listing in my Linux account. To indicate more than one option, simply code all the option letters one after another.

A "long form" line shows a file's owner, file size, and the date and time it was most recently modified. The first 10 characters indicate the file type and its access permissions. The file can be an ordinary file (data or a program), a directory, or a special file that corresponds to an input or output device. Three sets of permissions are included—one for the file's owner, a second for users in the owner's group, and a third for all other users. Based on the recorded values, a given user or group can be granted read (r), write (w), and/or execute (x) permission. A minus sign indicates no permission.



To change access permissions, use the `chmod` utility. For general syntax of the `chmod` command, refer to the lecture notes or a file with UNIX/Linux command syntax posted on BB.

Creating Directories

Before you create the directories on this page, type

`pwd`

to be sure you are at your own directory. If you are not, type

`cd`

You should then be back in your home directory. (The `cd` command will bring you to your own directory from anywhere.)

Next, create three subdirectories under your home directory with the `mkdir` (make directory) command. Type

`mkdir letters book programs`

Note that the command is followed by a list of directory names separated by spaces. The command can be followed by a name of a single directory as well. UNIX/Linux will respond by creating the requested directories but will not display a confirmation message. (You did not tell it to.)

To find out if the directories actually were created, type

`ls`

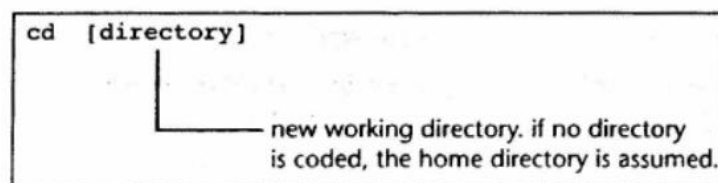
The output is shown below.

```
janzura01@mercury:~$ ls
a.out      eval5    letters  prog3.c   savels    simple2
arguments  eval50   log      prog4      savenow   test1
book       joe      names    prog4.c   savenow1  typescript
checkuser  joe1     namesorted program1.c savewho   users_on
designmenu  joe111   prog1    programs  script1
designmenu10 joe121   prog100.c random.dat script2
eval       joe5     prog10.c randsor.dat script3
eval100    lab4     prog1.c  sample10  script4
eval101    Lab4     prog2.c  sample11  simple1
janzura01@mercury:~$
```

You should see the three directories *book*, *letters*, and *programs*, among many other directories and files that I created over time. The `mkdir` utility creates a directory. Use `rmdir` to remove or delete one.

Changing Working Directories

The `ls` command lists the contents of the current working directory. Use a `cd` command to change the current working directory.



For example, code

`cd letters`

to move to directory *letters*.

Type

`cd ..`

to return to the parent of directory *letters*, your home directory.

Type

`cd /`

The slash identifies the root directory. UNIX/Linux will display no confirmation message, but the root directory will be your new working directory. Now code

`ls -a`

The output is shown below. Your screen may look a little different.

```
jhzura01@mercury:~$ cd /
jhzura01@mercury:/$ ls -a
.      boot  home      lib          media  proc  sbin  sys  var
..     dev   initrd.img lib64         mnt    root  snap tmp  vmlinuz
bin    etc    initrd.img.old lost+found  opt    run   srv  usr  vmlinuz.old
jhzura01@mercury:/$
```

Next, switch to a lower level, or child, directory. Type

`cd /home` or `cd home`

On many UNIX/Linux systems, home contains each user's home directory. Once again, list your working directory's contents; the screen with about two dozen of the user ids is shown below.

```
jhzura01@mercury:/home$ ls
a0ghos06  bgmatt01  dccoom01  j0winc01  laribi10  ndrigh02  rrvolk01
a0tase03  bhhedd01  ddarti02  jcstin04  larobi10  ndstew01  rwgolw01
aaarra01  bjbark03  ddguin01  jdgedd01  lgnico01  njturn03  s0newt01
abeppl01  bjlascl  djbola01  jdhal113  lncurd01  nkazza01  sdmerc01
acfroel  bmfish03  djheat01  djjala01  lnnGuy03  npcoly01  sdmorg05
```

Finally, move back to your home directory. Type

`cd`

with no options or arguments. To verify that you've returned to your home directory, type a *pwd* (print working directory) command. Once again, list the directory's contents.

Creating Files

Most files are created by programs, such as editors, compilers, interpreters, word processors, spreadsheets, and database managers. Most UNIX/Linux systems incorporate both a line editor (*ed*) and a powerful full screen "visual" editor (*vi*). I mentioned in class about two other editors available in Ubuntu Linux. These are *nano* and *pico* editors. While an in-depth discussion of its features is beyond the scope of this book, you can use *vi* to create a few simple files. A general structure of the *vi* command is shown below.

```
vi file
└── name of file to be created or modified
```

First, however, change the current working directory to *letters*. It is a subdirectory of your home directory. (If you have been following this lab, your home directory is your current working directory.) Unless you specify otherwise, UNIX/Linux always assumes a file reference starts with the current working directory, so

`cd letters`

changes the working directory to *letters*.

Start by requesting the visual editor. For example, to create (in the working directory) a file named *tom*, code

`vi tom`

Except for tilde signs (~) displayed in each row in the left column, the screen should go blank. The visual editor has two operating modes: command and insert. As you begin, you are in command mode. (We'll cover only a few essential commands.) To enter insert mode, press the `/` key. An `i` (for insert) key would also work. You will see a message `--INSERT--` at the bottom of the screen and you should be able to begin entering text. (Note: you may have to tell `vi` your terminal type--check with your instructor or your system administrator.)

Type anything you want. When you are done, exit insert mode by pressing the escape (`Esc`) key (or, on some systems, a function key), and then type `:wq` (for *write* and *quit*). You can switch between the modes by hitting the `Esc` key. On many systems, you will hear a beep. You should see a system prompt indicating that you're back in the shell. Type an `ls` command to verify that the file is on disk. Repeat this procedure to create two more files named *dick* and *harriet*. To do it you can use any of the three editors (*nano*, *pico*, or *vi*).

Type

nano dick or *pico dick* or *vi dick*

and then, type

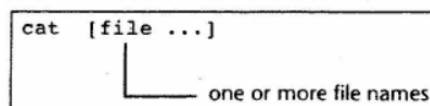
nano harriet or *pico harriet* or *vi harriet*

Again, if the *vi* editor is too cryptic use *nano* or *pico* editor to create the mentioned three files: *tom*, *dick* and *harriet*. Type anything you want in files *dick* and *harriet*.

Some operating systems require a programmer or user to specify a file's size when creating it. That is not necessary under UNIX/Linux. You may search on the Internet how UNIX/Linux dynamically allocates disk space.

Manipulating Files

Now that you have created some files, let us manipulate them. For example, the concatenate (*cat*) utility displays the contents of selected files.



To display *tom*, code

cat tom

The file should appear on your screen. To list the contents of more than one file, code a series of file names; for example,

cat tom dick harriet

You should see whatever you typed through *vi*, *pico* or *nano*.

Imagine that a file named *resume* contains enough data to fill several screens. If you code

cat resume

much of the data will scroll off the screen before you can read it. To display the file's contents one screen at a time, code

more resume

Press the space bar to view another screen, or the delete key to end the program. The more utility is not available on all versions of UNIX/Linux. If your system does not have *more*, you might be able to suspend a display by pressing *Ctrl-s*, and resume the display by pressing *Ctrl-q*. (Hold down the *Ctrl* key and press *s*. Hold down the *Ctrl* key and press *q*.)

Your current working directory is *letters*. Switch back to your home directory by coding

```
cd
```

Now try

```
cat harriet
```

You should get an error message. Try

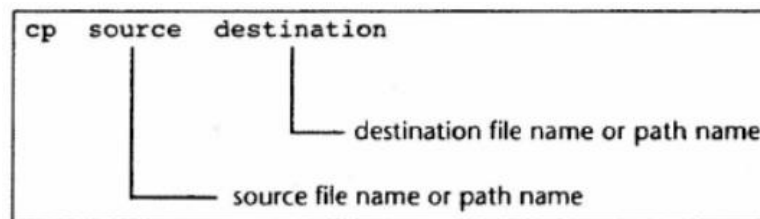
```
cat letters/harriet
```

You should get valid output. Why? List your working directory:

```
ls -a
```

Do you see a file named *harriet*? No. However, the directory *letters* does appear. If you follow a path from your current directory, through *letters*, you will find *harriet*.

To copy a file, use the copy (*cp*) utility.



For example, code

```
cp letters/dick book/chapt.2
```

Now,

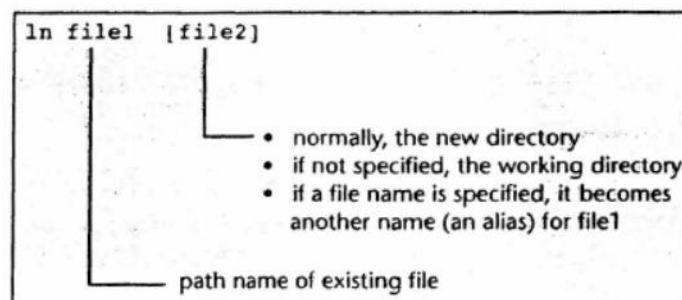
```
cat book/chapt.2
```

should display the contents of the newly created file. List the directory

```
ls -a book
```

File *chapt.2* should appear in the list.

The same file can be referenced in more than one directory by creating a link.



For example, code

```
ln letters/harriet book
```

followed by

ls -a book

You should see a new entry in the list. `Copy` duplicates a file. `Link (ln)` does not; it merely assigns another name to the same file by creating a new directory entry.

To rename a file, code an `mv` (move) command. To delete a file, code an `rm` (for remove link) command. If a file has multiple links, `rm` will not delete it; you'll still be able to access it through other links. When you remove the last link to a file, however, it is gone.

Wild-card characters make it easy to reference a number of related file names. A question mark (?) represents any single character; for example, the file name

term?

expands to *term1*, *term2*, *termc*, and any other file named *term* followed by any single character. An asterisk (*) represents multiple characters; for example,

*term**

stands for *term1*, *term.v6*, *term.abcdefgh*, and any other file named *term* followed by any combination of from 1 to 250 characters. (The limit, remember, is 255, and the period counts.)

Imagine you have been working on a C program. (You will do that in the next Linux lab). Your source module is called *mypgm.c*; the object module is *mypgm.o*. You want to copy both. You have two options. One is issuing two `cp` commands. The other is referencing *mypgm.** or *mypgm.?*. Seeing the wild-card characters, the shell will look for all files that fit; thus a single wild-card name references both.

Pipes, Filters, and Redirection

Many UNIX utilities and commands assume a standard input (*stdin*) or output device (*stdout*); for example, `cat` sends its output to the screen, while `vi` gets its input from the keyboard. By using redirection operators (< - input redirection, > - output redirection, >> - append) described in the following table, a user can tell the shell to change those defaults.

Operator	Meaning	Example
<	change source to a specified file or device	<myfile
>	change destination to a specified file or device	>tempfile
>>	change destination, usually to an existing file, and append new output to it	>>master.pay
	pipe standard output to another command or to a filter	cat file1 sort

Use `cat` to illustrate this feature. You already know that a `cat` command followed by a file name displays the contents of the file. Try coding `cat` with no options,

cat

Since no inputs or outputs are specified, the shell assumes the standard input and output devices (the keyboard and the screen). Thus, whatever you type is simply echoed back to the screen. Try typing a few lines. Press return at the end of each line; when you are finished, press `Ctrl-D` (the end-of-file sentinel value). Some UNIX/Linux systems echo line by

line; others display all the output only after you press *Ctrl-D*; in either case, data are copied from the standard input to the standard output device.

Redirect that output. Type

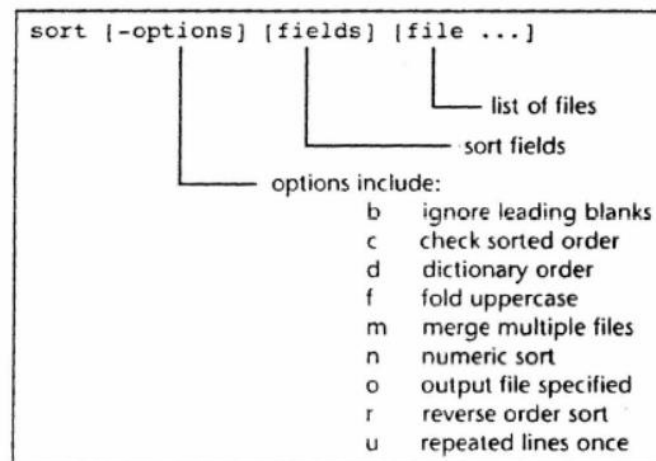
```
cat > book/intro
```

followed by several lines of text. Press *Ctrl-D* when you're finished. Now, list your directory

```
ls -a book
```

You should see the new entry, *intro*.

A filter accepts input from the standard input device, modifies (or filters) the data in some way, and sends the results to the standard output device. For example, consider *sort* (see below.)



It is a utility that reads input from the specified file or files (or the standard input device), sorts them into alphabetical or numerical sequence, and outputs the sorted data to the screen. It can also be used as a filter.

A **pipe** causes one utility's standard output to be used as another utility's standard input. Pipes are designated by a vertical line (`|`). (See the table at the top of the page). For example, earlier in the chapter, you coded

```
cat tom dick harriet
```

Assuming that you are in subdirectory *letters*, repeat that command. Now, try

```
cat tom dick harriet | sort
```

If you are in your home directory type *cd letters* to change the directory to *letters*. The screen shot below shows the output from both *cat* commands. In the second command, the standard output has been routed through *sort*, and the file contents are displayed, line by line in alphabetical order. Your output will obviously vary and you will see whatever you

```
jhzura01@mercury:~/letters$ cat tom dick harriet
I love Linux operating system.
I am looking forward to Thanksgiving and Christmas!
See you skiing on the Copper Mountain in Colorado.
jhzura01@mercury:~/letters$ cat tom dick harriet | sort
I am looking forward to Thanksgiving and Christmas!
I love Linux operating system.
See you skiing on the Copper Mountain in Colorado.
jhzura01@mercury:~/letters$
```

typed earlier through *vi*, *nano*, or *pico* editor and saved in files *tom*, *dick*, and *harriet*. UNIX/Linux utilities can be viewed as tools. Each one performs a single function. Instead of writing a massive program to perform a series of functions, it makes sense to use the

existing tools. Pipes, filters, and redirection make it easy to link programs and utilities. As you become more experienced with UNIX/Linux, you will find many uses for them.

Incidentally, most utilities send error messages to the standard error device - usually, the screen. The shell does not redirect error messages. Thus, they will not be lost in an output file or down a pipe.

This marks the end of the chapter tutorial. When you are ready, log off the system by pressing *Ctrl-D*. An option is typing a *logout* command.

Shell Scripts

Many data-processing applications are run daily, weekly, or at other regular intervals. Others for example a program test are repeated many times. When such applications are run, a set of commands must be issued. Repeating the application means repeating the commands. Retyping the same commands over and over again is annoying and error prone. An option is writing a **shell script**.

A **shell script** is a file that consists of a series of commands. (It is much like the Windows .BAT files shown in the lab on Windows.) The shell script is actually a highly sophisticated, interpretive programming language with its own variables, expressions, sequence, decision, and repetitive structures. From within a shell script you can call other shell scripts as well. You can also pass parameters to a shell script. Writing shell scripts is beyond the scope of this lab, but it is a powerful UNIX/Linux feature that you will eventually want to learn more about. In the next Linux labs you see several examples of scripts files.

Other Useful Commands

You have barely scratched the surface of the UNIX/Linux command language. However, assuming you have actually tried the commands described in this this, you should be able to read a reference manual or a UNIX/Linux textbook and determine how to use additional commands on your own. For example, command *lpr*. It sends the contents of a file to the line printer. Then command *pr*. This filter prepares output for the line printer, adding page headers, page numbers, and so on. Try using pipes to direct the output from *cat*, through *pr*, and then to *lpr*. The print commands *pr*, *lpr*, and *lp* are disabled on this installation of Ubuntu Linux. As a result, none of them will actually work.

You have already encountered the visual editor, *vi*. Several other utilities transform text prepared under *vi* into printable form. For example, *nroff* formats text, *troff* prepares output for a phototypesetter, *eqn* sets up equations, *tbl* formats tables, and *spell* checks for spelling errors. You may learn to use them; they are powerful tools.

Some tasks, for example, printing the contents of a file or performing a lengthy compile, can be quite time-consuming. Instead of idly waiting for such a process to finish executing, you can take advantage of UNIX's/Linux's multiprocessing capability and run it in the background. While it runs, the terminal is free to support another program.

To run a program/command in the background, type an ampersand (&) at the end of the command line; for example,

```
cat tom dick harriet | sort &
```

Usually, the shell starts a command as soon as you enter it, and then waits for it to terminate before displaying the next prompt. The ampersand tells the shell to start the process, immediately display a process identification number, and give you another prompt. Respond by typing your next command. To check the status of your background command, type a process status (*ps*) command. Again, the *lpr* and *pr* (print) commands will not work in Ubuntu Linux as the system is not connected to any printer in the lab.

Two other utilities support communications between systems. *Telnet* is a terminal emulator that allows a user to dial a remote computer and access it. UNIX/Linux-to-UNIX/Linux copy (*uucp*) transfers files between UNIX/Linux systems.

Graphic User Interface

If you do not want to use the shell, many UNIX/Linux systems include a standard graphical interface called X-Windows. Linux systems include KDE or GNOME as interfaces that are built on top of X-Windows. These provide an interface very similar to the Windows interface. You can do most operations such as changing passwords, creating or changing directories, and copying files using drag and drop via this graphical interface.

Summary

UNIX/Linux commands are processed by a shell. The basic structure of a command was illustrated, and normal log on procedures introduced. The *passwd* utility can be used to change a password. The *date* utility displays the system date and time, while *who* displays a list of users logged on the system.

The UNIX/Linux file system allows a user to store, retrieve, and manipulate files by name. The rules for defining file names were explained. Because UNIX/Linux uses a hierarchical directory structure, a path name must be specified to completely identify a file. The *pwd* command displays the current working directory's path name. You used *ls* to list a directory's contents, trying several different options. Next, you used *mkdir* to create three directories and experimented with the change directory (*cd*) command. Finally, you used the visual editor (*vi*) as well as *pico* and *nano* editors to create some files, and manipulated those files with *cat*, *cp*, and *ln* commands.

Many UNIX utilities and commands assume the standard input or output device. Redirection tells the shell to change these defaults. A filter accepts data from the standard input device, modifies them in some way, and sends the results to the standard output device. Pipes allow a user to link utilities and other programs, treating the standard output generated by one as the standard input for another. The *sort* utility was used to illustrate pipes and filters.

Many data-processing jobs are run frequently. Thus, the same set of commands must be entered again and again. An option is writing a shell script. This Lab ended with a brief overview of several other UNIX/Linux commands and a discussion of background processing. You will be introduced to more UNIX/Linux commands in the next Linux Labs.

Because the print commands are disabled on this installation of Ubuntu Linux, you may use the PSFTP client to transfer files to your own laptop/desktop and print it from there. If you have not already done so, download the PSFTP client to your personal desktop/laptop. If you work the lab from one of the computers in the labs, you may not be able to download the PSFTP client. However, you can run it.

Optional

Run *psftp.exe* to log in to Ubuntu Linux. You will find the *psftp.exe* at <http://www.putty.org/> and <http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html> and when you see the ftp prompt,

type the command: *open mercury.cbpa.louisville.edu*.

From the *login as:* prompt, type *your user id* and *Enter*.

Key in your *password*. (Note that it may be your new password, for example, if you changed it earlier.) You should see the *psftp>* prompt.

From the *psftp>* prompt, type *get filename* and *Enter*, where *filename* is the generic name for any file that you would like to download. This should transfer file *filename* from the Linux

server directory to a default directory on your local machine, if the Desktop is your local directory, you should see file *filename* on your desktop.

If you receive any errors, type *help* from the *psftp>* prompt and press Enter.

You will see a list of useful commands that will allow you to access your Linux machine (called the server) and your local machine. For example, command *lcd* will allow you to change the local working directory on your local computer, and command *lpwd* will allow you to display the local working directory on your local computer.

If you get errors during the file transfer, try to resolve the problem by yourself. If you still keep getting errors, please contact the instructor.

Type *exit* and press Enter to close PSFTP program.

Submit Lab 3 Report. (See the Assignments/Labs folder on Blackboard)