Warnings

First of all, these notes will cover only a small subset of the available commands and utilities, and will cover most of those in a shallow fashion.

Read the relevant material in Sobell!

If you want to follow along with the examples that follow, and you do, open a Linux terminal.

Second, most of the Linux commands have features that are enabled by using command-line switches; check the man pages for the commands for details!
Getting Started

The Linux terminal (or command shell) allows you to enter commands and execute programs.

A terminal displays a prompt and a cursor indicating it’s waiting for user input:

The prompt itself is configurable; precisely how depends on the particular type of shell you are running.

It is likely that by default you will run the *bash* shell.
What’s Running?

The **ps** command displays information about processes the shell is currently running:

![Screenshot of the ps command output]

We see that two processes are executing, **bash** and **ps**.

Moreover, we see that:
- each is assigned a unique numeric identifier called a process ID or PID
- each is associated with a terminal (TTY) named pts/0

Try executing **ps** a second time… you’ll notice that the PID for **bash** is the same as before but the PID for **ps** has changed.

Why? (That’s two questions.)
Try running **ps** with the **–l** (that’s ell, not one) switch:

```
#1009 wdm@Centos65:--> ps -l
F S UID PID PPID  C PRI NI ADDR SZ WCHAN  TTY     TIME CMD
0 S 500 12092 2717 0  80 0 - 27118 wait pts/2  00:00:00 bash
0 R 500 20331 12092 0  80 0 - 27033  -  pts/2  00:00:00 ps
#1010 wdm@Centos65:--> 
```

Don’t worry about the meaning of all that just yet, but do notice that the results of the **ps** command were altered by the use of a “switch” on the command line.

This is typical of Linux commands and many user programs.
One way to find out more…

The **man** (manual) command can be used to obtain more information about Linux commands:

The **man** pages are often the first resort for discovering options.

Try running **man man**…
The file system is a set of data structures that organizes collections of files.

Files are grouped into *directories* (although directories are themselves files).

Here’s one possible file system organization:
File Names

Each file and directory has a name:
- names are case-sensitive
- names within the same directory must be unique
- the use of characters other than letters, digits, underscores and periods tends to cause extra work when using many commands

File names are often of the form <name.ext>, such as BashTerminal.jpg.

While file extensions are not mandatory, it is standard (and good) practice to employ them.

You are required to use appropriate file extensions in this course.

It is bad practice to employ extensions incorrectly. Common ones include:
- c C language source files
- h C language header files
- txt plain text files
- gz file compressed with gzip
- tar archive file created with tar
- html hypertext markup language file
Each file (and directory) can be specified by a unique *absolute pathname*:

```
/home/hls/bin/log
```

This diagram illustrates the directory structure:

- `/` (root directory)
- `home`
- `tmp`
- `etc`
  - `/etc`
  - `max`
  - `sam`
  - `hls`
    - `bin`
      - `report`
    - `notes`
      - `log`

Home Directory

When you open a terminal, by default you will be in your *home directory*.

Typically, this will be `/home/<userid>`, but you can check the path to your current directory by using the `pwd` command:

```
#1013 wdm@Centos65:~> pwd
/home/wdm
#1014 wdm@Centos65:~>  
```
The `ls` command lists the files in the current directory:

```
#1017 wdm@Centos65:--> ls
2505 Curator extract.c Nov06 old.bashrc
2505Fall2014 Desktop extract.h officialscores.txt reader.c
2506 Documents HW02Soln.tar old2505 scores.txt
temp
2506Fall2014 Downloads marker old2506
bin examples moss old.bash_profile valgrind
#1018 wdm@Centos65:--> ls -l
```

```
total 108
```

```
drwxrwxr-x.  3 wdm wdm  4096 Jan  3 21:16 2505
drwxrwxr-x. 23 wdm wdm  4096 Dec 14 20:30 2505Fall2014
drwxrwxr-x.  3 wdm wdm  4096 Jan  3 19:20 2506
drwxrwxr-x. 15 wdm wdm  4096 Dec  1 10:50 2506Fall2014
drwxrwxr-x.  2 wdm wdm  4096 Nov 23 21:26 bin
drwxrwxr-x.  4 wdm wdm  4096 Sep  5 11:12 Curator
drwxr-xr-x.  3 wdm wdm  4096 Dec  6 18:13 Desktop
drwxr-xr-x.  2 wdm wdm  4096 May  7 2014 Documents
drwxr-xr-x.  3 wdm wdm  4096 Nov 27 14:12 Downloads
drwxrwxr-x.  4 wdm wdm  4096 Sep 30 12:15 examples
-rw-rw-r--.  1 wdm wdm  2843 Nov 19 10:13 extract.c
-rw-rw-r--.  1 wdm wdm  115 Nov 19 10:13 extract.h
-rw-rw-r--.  1 wdm wdm  10240 Sep 22 22:05 HW02Soln.tar
drwxrwxr-x.  5 wdm wdm  4096 Nov  9 20:28 marker
drwxrwxr-x.  4 wdm wdm  4096 Nov 14 22:42 moss
drwxrwxr-x.  2 wdm wdm  4096 Nov  6 13:40 Nov06
-rw-rw-r--.  1 wdm wdm  2104 Dec 11 22:22 officialscores.txt
drwxrwxr-x.  1 wdm wdm  4096 Apr 19 2014 old2505
```
Directory Tree

You can display a map of the directory tree rooted at your current directory:

```
#1001 wdm@Centos65:~ > tree

    2505
    ├── CProgs
    │   ├── Sqz0dds
    │   │   ├── DataIn.txt
    │   │   └── DataOut.txt
    │   └── p
    │       └── SqueezeInt.c
    └── 2505Fall2014
         ├── BinInt
         │   ├── code
         │   │   ├── BinaryInt.c
         │   │   └── BinaryInt.h
         │   └── driver.c
         └── datalab
             ├── bits.c
             └── bits.h
                 └── datalab-handout
                     ├── bits.c
                     └── bits.h
                         └── btest.c
                             └── btest.h
                                 └── decl.c
```
Directory Navigation

You can use the `cd` command to change your current (or *working*) directory:

Using `cd` with no destination moves you back to your home directory:
Relative Pathnames

You can also specify a pathname that’s relative to the current (working) directory.

Let’s say you’re in a directory at the top of the tree shown below:

```
#1001 wdm@Centos65:~$ tree

  2505
  ├── CProgs
  │   └── SqzOdds
  │       ├── DataIn.txt
  │       └── SqzOdds
  ├── 2505Fall2014
  │   ├── BinInt
  │   │   └── code
  │   │       ├── BinaryInt.c
  │   │       └── BinInt.h
  │   └── datalab
  │       ├── bits.c
  │       └── bits.h
  └── datalab-handout
      ├── bits.c
      └── bits.h

~/2505/CProgs/SqzOdds
~/2505/2505Fall2014/BinInt/code
```
Relative Pathnames

There are two special directory names:

. refers to the current directory

.. refers to the parent of the current directory

Suppose your working directory is `~/2505/`:

```
.. refers to wdm

../2505Fall2014/BinInt

../2505Fall2014/BinInt/code
```
Making/Removing a Directory

You can create a new directory with the `mkdir` command:

```
#1010 wdm@Centos65:~$ ls
CProgs
#1011 wdm@Centos65:~$ mkdir scripts
#1012 wdm@Centos65:~$ ls
CProgs  scripts
```

You can remove an empty directory with the `rmdir` command.

```
#1013 wdm@Centos65:~$ mkdir scripts/util
#1014 wdm@Centos65:~$ ls
CProgs
    SqzOdds
    scripts
        util
4 directories
#1015 wdm@Centos65:~$ rmdir scripts
```
Copying Files: cp

You can create a copy of a file with the `cp` command.

Assume we’re in a directory containing a file named `infloop.c`:

```
$ cp infloop.c infloop2.c
```

makes a copy of `infloop.c` named `infloop2.c` in the same directory

```
$ cp infloop.c ..
```

makes a copy of `infloop.c` with the same name in the parent directory

```
$ cp infloop.c ../infloop2.c
```

makes a copy of `infloop.c`, named `infloop2.c`, in the parent directory
Renaming/Moving Files: mv

As before, assume we’re in a directory containing a file named `infloop.c`:

```
mv infloop.c infiniteloop.c
```

changes the name of the file `infloop.c` to `infinitefloop.c`

```
mv infloop.c ../attic
```

moves the file `infloop.c` to the subdirectory of the parent named `attic`

```
mv infloop.c ../infinitefloop.c
```

removes the file `infloop.c` from this directory, and creates a copy named `infinitefloop.c` in the parent directory
Viewing a File: cat and less

You can use the `cat` command to display the contents of a file to the terminal:

```
#1024 wdm@Centos65:~/2505/CPrgs/SqzOdds> cat DataIn.txt
```

You can use the `less` command to display the contents of a file to the terminal, one screenful at a time.
You can view the first few lines of a file by using the **head** command:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// We will expect the longest line in the input file to contain no more
// than 100 characters (which is generous in this case).
const unsigned int MAX_LINE_LENGTH = 101;

// Rather than use a nondescriptive label, or none at all, we will use
// an enumerated type to make the logic of of the code clearer:
```

You can control how many lines are shown; see the **man** page.

There is an analogous **tail** command for viewing the last few lines of a file.
Searching File Contents: grep

The **grep** command can be used to display lines of a file that match a pattern:

```
#1033 wdm@Centos65:SqzOdds> ls
DataIn.txt  DataOut.txt  p  SqueezeInt.c
#1034 wdm@Centos65:SqzOdds> grep -n fprintf SqueezeInt.c
98:        fprintf(Output, "%d: %d\n", N, squeezedN);
102:       fprintf(Output, "%d: %d\n", N, squeezedN);
105:       fprintf(Output, "Unrecognized action flag: %s\n", actionFlag);
#1035 wdm@Centos65:SqzOdds>
```

We will examine the **grep** command in considerably more detail in a future assignment.
You can obtain information about a file with the `file` command:

```
#1036 wdm@Centos65:~/2505/CProgs/SqzOdds> ls
DataIn.txt  DataOut.txt  p  SqueezeInt.c
#1037 wdm@Centos65:~/2505/CProgs/SqzOdds> file DataIn.txt
DataIn.txt: ASCII text
#1038 wdm@Centos65:~/2505/CProgs/SqzOdds> file p
p: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.18, not stripped
#1039 wdm@Centos65:~/2505/CProgs/SqzOdds>  
```
Bundling Files into an Archive: tar

You can create a single file that contains a collection of files, including a directory structure with the `tar` utility:

```bash
#1095 wdm@Centos65:Assembler> ls ./code
AddressExpr.c  IData.c  Parser.c  Pseudos.h  SystemConstants.h
AddressExpr.h  IData.h  Parser.h  Registers.c  Translator.c
assemble.c    Instruction.c Preprocessor.c Registers.h Translator.h
DataSegmentHandler.c Instruction.h Preprocessor.h SymbolTable.c Variables.c
DataSegmentHandler.h makefile Pseudos.c  SymbolTable.h Variables.h
#1096 wdm@Centos65:Assembler> tar cf AssemblerSource.tar code/*.c code/*.h code/makefile
#1097 wdm@Centos65:Assembler> ls -l
```

```
total 256
-rw-rw-r--. 1 wdm wdm 81920 Jan  6 21:37 AssemblerSource.tar
-rw-r-w--. 1 wdm wdm 61440 Oct 19 19:27 C4TestHarness.tar
-rw-rw-r--. 1 wdm wdm 51200 Oct 19 19:27 C4Tests.tar
drwxrwxr-x. 2 wdm wdm 4096 Jan  6 21:35 code
  4 drwxrwxr-x. 4 wdm wdm 4096 Apr 26 2014 extraction
drwxrwxr-x. 3 wdm wdm 4096 Oct 19 19:51 fall2014
  4 drwxrwxr-x. 4 wdm wdm 4096 Oct 19 20:18 milestone
-rw-rw-r--. 1 wdm wdm 30720 Mar  31 2014 MilestoneTest.tar
-rw-rw-r--. 1 wdm wdm 1914 Oct 19 19:25 runAll.sh
-rw-rw-r--. 1 wdm wdm 2497 Oct 19 19:25 runOne.sh
drwxrwxr-x. 3 wdm wdm 4096 Mar 31 2014 temp
drwxrwxr-x. 2 wdm wdm 4096 Mar 31 2014 testfiles
#1098 wdm@Centos65:Assembler> 
```

**cf**  _create archive, write to a file_

Note the name of the new tar file is listed before the target (files to be tar'd up).

**DO NOT** get that backwards!
Checking Contents

You can check the contents of a tar file:

```
#1099  wdm@Centos65:Assembler> tar tf AssemblerSource.tar
code/AddressExpr.c
code/assemble.c
code/DataSegmentHandler.c
code/IData.c
code/Instruction.c
code/Parser.c
code/Preprocessor.c
code/Pseudos.c
code/Registers.c
code/SymbolTable.c
code/Translator.c
code/Variables.c
code/AddressExpr.h
code/DataSegmentHandler.h
code/IData.h
code/Instruction.h
code/Parser.h
code/Preprocessor.h
code/Pseudos.h
code/Registers.h
code/SymbolTable.h
code/SystemConstants.h
code/Translator.h
code/Variables.h
code/makefile
#1100  wdm@Centos65:Assembler>     
```
tf table of contents of a tar file
Extracting a tar File

Use the `x` switch to extract the contents of a tar file:

```bash
#1118 wdm@Centos65:~$ ls
AssemblerSource.tar
#1119 wdm@Centos65:~$ tar xf AssemblerSource.tar
#1120 wdm@Centos65:~$ tree

.  AssemblerSource.tar
   |   code
   |       AddressExpr.c
   |       AddressExpr.h
   |       assemble.c
   |       DataSegmentHandler.c
   |       DataSegmentHandler.h
   |       IData.c
   |       IData.h
   |       Instruction.c
   |       Instruction.h
   |       makefile
   |       Parser.c
   |       Parser.h
   |       Preprocessor.c
   |       Preprocessor.h
   |       Pseudos.c
   |       Pseudos.h
   |       Registers.c
   |       Registers.h
   |       SymbolTable.c
   |       SymbolTable.h
```
Compressing Files: bzip2

The **bzip2** utility can frequently reduce the amount of space a file occupies:

```
#1122 wdm@Centos65:~$ ls -l
 total 84
-rw-rw-r--. 1 wdm wdm  81920 Jan 6 21:42 AssemblerSource.tar
drwxrwxr-x. 2 wdm wdm    4096 Jan 6 21:43 code
#1123 wdm@Centos65:~$ bzip2 AssemblerSource.tar
#1124 wdm@Centos65:~$ ls -l
 total 16
-rw-rw-r--. 1 wdm wdm 10967 Jan 6 21:42 AssemblerSource.tar.bz2
drwxrwxr-x. 2 wdm wdm    4096 Jan 6 21:43 code
```
You can uncompress a file created with bzip2 by using **bunzip2**:
Creating a Compressed tar File

You can also create a compressed tar file:

```
#1135 wdm@Centos65:~/.tmp
AddressExpr.c  IDdata.c  Parser.c  Pseudos.h  SystemConstants.h
AddressExpr.h  IDData.h  Parser.h  Registers.c  Translator.c
assemble.c     Instruction.c Preprocessor.c Registers.h  Translator.h
DataSegmentHandler.c Instruction.h Preprocessor.h SymbolTable.c Variables.c
DataSegmentHandler.h makefile  Pseudos.c  SymbolTable.h Variables.h
#1136 wdm@Centos65:~/.tmp> tar cjf AssemblerSource.tbz code/*.c code/*.h code/makefile
#1137 wdm@Centos65:~/.tmp> ls -l
-rw-rw-r--. 1 wdm wdm 19967 Jan 6 21:57 AssemblerSource.tbz
drwxrwxr-x. 2 wdm wdm  4096 Jan 6 21:43 code
#1138 wdm@Centos65:~/.tmp>
```

Note that the resulting file is about 13% as large as the uncompressed version.

The **j** switch causes **tar** to apply the **bzip2** utility to compress the results.

Alternatively, the **z** switch causes **tar** to apply **gzip**.
Traditional Access Permissions

There are three types of users:
- owner
- group
- other (aka world)

A user may attempt to access an ordinary file in three ways:
- read from
- write to
- execute

Use `ls -l` to view the file permissions:

```
williammcquain@MSI-Ubuntu:~/Documents$ ls -l
total 28
drwxr-xr-x 3 williammcquain williammcquain 4096 2011-08-24 12:52 2104
-rw-r--r-- 1 williammcquain williammcquain 10240 2011-08-24 21:41 2104.tar
-rw-r--r-- 1 williammcquain williammcquain 146 2011-08-24 21:49 2104.tgz
drwxr-xr-x 6 williammcquain williammcquain 4096 2011-08-25 10:21 2505
-rw-r--r-- 1 williammcquain williammcquain 1482 2011-08-22 22:34 GettysburgAddress.txt
williammcquain@MSI-Ubuntu:~/Documents$
```
Traditional Access Permissions

- File type
  - File permissions (owner group other)
  - Number of links
    - Owner
    - Group
    - Size
    - Modification time
    - File name

- rw-r--r-- 1 williammcquain williammcquain 10240 2011-08-24 21:41 2104.tar
Use the **chmod** command to set or alter traditional file permissions:

```bash
williammcquain@MSI-Ubuntu:~/Documents$ chmod g+w 2104.tar
williammcquain@MSI-Ubuntu:~/Documents$ ls -l 2104.tar
-rw-rw-r-- 1 williammcquain williammcquain 10240 2011-08-24 21:41 2104.tar
williammcquain@MSI-Ubuntu:~/Documents$
```

**chmod** also allows the use of numeric arguments:

- 0  no access permissions
- 1  execute permissions
- 2  write to permissions
- 4  read from permissions

So, **chmod 740** would set

- owner permissions to `r w x`
- group permissions to `r- -`
- other permissions to `- - -`

**WHY?**
Binary representations:

<table>
<thead>
<tr>
<th>none</th>
<th>0</th>
<th>000</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>1</td>
<td>001</td>
</tr>
<tr>
<td>w</td>
<td>2</td>
<td>010</td>
</tr>
<tr>
<td>r</td>
<td>4</td>
<td>100</td>
</tr>
</tbody>
</table>

Now notice that $7 = 111$ which is the logical OR of $001$ and $010$ and $100$

And, $740$ thus specifies permissions $7$ for the owner, $4$ for the group and $0$ for others.
The Importance of Access Permissions

When working on a shared environment, like the rlogin cluster, it is vital that you make sure that your access permissions are set correctly.

As a general rule, you will rely on the default access permissions, which are controlled via shell configuration files we will discuss later.

When in doubt, use `ls -l` to check!
Removing a File: \texttt{rm} and \texttt{shred}

If you have sufficient permissions, a file can be deleted from the file system by using the \texttt{rm} command.

Be \texttt{very} careful with \texttt{rm}!

You can also securely remove a file by using the \texttt{shred} command, but see Sobell for a discussion of the limitations.

See the discussion of \texttt{dd} in Sobell for an alternative way to wipe a file.
Many Linux commands support the use of special characters (aka wildcards) to specify a pattern that identifies a set of files:

- `?` matches any single character (in the name of an existing file)
- `*` matches zero or more characters (in the name of an existing file)
- `[]` matches any of the characters within the braces (in the name of an existing file)

- `*.txt` matches any file with extension "txt"

- `foo?.*` matches a file with any extension and name consisting of "foo" followed by a single character

- `[abc]foo.html` matches a file with extension "html" and name "afoo" or "bfoo" or "cfoo"
**scp** can be used to copy a file between the local machine and a remote machine (or between two remote machines).

For example, the following command would copy *GettysburgAddress.txt* from my computer to a directory named **documents** on **rlogin**:

```
scp GettysburgAddress.txt wmcquain@rlogin.cs.vt.edu:documents
```

If you haven’t set up password-less login, you’ll be prompted for the necessary authentication information.

And the following command would copy *GettysburgAddress.txt* from my rlogin account to my current directory on my machine:

```
scp wmcquain@rlogin.cs.vt.edu:documents/GettysburgAddress.txt .
```
Identifying a Command: which

If you’re not sure where a command resides, the `which` command will tell you:

```
williammcquain@MSI-Ubuntu:~/.Documents$ which ls
/bin/ls
williammcquain@MSI-Ubuntu:~/.Documents$ which gcc
/usr/bin/gcc
williammcquain@MSI-Ubuntu:~/.Documents$ gcc --version
gcc (Ubuntu/Linaro 4.5.2-8ubuntu4) 4.5.2
Copyright (C) 2010 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

williammcquain@MSI-Ubuntu:~/.Documents$
```

Many Linux applications also support a `--version` switch which can help identify which specific version of an application you’re invoking.
By default when you execute a command in a shell, the shell program waits (doesn’t provide a prompt and allow entry of another command) until the current command completes (or is otherwise interrupted).

We say the command is running in the foreground.

You can modify this behavior and run a command in the background:
Redirecting stdout

If a process writes output to stdout (the console window), you can *redirect* that into a file:

```
williammcquain@MSI-Ubuntu:~/Documents/2505/examples$ ./sleeper2 5
Still need to sleep for 5 seconds.
Still need to sleep for 4 seconds.
Still need to sleep for 3 seconds.
Still need to sleep for 2 seconds.
Still need to sleep for 1 seconds.
williammcquain@MSI-Ubuntu:~/Documents/2505/examples$ ./sleeper2 5 > sleeper2log.txt
williammcquain@MSI-Ubuntu:~/Documents/2505/examples$ cat sleeper2log.txt
Still need to sleep for 5 seconds.
Still need to sleep for 4 seconds.
Still need to sleep for 3 seconds.
Still need to sleep for 2 seconds.
Still need to sleep for 1 seconds.
williammcquain@MSI-Ubuntu:~/Documents/2505/examples$ 
```
You can use the *pipe operator* to channel the output from one process as input to another process:

```
./sleeper 5 | grep 3 > filtered.txt
```

What do you think the following command would do?
Killing a Process

A (foreground) running process can be killed by using Ctrl-C.

A (background) running process or a suspended process can be killed by using the `kill` command: