Class Exercise 2

- Some of you have not completed Class Exercise 2.
- You **must** do this today or you will lose points.

The Command Line

- In this course we will be using the command line
- On the command line, you talk to the machine by typing
- You have to be careful what you type
- If you make a mistake of even one character, then the command will not work

The Command Line

- This is not the way most of you have interacted with computers
- You are used to a **G**raphical **U**ser **I**nteface (GUI), where a great deal of effort has been spent trying to make things easier for the user
- The command line is **NOT** a user friendly environment
- I sometimes call it a "user hostile" environment

The Command Line

- So why do we study the command line?
- Because...the command line has more ***power***
- GUI's take time to create and maintain, so they will never have all the features of the command line
- Most Linux/Unix server installations never install a GUI
- A GUI uses system resources that are better spent on services

The Command Line

- The commands you type at the command line can be put into text files
- These files are called shell scripts
- Instead of typing the commands, you can simply run the shell script
- This makes it easy to *automate* routine tasks

The Unix *script* Command

- Sometimes it is useful to keep a record of your Unix session
- If you are installing new software, it is good to keep a ***record*** of the options you chose
- Unix provides the *script* command for this purpose
- When you run *script*, Unix creates a new **shell** session.
- Everything you type at the keyboard and everything that gets printed to the terminal is stored in a text file

The Unix *script* Command

- This text file is a record of everything that happened in that shell session
- If you run *script* with no arguments, the session will be recorded in a file named typescript
- This file will be created in the directory you were in when you started *script*
- If you run *script* with an argument, *script* will use that argument as the name of the transcript file

The Unix *script* Command

- To end the *script* session, type *exit* at the command line
- ***script*** does not save anything to disk **until you type *exit***
- If you quit your ssh session before typing *exit*, **NOTHING** will be recorded
  - Be sure you type *exit* before disconnecting from your ssh session
  - If you quit *script* and then later run *script* again in the same directory, you will lose your first session results
  - The file created by the second run of *script*, will overwrite the contents of the first session

The Unix *script* Command

- If you need to add to a previous *script* file, use the -a option
- If you run this:

```
script -a
```

your new session will be added – or appended – to the *end* of your old session file

- When you type *exit* to end a *script* session, you will find yourself back in the directory from which you ran *script*
- This will happen even if you changed directories in the process of your *script* session

The Editor Used in This Class

- Most Unix system administrators use the *vim* text editor
- *vim* is an updated version of the *vi* editor
- The textbook devotes an entire chapter to *vim* -- and another chapter to *emacs*
- *emacs* is preferred by programmers
- You may wish to study the relevant chapters, on your own, for your personal edification and knowledge

The Editor Used in This Class

- I don't want to spend class time teaching you either editor
- The only way to learn an editor is to use it
- Both *vim* and *emacs* have many features, and using them can be very confusing at first
- Today I'll show you ***nano***, a simple text editor that will suffice well for our purposes
- Today's class exercise will give you some practice with *nano*

The Editor Used in This Class

- You are free you use any Unix text editor in this class
- **Do not use a Windows text editor, such as Notepad**
- Such editors create non-printing format characters in the file, which makes it very difficult to read them
- If you use something that is **NOT** a Unix text editor when you are creating a homework file, **I'll deduct 10 percent (of the total) from your grade**

The *nano* Text Editor

- *nano* is a simple text editor created as part of the GNU project
- In *nano* you issue a command by holding down the ***Control*** key while pressing a letter key
- You can move to the *beginning* of a line of text by pressing ***Control-A*** -- and move to the *end* of the line with ***Control-E***
- When I write something like "***Control-A***", I mean hold down the ***Control*** key while pressing the ***A*** key

The *nano* Text Editor

- Although I used a capital A here, I do not mean you should hold down the ***Shift*** key
- I use capital letters when writing control key sequences because the capital letters are easier to read
- Some of the basic *nano* commands appear at the bottom of the page
- The ^ in this list of commands stands for the ***Control*** key, so ^O means ***Control-O***

The *nano* Text Editor

- The *nano* feature set is limited
- You can only work with one file at a time
- You can search for text, but there is no search and replace feature – which may take some getting used to
- However, *nano* does have a limited cut and paste feature
- If you press ***Control-K*** the entire line will disappear, but if you go to another line and press ***Control-U***, the line will be pasted back at that point

The *nano* Text Editor

- When you want to save a file, you press ***Control-O***, and the name of the file will appear at the bottom of the screen
- (You may also be offered a yes-or-no option for saving the file, in which case you should answer ***y*** for Yes or ***n*** for No)
- You need to hit ***Enter*** to accept that name and complete the save process
- ***Control-X*** will quit *nano*

The *nano* Text Editor

- Prof. Hoffman has created a web page with instructions for using *nano* here:

  `http://www.cs.umb.edu/~ghoffman/linux/nano_text_editor.html`

- There is a link to it on the class web page
- (If the link is broken, please let me know..)

Correcting Mistakes on the Command Line

- The command line is **NOT** a user friendly environment
- There are no menus, you have to *remember* the names of all commands
- This is one reason Unix command names tend to be short
- If you make a mistake typing a command, Unix will respond with a cryptic error message

Correcting Mistakes on the Command Line

- Fortunately, Unix provides **command line editing** features that make it relatively easy to correct mistakes
  - *Control-A* **moves to the beginning of the command line**
  - *Control-E* **moves to the end of the command line**
  - *Control-U* **removes all text from your *current* position to the *beginning* of the line**
  - *Control-K* **removes all text from your *current* position to the *end* of the line**
- The right and left arrow keys can also be used to move back and forth over the command line
- Today's class exercise will let you practice these features

Retrieving Your Last Command Line Entry

- When you get an error message from the command line, you need to enter the command again
- This can be very annoying for long and complicated commands
- Fortunately, Unix provides the history mechanism
- This feature allows you to retrieve previous commands that you typed earlier
- You use this feature by pressing the *up* and *down* arrow keys

Retrieving Your Last Command Line Entry

- To retrieve the last command, simply hit the up arrow
- To retrieve the next to last command, hit the up arrow twice
- Once you have used the up arrow, you can then use the *down* arrow to go in the opposite direction
- Get in the habit of using the history feature
- It can save you a lot of typing, especially when you learn how to use the ***history*** command in combination with ***grep***...

Aborting a Running Program

- Most Unix commands execute quickly
- But some commands, like a compiler, can take a long time
- When you need to abort a running program use ***Control-C***
- This will work on most Linux/Unix systems
- We will also learn about other options later, such as:
  - Suspending
  - Foregrounding and backgrounding

Using Options with Unix Commands

- Most Unix commands have options which modify their behavior
- These options appear after the command
- You must type a space before entering the option
- Before the GNU project, most options used a single letter and were preceded by a *single dash*
- Most GNU utilities use options which consist of words preceded by *two dashes*
- Often, commands will support both option formats...

Using Options with Unix Commands

- Example:

```
$ cat --help
Usage: cat [OPTION]... [FILE]...
Concatenate FILE(s), or standard input, to standard output.

  -A, --show-all           equivalent to -vET
  -b, --number-nonblank    number nonempty output lines
  ...
```

Getting Help with Unix Commands

- Most Unix commands have a *help* option, which will provide a brief description of the command – along with a list of options:

```
$ ls --help
Usage: ls [OPTION]... [FILE]...
List information about the FILEs (the current directory by
default).
Sort entries alphabetically if none of -cftuvSUX nor --sort.

Mandatory arguments to long options are mandatory for short options
too.
  -a, --all                  do not ignore entries starting with .
  -A, --almost-all           do not list implied . and ..
        ...
```

- Depending on the command, you can use ***-h***, or ***--help***, or both to get help with it

The System Manual

- Unix comes with two extensive documentation systems
- The first system are the *man* pages
- To use these pages type *man*, followed by the name of the command:

```
man ls
```

- When you run *man* you will see a series of help pages...
  - Hit the ***Enter*** key to move down one line
  - Hit the ***spacebar*** to move to the next screen
  - To exit, type **q**
- You can move up and down through the *man* pages by pressing the *up* and *down* arrow keys

The System Manual

- The *man* pages are often very technical
- They can be quite intimidating, but you don't have to read all of the page
- You only have to read enough to answer your question
  - You might read the first few lines which show the arguments expected
  - Then you can skim down the list of options
  - Learn the art of reading *just enough to get the job done*
  - This has been called this "guerrilla reading"
  - Go in, get what you need, and get out

The *info* System

- Linux also provides an entirely different documentation system
- This system is menu-based and was created by the GNU project
- To enter this system use the *info* command
- You can follow *info* with the name of a command
- The up and down arrow keys will move up and down -- one line at a time
- The spacebar moves you down *one screen's height*

The *info* System

- The *info* system uses links
- Move down to a line with an asterisk ***** and hit Enter
- This will take you to a new page
- Type **h** for help
- Type **q** to quit
- For any utility created by the GNU project,
  the *info* documents are superior to the *man* pages

Searching for a Keyword with *apropos*

- *man* pages are useful, but only if you know the name of a command
- What if you don't know the name of a command?
- For this situation *apropos* was created
- Follow *apropos* with a key word, and it will give you a list of *man* page topics
- *apropos* takes the word you give it as an argument and searches the short description line at the top of all *man* pages for a match

Searching for a Keyword with *apropos*

- Example:

```
$ apropos who
at.allow (5)          - determine who can submit jobs via at or batch
at.deny (5)           - determine who can submit jobs via at or batch
bsd-from (1)          - print names of those who have sent mail
from (1)              - print names of those who have sent mail
rwho (1)              - who is logged in on local machines
rwhod (8)             - system status server
w (1)                 - Show who is logged on and what they are doing.
w.procps (1)          - Show who is logged on and what they are doing.
who (1)               - show who is logged on
whoami (1)            - print effective userid
whom (1)              - report to whom a message would go
```

Homework Directories

- When you first log in, you will be in your home directory
- You homework needs to go into another directory
- Each of you should have a class directory, it244, in your home directory
- For each homework assignment you **must** create a new homework directory inside the hw directory inside your it244 directory

Homework Directories

- I use a shell script to collect your homework
- If you put your homework in a different place, the script will fail
- Grading homework takes a lot of time
- Having to search for your work makes the grading take longer
- If you put your homework somewhere else **you will lose 10 percent of the possible total**

Using *script* in Homework Assignments

- Homework 2 is your first homework assignment using Unix
- Most of the assignment asks you to perform certain tasks on the Linux machine
- You have to use the *script* command to create a record of what you have done
- I will collect this file and use it to give you a score

Using *script* in Homework Assignments

- Your *script* session does not have to be perfect
- If you make a mistake while running *script*, simply try again
- But it is best for me if you practice what you need to do *before* you use *script*
- If you don't, the typescript files can become very long, which can make the grading take much more time than it has to
- So, please, run through the homework first without using *script*

Using *script* in Homework Assignments

- As you complete each step of the homework, you can cut and paste the Unix command into a text file
- Then when you are **sure** that you have everything working, run *script* copying command from your text file
- This will save me a lot of time
- Please **DO NOT** use an editor, like *nano*, while running *script*
- The control key sequences mess up the output
- This makes it hard for me to read and grade your homework