echo - Print Text to the Terminal

- *echo* simply prints whatever comes after it to the terminal

```
$ echo Hello world!
Hello world!
```

- It's used a lot in shell scripts to prompt for input or to let the user know what is happening
- It is also useful for checking the value of a variable

```
$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin
:/sbin:/bin:/usr/games:::
```

echo - Print Text to the Terminal

- When using *echo* with a variable you must precede the variable name with a dollar sign, $
- When echo is used with the `-n` option it does not advance to the next line
- This is useful in writing prompts inside shell scripts

hostname - Print the Name of Your Host Machine

- Every machine on the nextwork has a name
- This name is known as the **hostname**
- The *hostname* command prints the network name of the machine you are using

```
$ hostname
vm75
```

hostname - Print the Name of Your Host Machine

- it244a is a virtual machine that calls itself vm75
- When used with the -i option *hostname* will print the IP address of the host machine

```
$ hostname -i
192.168.106.240
```

Pagers - View a File One Screen at a Time

- **Pagers** show the contents of a file one screenful at a time
- Unix provides two paging programs, *more* and *less*
- *less* is an improved version of *more*
  - The name is an example of dry Unix humor
  - It's not *more* -- it's *less*
  - Both allow you to advance to the next screen by hitting the space bar
  - To move down one line, hit Enter
  - To get help, type "h"

Pagers - View a File One Screen at a Time

- *less* has more features
- In *less*, you can use the arrow keys to move up and down one line
- When *more* gets to the end of a file, hitting the Space bar will bring you back to the command line
- When you get to the end of a file in *less*, you must type "q" to quit

Pathname Completion

- When typing a long file name it is easy to make a mistake
- Unix helps with a feature called **pathname completion**
- We'll talk more about pathnames in a future class
- It works with directories as well as files
- You type a few characters, then hit the Tab key, and Unix will supply the rest....
- .....but only if there is only **one** file or directory that matches what you have started to type

Pathname Completion

- If there is more than one match Unix will supply as much of the name as it can and then beep
- If there is no match, it will also beep
- In the Bash shell, if you hit Tab twice you will see a list of all possible matches to what you have typed

grep - Finding Strings inside Files

- *grep* (**g**et **r**egular **ex**p**r**ession) is a utility which searches a text file for lines containing a specific **string**
- A string is a collection of characters that may, or may not, be a word
- *grep* uses the following format

```
grep [-OPTIONS] STRING FILE ...
```

- The words in capital letters are things that you must supply
- Anything enclosed in square brackets is optional
- Here is an example of how grep can be used...

grep - Finding Strings inside Files

- Say we have a text file

```
$ cat  red_sox.txt
2011-07-02	Red Sox @  Astros        Win 7-5
2011-07-03      Red Sox @  Astros        Win 2-1
2011-07-04      Red Sox vs Blue Jays     Loss 7-9
2011-07-05      Red Sox vs Blue Jays     Win 3-2
2011-07-06      Red Sox vs Blue Jays     Win 6-4
2011-07-07      Red Sox vs Orioles       Win 10-4
2011-07-08      Red Sox vs Orioles       Win 10-3
2011-07-09      Red Sox vs Orioles       Win 4-0
2011-07-10      Red Sox vs Orioles       Win 8-6
2011-07-15      Red Sox @  Rays          Loss 6-9
2011-07-16      Red Sox @  Rays          Win 9-5
2011-07-17      Red Sox @  Rays          Win 1-0
...
```

grep - Finding Strings inside Files

- ...and we want to find all the games the Sox won; we can use grep

```
$ grep Win red_sox.txt
2011-07-02      Red Sox @  Astros          Win 7-5
2011-07-03      Red Sox @  Astros          Win 2-1
2011-07-05      Red Sox vs Blue Jays       Win 3-2
2011-07-06      Red Sox vs Blue Jays       Win 6-4
2011-07-07      Red Sox vs Orioles         Win 10-4
2011-07-08      Red Sox vs Orioles         Win 10-3
2011-07-09      Red Sox vs Orioles         Win 4-0
2011-07-10      Red Sox vs Orioles         Win 8-6
2011-07-16      Red Sox @  Rays            Win 9-5
2011-07-17      Red Sox @  Rays            Win 1-0
...
```

grep - Finding Strings inside Files

- *grep* is case sensitive, unless you run it with the -i option

```
$ grep win red_sox.txt
```

- (NOTHING!)

```
$ grep -i win red_sox.txt
2011-07-02      Red Sox @  Astros       Win 7-5
2011-07-03      Red Sox @  Astros       Win 2-1
2011-07-05      Red Sox vs Blue Jays    Win 3-2
2011-07-06      Red Sox vs Blue Jays    Win 6-4
2011-07-07      Red Sox vs Orioles      Win 10-4
2011-07-08      Red Sox vs Orioles      Win 10-3
2011-07-09      Red Sox vs Orioles      Win 4-0
2011-07-10      Red Sox vs Orioles      Win 8-6
2011-07-16      Red Sox @  Rays         Win 9-5
2011-07-17      Red Sox @  Rays         Win 1-0
…
```

grep - Finding Strings inside Files

- The first command failed because I spelled "win" with a lowercase "w"
- If your search string contains one of Unix's special characters
- You must use a backslash to escape it or use quotes
- *grep* **-***r* will search **r**ecursively through a directory looking at all files in the directory and all the subdirectories

grep - Finding Strings inside Files

- *grep -v* returns all lines that **do not** match the search string

```
$ grep -v Win red_sox.txt
2011-07-04      Red Sox vs Blue Jays    Loss 7-9
2011-07-15      Red Sox @  Rays         Loss 6-9
2011-07-19      Red Sox @  Orioles      Loss 2-6
2011-07-25      Red Sox vs Royals       Loss 1-3
2011-07-28      Red Sox vs Royals       Loss 3-4
2011-07-29      Red Sox @  White Sox    Loss 1-3
```

grep - Finding Strings inside Files

- There are many more useful options for *grep* which can be found on the *man* page
- I use *grep* on an almost daily basis
- For example, let's say you want to add a field to a database table
- You could keep all my SQL code in text files
- Then, you can use *grep* to find every file that references that table

head - View the Top of a File

- Sometimes, the first few lines of a file are all you need to see
- *head* displays the first 10 lines of a file

```
$ head red_sox.txt
2011-07-02	Red Sox @  Astros	Win 7-5
2011-07-03	Red Sox @  Astros	Win 2-1
2011-07-04	Red Sox vs Blue Jays	Loss 7-9
2011-07-05	Red Sox vs Blue Jays	Win 3-2
2011-07-06	Red Sox vs Blue Jays	Win 6-4
2011-07-07	Red Sox vs Orioles	Win 10-4
2011-07-08	Red Sox vs Orioles	Win 10-3
2011-07-09	Red Sox vs Orioles	Win 4-0
2011-07-10	Red Sox vs Orioles	Win 8-6
2011-07-15	Red Sox @  Rays	Loss 6-9
```

head - View the Top of a File

- If you give *head* a number as an option it will display *that number* of lines

```
$ head -5 red_sox.txt
2011-07-02      Red Sox @  Astros          Win 7-5
2011-07-03      Red Sox @  Astros          Win 2-1
2011-07-04      Red Sox vs Blue Jays       Loss 7-9
2011-07-05      Red Sox vs Blue Jays       Win 3-2
2011-07-06      Red Sox vs Blue Jays       Win 6-4
```

tail - View the Bottom of a File

- *tail* is like *head* except it prints the **last** 10 lines of a file

```
$ tail red_sox.txt
2011-07-22      Red Sox vs Mariners     Win 7-4
2011-07-23      Red Sox vs Mariners     Win 3-1
2011-07-24      Red Sox vs Mariners     Win 12-8
2011-07-25      Red Sox vs Royals       Loss 1-3
2011-07-26      Red Sox vs Royals       Win 13-9
2011-07-27      Red Sox vs Royals       Win 12-5
2011-07-28      Red Sox vs Royals       Loss 3-4
2011-07-29      Red Sox @  White Sox    Loss 1-3
2011-07-30      Red Sox @  White Sox    Win 10-2
2011-07-31      Red Sox @  White Sox    Win 5-3
```

tail - View the Bottom of a File

- You can give *tail* a number as an option to specify the number of lines printed

```
$ tail -4 red_sox.txt
2011-07-28      Red Sox vs Royals       Loss 3-4
2011-07-29      Red Sox @  White Sox    Loss 1-3
2011-07-30      Red Sox @  White Sox    Win 10-2
2011-07-31      Red Sox @  White Sox    Win 5-3
```

- *tail* is especially useful when looking at log files
  - Log files are text files that record significant events
  - They are created automatically by programs that provide services like a web server
  - The most recent entries are at the end of file and those are usually what you want to see when you are trying to solve a problem

sort - Print a File in Sorted Order

- Unix has a number of utilities for manipulating text files
- *sort* prints the contents of a file with the lines sorted

```
$ cat  fruit.txt
grapes
pears
oranges
cranberries
apples
melons
blueberries
```

sort - Print a File in Sorted Order

- Now, let's sort those lines...

```
$ sort  fruit.txt
apples
blueberries
cranberries
grapes
melons
oranges
pears
```

***sort* - Print a File in Sorted Order**

- *sort* looks at the beginning of each line of a file and sorts the line based on these characters
- ***sort* does not change the file itself**
- It simply prints the sorted contents of the file to the terminal
- *sort* has many useful options, which you can find in the *man* pages:
  - numeric sort
  - case-insensitive
  - unique detection

sort - Print a File in Sorted Order

- *sort -r* (**r**everse) will sort the file in reverse alphabetical order

```
$ sort  -r  fruit.txt
pears
oranges
melons
grapes
cranberries
blueberries
apples
```

sort - Print a File in Sorted Order

- *sort -n* (number) will sort a file by number ...
- so a line starting with "2" will appear before a line starting with "11"

```
$ cat  numbers.txt
11
1
17
2
3
15
...
```

```
4     9
5     10
14    12
6     16
13    18
7     19
8     20
```

sort - Print a File in Sorted Order

```
$ sort numbers.txt
1
10
11
12
13
14
15
16
17
...
```

```
18
19
2
20
3
4
5
6
7
8
9
```

sort - Print a File in Sorted Order

```
$ sort  -n  numbers.txt
1                                   7           14
2                                   8           15
3                                   9           16
4                                   10          17
5                                   11          18
6                                   12          19
                                    13          20
...
```

- *sort* does not change the file on which it is run; it merely prints the contents of the file in sorted order

uniq - Eliminate Duplicate Lines

- *uniq* prints a text file, removing **adjacent** identical lines

```
$ cat numbers2.txt
4
5
6
7
8
...
```

```
9     14
10    15
11    16
11    17
11    18
12    19
13    20
```

uniq **- Eliminate Duplicate Lines**

```
$ uniq numbers2.txt
4                                   10      15
5                                   11      16
6                                   12      17
7                                   13      18
8                                   14      19
9                                   15      20
```

- Note, the duplicate lines must be **adjacent** for *uniq* to work
- For this reason, it's good to use *sort* along with *uniq*
- *uniq -i* ignores case when looking for duplicate lines

diff - Differences between Files

- *diff* compares two files and displays the lines that are different
- The default output format of *diff* is confusing

```
$ cat numbers1.txt
1                                   8       15
2                                   9       16
3                                   10      17
4                                   11      18
5                                   12      19
6                                   13      20
7                                   14
...
```

diff - Differences between Files

```
$ cat numbers2.txt
4
5
6
7
8
9
10
11
11
...
```

```
11
12
13
14
15
16
17
18
19
20
```

diff - Differences between Files

```
$ diff  numbers1.txt  numbers2.txt
1,3d0
< 1
< 2
< 3
10a8,9
> 11
> 11
```

- *diff* was created for use with the Unix *patch* utility, which creates a new version of a file from the changes given by *diff*
- **This output was never meant to be read by people!**

diff - Differences between Files

- The most readable output is obtained by using the -y option

```
$ diff  -y  numbers1.txt numbers2.txt
1                               <
2                               <
3                               <
4                               4                               11                              11
5                               5                               12                              12
6                               6                               13                              13
7                               7                               14                              14
8                               8                               15                              15
9                               9                               16                              16
                                                                ...
```

- *diff -i* ignores case when looking for differences!