Helpful Commands for Linux
All your files are contained in your own directory. The name of your directory is your user name (bjabram is mine) and is located in the /home directory. Your personal directory should have permissions set at 755 if you want it to be web accessable (see chmod below).

cd (directory) – changes directories
- Current directory is denoted as one period (.)
- Host directory (the directory directly above the current directory) is denoted as two periods (..)
- Root directory is denoted as a backslash (/).
- Your personal home directory is denoted as a tilde followed by your own username (i.e. ~bjabram)

ls [arguments] [directory] - returns the listing of files for the current directory (or the specified directory, if given).
- -l lists files in one-column format, showing size, permissions, etc.
- -a lists all files, including system files and dot-files

cp (source) (destination) - copies files or directories from one location to another.
- Both the source and destination are required.

mv (source) (destination) - moves files or directories from one location to another or renames a file
- Both the source and destination are required.
- To move the file, the destination should be a directory
- To rename the file, the destination should be another file name.

chmod (octal) (source) - alters the permissions of a given file or directory
- Argument is a three-digit number, each digit between 0-7 (octal number)
- Each digit of the three-digit number refers to a person's permissions: user, group, and others
- A person has three possible permissions: read, write, and execute; there are eight combinations of these
- Read = 4, Write = 2, Execute = 1; add these up for each person to get the three-digit number
- At the beginning of the line in a long directory listing (ls –l *filename*) permissions are given in a ten-character code, separated in four areas. The first character says whether the filename is a file (-), directory (d) or link (l). The next three give user's permissions (rwx, for example), the next three give group's permissions (not used for this class; keep group's the same as others) and the final three give other's permissions.
- chmod 755 for Perl scripts (executed from the command line) and directories
- chmod 644 for Perl scripts (executed from perl program), html scripts, and graphics files

pico [file] - the standard editor in this class.
- CTRL-O saves a file
- CTRL-W finds a string in the file
- CTRL-X exits pico. You'll be asked if you want to save the file if you altered it.

man [-k] (keyword or command) – invokes the online reference manual
- Used to get help on commands like ls, cp, mv, etc.
- Searchable; use the –k option and enter a keyword to find suitable commands

Perl Scripts and Other Programs
- Programs in the same directory must be run with "./" in front of it for security reasons
- Programs for Perl can be in any directory (I recommend /home/username/math210 for reasons of ease)
- Programs for the web <u>must</u> be somewhere in the html/cgi-bin directory (home/username/html/cgi-bin) or directory hierarchy (home/username/html/cgi-bin/math210/dayoftheweek, for example)

Helpful Commands for HTML

Most HTML commands are like parentheses: they need both the opening "(" and closing ")" types
Opening tags are enclosed with < and >, while closing tags are enclosed with </ and >
Exceptions for our purposes are <img>, <input>, and <submit> tags
Attributes of tags are contained in the opening tag (<img src="location">, <body bgcolor=white>)
HTML scripts always start with <HTML> and end with </HTML>
HTML is pretty lax on syntax; if you forget something, the page will still be displayed (just not how you want it)
The <HTML> tag has two main subsections: <HEAD> and <BODY>

<HEAD> elements

- <TITLE> gives the title that appears in the menu bar at the top of the browser window
- <SCRIPT> can be used to give JavaScript statements (not used in our class)

<BODY> elements

- <p> creates new paragraph (align={left, center, right}) (note: </p> not mandatory, but helpful)
- <h1>...<h6> creates headers (align={left, center, right}) (</h1>...</h6> is mandatory)
- <br> breaks to a new line (note: </br> does not exist)
- <hr> creates a horizontal line (note: </hr> does not exist)
- <blockquote> tabs the left margin about one inch
- "&nbsp;" is a non-breaking space; if the phrase "Math 210" was wrapped on the screen, "Math&nbsp;210" wouldn't be wrapped. This is handy for blank table cells (see below).
- <b>, <u>, <i> creates bold, underlined, and italicized text
- <sub> and <sup> creates subscript and superscript text
- <a> creates an anchor or link to other webpage (href="location of file" name="text")
- <img> displays a picture on the screen (src="location of file", border={0,1}, width, height)
- <ol> and <ul> creates ordered (numbered) and unordered (bulleted) lists; <li> creates the list's elements (you must end your list with </ol> or </ul>, but it is not mandatory to end each element with </li>)
- <form> creates a form (method={get, post} action="location of perl script").
- <table> creates a table (uses <tr> and <td> for new row and new cell, respectively)

THE BEST WAY TO LEARN HTML IS TO LOOK AT THE SOURCE OF OTHER HTML DOCUMENTS!!!
You can do this through by selecting "Source" from the View menu in either Internet Explorer or Netscape

HTML Documents and Apache Web Server:

- johnny runs Apache as its web server; this is the program that sends documents to the remote user
- HTML documents should have 644 set as their permissions.
- Your URL http://johnny.sas.upenn.edu/~username. When this is called, Apache looks for index.html in your html directory (/home/bjabram/html). If it does not find this file (or others like it, index.htm, home.html, etc) then it will display the current contents of the directory. This goes for all other directories under the html directory. ONLY THE HTML DIRECTORY IS VISIBLE TO APACHE!
- All programs that you wish to be executed by Apache must be in the html/cgi-bin directory (home/username/html/cgi-bin). These types of programs are usually accompanied by an HTML document with a form (<FORM>) in it.

A sample HTML script, which can be found at http://johnny.sas.upenn.edu/~bjabram/index.html

```
<html>
<head>
<title>Math 210</title>
</head>
<body background="graphics/graph.gif" bgcolor=white>
<img src="graphics/banner.gif" width=576 height=72 border=0>
<h3>News</h3>
<ul>
<li>January 8, 2002 - Welcome back!</li>
</ul>
<h3>Information</h3>
<ul>
<li><a href="hw">Problem Sets and Solutions (/homework)</a></li>
<li><a href="math210">Sample Perl Scripts (/math210)</a></li>
<li><a href="doc">Maple Scripts and PDF Files (/doc)</a></li>
</ul>
<h3>Help Topics</h3>
<ul>
<li><a href="help/telnet.html">Help with Telnet and FTP</a></li>
<li>Help with Perl and CGI: see <a
href="http://www.math.upenn.edu/~kazdan/210/bib.html#perl">Dr. Kazdan's Perl
Help</a></li>
<li><a href="help/vnc.html">Help with VNC</a></li>
<li><a href="http://www.cas.lancs.ac.uk/glossary_v1.1/prob.html">Statistics
Glossary (with formulas for probability)</a></li>
<li><a href="http://einstein.et.tudelft.nl/~arlet/puzzles/probability.html">A
nice collection of statistics and probability problems (with
answers)</a></li>
</ul>
<p>
If you would like more information about the class, e-mail <a
href="mailto:bjabram@sas.upenn.edu">me</a> or <a
href="mailto:kazdan@math.upenn.edu">Dr. Kazdan</a>.
</body>
</html>
```

Helpful Commands for Perl
All Perl scripts should start with #!/usr/bin/perl
On johnny, the perl binaries are located at /usr/bin/perl (differs on other machines, such as sas or seas)
Comments are designated with the pound sign (#)
All statements must end with a semicolon (;)
All operators are similar to standard math (+, -, *, /, >, <, <=, >=), with the exception of exponent (**)
Placing a exclamation point (!) before a clause negates it (1 + 1 = 1, but 1 + 1 != 3)

Variables, Strings, and Arrays
- All variables and strings have a dollar sign ($) before them ($n, $susan)
- Arrays are designated with an at-sign (@) before them (@array1, $n)
- Elements in arrays are called with brackets: if A is an array, then $A[0] is its first element, etc.
- Elements in arrays can be set just like ordinary variables ($A[0] = 5)
- Strings can be added together with the period (.) ("Cow" . "boy" would equal "Cowboy")
- The string "\n" prints a carriage return, like hitting Enter on the keyboard (i.e. "Cowboy\n")

Print statement
- syntax: *print "string";*
- The string may be made up of ASCII characters ("n"), variables ("$n"), and special codes ("\n")
- If $n = 2, then *print "n = $n";* would print "n = 2" on the screen

<STDIN> and chomp (variable) statements
- syntax: *$input = <STDIN>; chomp($input);*
- Used to gather input from the user's keyboard
- $input can take any valid name ($n, $hey,…)
- chomp removes the last character from the string; this is necessary because when the user hits return to enter a value, the return character (\n) is added to the string. You will probably want to get rid of this excess. (Consider chomping the string a bug in Perl that we need to overcome)

for loops:
- syntax: *for (init; clause; run)*
- *init* command is run first; usually this initializes a temporary variable for the loop (ex. $i = 0)
- *clause* is a binary clause checked at the beginning of each loop; loop runs if true, stops if false
- *run* is a command run at the end of each loop

if clauses:
- syntax: *if (clause) {statements}*
- *clause* is a binary clause; if true, then the statements are run
- *statements* are a set (one or more) of commands that are run as a group if the *if* clause is true
- *else {statements}* can follow an if-loop; if the *if* clause is false, then the *else* statements are executed instead
- Checking equality is done with two equal signs (==).

while loops / until loops:
- syntax: *while (clause) {statements}* and *until (clause) {statements}*
- These loops can be considered looping if-clauses
- After the statements are executed, the clause is checked for validity again
- until loops parallel while loops; the statements in while loops are executed while the clause is true, whereas the statements in until loops are executed until the clause is true (or while the clause is false).
- for loops can be converted into while or until loops by including the *init* statement before the loop and including the *run* statement at the end of the loop (but inside it). There may be cases where a while or until loop is more appropriate than a for loop (although no examples come to mind).

Some sample Perl Scripts can be found (soon) in my directory at /home/bjabram/html/math210 (johnny.sas.upenn.edu/~bjabram/math210) or Dr. Kazdan's site (www.math.upenn.edu/~kazdan/210/computer/perl).

A sample Perl script, which can be found at http://johnny.sas.upenn.edu/~bjabram/math210/sample.pl

```
#!/usr/bin/perl

#
# sample.pl - This program calculates the sum of all numbers divisible
# by n between 1 and m.
#

print "This program finds the sum of all numbers divisible by n between 1 and
m.\n";
print "Enter n, the dividend    : "; $n = <STDIN>; chomp($n);
print "Enter m, the upper bound : "; $m = <STDIN>; chomp($m);

$c = 0;                              # Temporary counter to add up squares

for($i = 1; $i <= $m; $i++) {        # All numbers between 1 and $m, inclusive
  if($i % $n == 0) {                 # If $i mod $n equals 0 (divisible)
    $c = $c + $i;                    # Add $i to current value of $c
  }
}

print "The sum of all numbers divisible by $n between 1 and $m is $c.\n";
```