Command-line Basic Concepts

Contrary to popular perceptions, GUIs (Graphical User Interfaces) are not always superior to working from the command-line. In fact, when we are talking about running a simulation program with hundreds, thousands, or even more variations of input, automating the process becomes essential.

Here are a handful of useful commands and concepts for a Mac/*nix environment. Arguments between square brackets are optional (and you don't actually type the brackets). In most cases the command for Windows/DOS is identical, I have noted the equivalent DOS command when it is different.

STDIO, file redirection, appending, and piping

Standard Input/Output is a concept used by both Mac/*nix and DOS systems. By default, STDIO attaches both output (STDOUT) and input (STDIN) to the terminal/console. This means that programs based on STDIO will read what you type into the terminal, and will display their results back to the terminal. However, this default behavior can easily be overridden. Appending "< inputFile" after a STDIO program will tell it to get its input from a file named "inputFile" - replace "inputFile" with the actual name of the file containing input data. Appending "> outputFile" will send the output of the program to the specified "outputFile", overwriting it if the file already exists or creating the file if it didn't already exist. If a file already exists and you wish to append to its current contents, replace the single greater-than (">") with a double greater-than (">>"). Both input and output redirection can be used, but only one of ">" or ">>" can be specified. The final variant of STDIO redirection is called piping, and is represented by a vertical bar ("|"). When placed between two programs, it makes the STDOUT of the first program become the STDIN of the second program, using no intermediate file storage.

Examples:

```
# run command "cmd" with input from file "infile", output goes to console
cmd < infile

# run command "cmd" with input from file "infile", destructive output to "outfile"
cmd < infile > outfile

# run command "cmd" with input from file "infile", appending output to "outfile"
cmd < infile >> outfile

# run command "cmd1" with input from file "infile", output from "cmd1" becomes
# the input for "cmd2", output from "cmd2" saved in "outfile" destructively
cmd1 < infile | cmd2 > outfile
```

`ls` [**`-l`**] [explicitlyNamedDirectory]
Give a directory listing of the current directory if none is provided, or the explicitlyNamedDirectory if one is provided. The **-l** argument will give a "long" listing describing file size, ownership, and access/execution permissions. The equivalent DOS command is "**dir** [**/w**]".

`cd` [**`..`** OR ~ OR explicitlyNamedDirectory]
Change directory to the specified location. Double dot ("..") means move up one level from the current location. A tilde ("~") means change to your home directory, which is also the default if no argument is provided. Providing an explicit directory path will move focus to that location.

`cat` [list of files]
concatenate the contents of the list of files into a single output stream on STDIO. The output stream can be redirected or piped if so desired. If no list of files is proved, the **cat** command uses STDIN for input. A roughly equivalent DOS command is "type".

`more` or **`less`** [list of files]
Display the contents of the list of files, or of STDIN if no list is provided, one screenful at a time on the console to make it easier for humans to browse large files. Both variants are searchable, but **more** can only search in a forward direction while **less** allows you to search and scroll both forward and backward. DOS has the **more** command but doesn't provide a **less** command.

`mkdir` dirName
Create a new directory with the specified name.

Running a Ruby script
If a Ruby script has the proper header line as its first line, has been set to be executable, and is in a location on your environment's execution PATH, just typing the name of the script will run it.

If the script has not been set to executable or is missing the header line, explicitly typing "`ruby scriptname.rb`" will run it. Replace `scriptname.rb` with the actual name of the script. If such a script is not in the current folder, you will need to specify its fully qualified name to run it.

DOS doesn't use the same concept of executable files, it decides whether a file is executable or not based on its suffix. Ruby must be manually installed on DOS systems. If Ruby was properly installed, files with a ".rb" suffix will be executable.

Running a Java program from an executable jar file
Java is not installed by default on either Windows or Mac systems. After downloading and installing a suitable Java Runtime Environment (JRE), a properly constructed Java Archive (jar) file is run by typing "`java -jar filename.jar`", replacing `filename.jar` with the actual name of the jarfile.