Introduction to Big Data Analysis for Scientists and Engineers

About this white paper: This paper was written by David C. Young, an employee of CSC. It was written as supplemental documentation for use by the HPC account holders at the Alabama Supercomputer Center (ASC). This was originally written in 2012, and updated in 2013.

There have been a number of pushes over the years to make the most of existing data. These have come with corporate buzzwords like "Big Data", "Analytics", "Data Mining", and "Decision Support". Although the media has focused on the successes of data rich companies like Facebook, LinkedIn, Amazon, and Google, there are also some good ideas here that can be useful to people in science and engineering fields. The following is a general discussion of these ideas.

For HPC users at ASC: The subdirectories underneath /opt/asn/doc/data_analysis contain a number of interesting examples.

Commercial Examples

Companies try to use the data available to them to improve their business. Google attempts to identify when a search term is usually looking for a local business and will thus show results for businesses near you. LinkedIn improved their volume by suggesting "People You May Know". Amazon suggests package deals of bundling product A with product B, or telling you what other products were purchased by people that got the one you just added to your cart. These simple improvements have been responsible for millions if not billions of dollars of additional revenue.

In addition to the customer-facing examples mentioned in the previous paragraph, many companies analyze their old sales data to see how their business has historically fluctuated with their pricing, the economy, and their competitor's activities. This analysis can be used for future planning, and current pricing.

At the time this was being written, the current buzzword was "Big Data". This movement points to Internet companies as examples, but it is more generally the use of available data, which is often unstructured data. Unstructured data is information that is not already organized into a relational database. It might be in text files, publicly available on the Internet, or (perhaps worst of all) in some proprietary format.

Applying Big Data Analysis Ideas to Science and Engineering

Certainly the use of statistics is nothing new to those in scientific and technical fields. What may be new to many is the use of a wealth of tools, technologies and data sources that are relative new comers to the world. For example, social scientists can now access an incredibly large amount of data generated on social network sites. While there is data there, it is not as neatly organized, controlled, or structured as a scientifically conducted survey. Thus the opportunity comes with a whole new set of issues to be addressed.

This document is meant to be an introduction to these ideas. If successful, it will give you some things to think about and some examples of how to perform various functions using tools available on many High Performance Computing (HPC) systems and available for desktop computers.

Obtaining Data

Most users of the HPC systems transfer files between their computer on campus and the login node with commands like "sftp" and "scp" or some Windows based graphic front end that runs these same protocols. Often people start with "sftp" as they are attracted to the interactive nature of it. A typical sftp session may involve typing the following commands

```
sftp myname@hpc.cluster.name
type your password
cd calc
cd gaussian
cd tests
get test002.com
get test002.log
     :
     :
quit
```

Within sftp, the process of getting each file could be made easier by typing a command like this.

```
mget test002*
```

This command results in getting all of the files that have names starting with test002

People that use systems heavily, particularly Linux and Macintosh systems, often switch to using "scp" in place of "sftp" most of the time. This is because it takes far less typing to transfer a large number of files. For example, the following command transfers a directory including all files, subdirectories, and files in subdirectories from a remote computer to the local system.

```
scp -r myname@hpc.cluster.name:~/calc/gausian/test .
```

This command asks you to type your password, then transfers all of those files. The period indicates that the local copy of the test directory should be placed in the directory you are in when you type the command.

Example of using wget

Not all data is sitting in your directories on machines where you have accounts. It is sometimes desirable to download files off of a web site. You could do this by clicking on files in your web browser, but this can get tedious if there are hundreds of files and impossible if there are tens of thousands. The following is an example of using the "wget" command to automate this task.

For HPC users at ASC: The directory **/opt/asn/doc/data_analysis/wget-example** has sample files to accompany this wget example. However, you can generate the same files by following the steps below.

"wget" is a Linux command for downloading files from the web. It is non-interactive, meaning that it can be put in a script or left to do it's work without having to type things or click on anything. To see the manual page for wget, type "`man wget`"

For this example, let's say that you would like to analyze trends in the computational chemistry field by looking at all of the messages posted to the Computational Chemistry List. It is an email list server, which as an archive of previous messages available on the web at **http://www.ccl.net** The file and directory structure is not published, so the following paragraphs walk you through how to figure it out.

If you browse to one of the archived messages from the list, you find that it has a URL like this **http://www.ccl.net/cgi-bin/ccl/message-new?1994+08+24+002** The web page shows the message, which is the data you want, but it has other information and navigation links above it and to the left. You can download this web page with the command;

```
wget http://www.ccl.net/cgi-bin/ccl/message-new?1994+08+24+002
```

This puts a file named **message-new?1994+08+24+002** in your directory. This is a text file, so you can look at the file contents with a number of Linux utilities such as "more" or "nano".

This file is not what you really wanted. It defines the frames that divide the web page into a top section, left section, and the area with the message you really wanted. Looking at those frame tags, it would be a good guess that the message you really want is defined by the following tag.
<frame name="message" src="/chemistry/resources/messages/1994/08/24.002-dir/index.html" />

This has the location of a web page, but as a relative location in the directory tree with no "http" out front. You can add the site's address and try loading the resulting web address in a different tab of your web browser using a URL like this
http://www.ccl.net/chemistry/resources/messages/1994/08/24.002-dir/index.html

This gives the desired data in the web browser, so you can download that file with the command.

```
wget http://www.ccl.net/chemistry/resources/messages/1994/08/24.002-dir/index.html
```

This puts a file named index.html in your directory. Looking at this text file, you see that it does have the message text that you wanted, but also a bunch of html tags that you don't really want to look at. We'll discuss manipulating the contents of files in another example.

Now we want to get all of the messages. The simplest idea is to use the "-r" flag to wget to recursively get all of the messages, say from August of 1994, as shown below.
WARNING: DO NOT RUN THIS COMMAND.

```
wget -r http://www.ccl.net/chemistry/resources/messages/1994/08/
```

This command does not do what you expected. It does get the messages posted to the CCL in August of 1994, but it also recursively follows the links in those html pages. By following those links, it downloads the index of messages in August of 1994, which is linked to the index of all months in all years, which is linked to the site main page with links to archives of other types of software, data, images, etc. Many hours and gigabytes later you will have downloaded everything on the CCL web site that has any link pointing to it. You also will have had information about what wget is doing continuously scrolling in the window.

Taking the wget recursive flag a bit more cautiously, we add the "-np" flag which tells it not to go into the parent directories, thus preventing it from picking up the rest of it's own web site. We also add "-D ccl.net" to specify the domain so that it will not follow links to any other web sites and thus attempt to download the entire web. We also add "-o log.txt" to put all of those scrolling messages into a file named log.txt Now our command to download messages from August of 1994 is as follows.

```
wget -r -np -D ccl.net -o log.txt http://www.ccl.net/chemistry/resources/messages/1994/08/
```

NOTE: If you don't include the slash after 08 you will get all of 1994 because the syntax told it no parents of 1994 instead of no parents of 08.

The command above will take a few minutes to download 22 megabytes of data. It maintains the directory structure by creating the directory tree **www.ccl.net/chemistry/resources/messages/1994/08** but the **08** directory is the only one that contains something other than the one subdirectory.

A bit of looking at the resulting files and directories shows that each message is in raw form with a name like **18.007**, has a little html file pointing to it with a name like **18.007.index** and has the message with html tags included in a directory with a name like **18.007-dir/index.html** We can quickly remove the extra files and directories with a few command like this

```
rm -r *dir
rm *index*
```

Now we have a directory with just the raw email messages from August 1994. Our 22 MB of downloaded data has now been reduced to the 5 MB that we really wanted.

Macintoshes come with "curl" in place of wget. Many Linux distributions include both wget and curl. There are many other utilities for transferring data over networks, each giving improvements in specific situations. Most notable are multithreaded transfer programs, which can move data significantly faster than wget. The trade off is that wget is designed to be robust and reliable even if various types of network problems are encountered. If you are moving large amounts of data every day for months, it might be worth investigating other tools, most of which do not come with the operating system.

Structuring Data

The wget example shown above pulled down a bunch of email messages from a list server devoted to computational chemistry. This data isn't tabulated, sorted, or annotated. The only existing structure consists of things common to all email messages, which could be about anything.

Linux is the dominant operating system for servers and high performance computing systems. One big reason for this is that the Linux command set provides an incredibly flexible toolbox that can be used to manipulate information, particularly data that is available as text in files or as the output from other programs. The following examples are based on working with the files in the **08** directory that was created in the wget example above.

Examining File Types

Change directories into the **08** subdirectory.

First consider that files may contain text, programs, or binary data. Sometimes the file name suggests what is in the file, but that is not a guarantee. Also, there are thousands of file names that used for text files with various types of data in them. Use the Linux "file" command to see what type of files these are like this

```
file *
```

The "file" command looks inside the files and gives it's best description of what type of data is in them. Most say "ASCII mail text", but a few say "ASCII English text", or "ASCII Pascal program text", or simply "data".

Take a look at the content of one of the files identified as "ASCII mail text". Use your favorite tool for viewing text files such as "more" or "nano". The first few lines of the file contain things you would expect in an email message like "From:", "To:", "Date:", and "Subject:". These are followed by the typical text of an email message.

Now take a look at a few of the files that the "file" command identified as being something other than email messages. These are also email messages, but they are different somehow. Some don't contain all of the header information that most email programs put on messages. Some have stray binary characters. If the message contained a programming example, the "file" program may have classified it as source code rather than email. One reason for using the Linux data handling commands is that they can robustly handle this variation in input data.

Extracting Data

Now let's do something with this data. Perhaps we want to know which software packages the computational chemists are talking about this month. Let's look for discussions about the Gaussian program.

The "grep" command extracts lines that contain a given sequence of characters. For example, we can look for the word "Gaussian" with the command

```
grep Gaussian *
```

This shows lines from all of the files (the wild card "*" meaning all was put where we give grep a file name). However, our grep command did not show instances of the word "gaussian". We can look for both capitalizations like this.

```
grep [Gg]aussian *
```

The "grep" command supports a list of options and regular expression matching patterns. There are books and web sites with examples of using grep in ways that have never occurred to you.

We may also want to look for instances of GAUSSIAN or other capitalizations. Case insensitive searching is common enough that grep has a special flag for it, like this.

```
grep -i gaussian *
```

If we want to determine how many lines contain the word Gaussian, we can combine this with the "wc" command, which stands for word count. Counting lines is done like this

```
grep -i gaussian * | wc -l
```

This found 68 occurrences of the word Gaussian. The vertical bar (shift-backslash) is called a pipe and tells Linux to send the output from one command into the next command as it's input.

We may want to distinguish between messages where Gaussian is the main topic of discussion from ones where it was mentioned in passing. In email messages, the topic is usually on a line that starts with the word "Subject:". More specifically, the word "Subject:" should start at the left most letter of a line. The beginning of a line is denoted by a carat (^) like this.

```
grep ^Subject: * | grep -i gaussian | wc -l
```

Now we find out that five messages contained gaussian in the subject line.

Sorting

Perhaps we would like to see the list of subjects discussed this month. We can extract the Subject lines like this

```
grep ^Subject: *
```

The first 16 characters of each line contain the file name and the word Subject: which is unneeded. Keep the information from the 17th character on in each line, like this.

```
grep ^Subject: * | cut -c17-
```

These topics would be easier to look through if they were alphabetized like this.

```
grep ^Subject: * | cut -c17- | sort
```

Some topics such as "This is a test" are listed multiple times. We can remove the duplicates by asking the sort command for unique entries only with the following command.

```
grep ^Subject: * | cut -c17- | sort -u
```

Extracting Columns

We may want to collect metrics on how long the email messages are. We can see the file sizes in bytes with the command

```
ls -l
```

The file sizes are in the fifth column. The numbers in that column may be three, four, five, or more digits long. Thus the numbers don't always start at the same position, so the "cut" command is a poor choice. The awk more robustly handles variable numbers of white space characters, like this.

```
ls -l | awk '{print $5}'
```

Unlike the "expr" command, awk can also be used to do floating point mathematics. For example, we might extract the file sizes, then compute the average file size like this.

```
ls -l | awk '{sum+=$5} END { print "Average = ",sum/NR}'
```

awk is an interpretive programming language in it's own right. However, it is often seen filling in the gaps that other shell scripting functions can't handle.

Creating Files of Formatted Data

The “tr” command translates one character into another. This can be a handy tool for reformatting data.

For example, we can make a list of the files in the directory and their file sizes like this.

```
ls -l | awk '{print $8,$5}'
```

The two columns are separated by a space. That space can be changed into a comma with the tr command like this.

```
ls -l | awk '{print $8,$5}' | tr ' ' ','
```

Then the resulting information can be put into a file called **files.csv** like this

```
ls -l | awk '{print $8,$5}' | tr ' ' ',' >files.csv
```

The extension .csv is a common way of denoting the contents of the file as "comma separated values". Many programs, such as spreadsheets, can import data in this format.

Visualization

Once you have data, the first order of business is often to get some sort of understanding of the data. The term "visualization" refers to a wide range of techniques for displaying data as pictures or graphs of some sort.

See the white paper "Getting Started with Visualization" for a discussion of various visualization techniques and software package used to generate those images.

For HPC users at ASC: See the /opt/asn/doc/data_analysis/heatmap-example directory for an example of how geographically distributed data can be used to create a colorized map, in this case a map of the state of Alabama showing where the supercomputer users are located. This gives another example of extracting usable data from a data source that, at first glance, doesn't seem to give you the correct information. The “Getting Stared with Visualization” paper is at
/opt/asn/doc/visualization/visualization.pdf

Statistical Analysis

Shell scripts are handy as a way to quickly do simple mathematical calculations. However, they aren't the best option for advanced mathematical and statistical analysis of data.

For small data sets, a spreadsheet program such as Excel can be a very convenient way to generate statistical descriptions of rows or columns of data. However, spreadsheets have a limit on the number of rows and columns they can handle. Depending upon the version of Excel, the maximum number of rows ranges from 65,536 to over 1 million.

A popular tool is "R". R is an environment for data handling, storage, analysis and display (providing you logged in via X-Windows). R can be used interactively through a non-graphical ssh session. It can display a variety of graphs provided you logged in through and X-Window capable ssh session. It can also be run in batch mode and run through the queue system. R implements a programming language called "S". R can be used for a variety of programming tasks, but it's strength is it's utility for mathematical, and in particular statistical, analysis of data. R is well documented in the form of a number of pdf files. The R-intro.pdf document is a good place to start. These can be found on the R web site at **http://www.r-project.org/**

For HPC users at ASC: The R documentation is in the directory /opt/asn/doc/R

Octave is a public domain clone of Matlab. Octave will run Matlab .m files, but will not run precompiled Matlab programs and does not have the Matlab toolboxes.

For HPC users at ASC: The octave documentation is in the directory /opt/asn/doc/octave

R and Octave are interpretive languages. This means that they can be used interactively, and commands typed into them are converted into machine language for execution on the fly. The disadvantage of interpretive languages is that they run slower and require more memory than programs written with compiled languages, such as C++. We have had a case in the past where a researcher that had been using Matlab rewrote their program in C++ and found that it ran far faster and used 1/10 the memory, thus allowing them to do much larger simulations with the C++ version. When the performance of an interpretive language becomes unacceptable, it is time to switch to a compiled language, specifically C, C++, or FORTRAN. The current generation of compilers tends to give the fastest executing program with C++, although the difference between the three is fairly small.

Further Information

For a discussion of how big data is used to increase profits in the business world, see the October 2012 issue of the Harvard Business Review.

Some examples of manipulating data with Unix commands are at
http://www.drbunsen.org/explorations-in-unix.html

There is much more that can be done simply with shell scripts calling programs that come with most Linux distributions. There are some more examples in Chapter 7 of the HPC User Manual at http://www.asc.edu/html/man.pdf

Also, see the following books.

Burtch, Ken. Linux Shell Scripting with Bash. Indianapolis, IN, Sams Publishing. 2004.

Robbins, Arnold, Nelson Beebe. Classic Shell Scripting. Sebastopol, CA, O'Reilly & Associates, Inc., 2005.

Cooper, Mendel. Advanced Bash-Scripting Guide.
http://www.tldp.org/LDP/abs/html/

Kochan, Stephen, Patrick Wood. Unix Shell Programming. Carmel, IN, Hayden Books, 2003.

Some Linux commands are rich enough in functionality that there is an entire book devoted to that one command.

There are also quite a few examples that can be found by a simple web search. However, it is highly advisable that you take the time to understand commands you find on web searches. Many will do something similar to what you want, but not exactly correct.