What you can do with corpora using just standard Unix command line tools?

There are a lot of specialist pieces of software for dealing with text corpora, and of course you can also write programs, but working with corpora invariably involves various operations of moving, munging, and reformatting text, and it's really useful to have an understanding of how much of that can be done with standard Unix tools without having to write programs. This also makes it a lot easier to do exploratory data analysis.

Unix was initially invented in the context of supporting text processing, and developed many tools that work on text files. Moreover, the original Unix philosophy concentrated on small tools that do one job well, and which can be flexibly combined (see, for instance, http://www.gnu.org/manual/textutils/html_chapter/textutils_10.html).

There are some limitations to that philosophy: the success of Perl has shown the power of flexible text processing embedded in a complete, if ugly, programming language, and most people who do corpus-based NLP learn Perl at some point. But you can do a lot just by piping together command line tools in Unix, as we'll investigate here.

What you should do: first, you should look through and make sense of (using man pages, etc. as needed) the first 20 pages of Ken Church's Unix for Poets (attached). We put the whole tutorial online, but my personal opinion is that the first 20 pages is enough for the following reason: after that the tutorial moves into using awk (and later still C). Now, while choices among tools of overlapping functionality are a personal matter (though sometimes marked by zealotry), my personal opinion is that awk is an anachronism, and learning it is a waste of time. If you need more power than you can get from the the basic command line tools, then you're probably better off using Perl (or another programming language). Awk sits in an awkward place where it can do half of what you'd hope for from a programming language, but then you always find that it can't do the other half (or certainly not without pain). I will mention (and the assignment uses) a little more of sed than is described by Church (though again, if you end up trying to write loops in sed, you probably should be using Perl).

Second note: We'll assume that you are using the GNU version of these tools (GNU TextUtils, sed, grep, etc.). This is what you get by default on the Leland systems (i.e., providing you have /usr/pubsw/bin first in your PATH). Most of these commands are fairly common to all Unix variants, but in a couple of places (options to sort, regular expressions supported by sed, options to ptx) there can be a bit more variation in what is supported than optimal. You can find documentation for the GNU TextUtils at the URL below (or using the man pages, or info):

http://www.gnu.org/manual/textutils/textutils.html

Looking at corpora

We give here a quick illustration of using sed to pick out the text part of corpora marked up in a simple form of "stand off" SGML notation (where each line is either text or an SGML element). You need a real SGML/XML parser to properly parse SGML/XML.

The first is material from the Wall Street Journal from the files:

```
/afs/ir/data/linguistic-data/Newswire/wsj/1996/WS96010*
```

Look at one of them:

```
cd /afs/ir/data/linguistic-data/Newswire/wsj/1996
more WS960102
```

Note that the main text of articles is enclosed in <TEXT> . . . </TEXT> blocks in SGML/XML style. We'll just ignore the headlines.

You can almost get out the text out with the following sed command:

```
sed -n -e '/<TEXT>/,/<\/TEXT>/p' WS960102 | more
```

By default sed does something to lines of text, and then outputs them, so you can get the contents of a file with the spelling of color changed to (British/Australian) colour by doing:

```
sed -e 's/color/colour/g' wsj7_001 | more
```

The s command does a substitution, and the g modifier says to apply it multiple times if possible. In the previous sed command, we only want text printed when we want it printed, so we use -n to suppress the default printing of lines read in and then manually ask to print with the p command text that is in ranges between a <TEXT> and a </TEXT>. This is almost what we want, but it leaves those <TEXT> markers in. One of several ways to get just the body without the tags is to instead delete with the d command things that are outside TEXT elements:

```
sed -e '1,/<TEXT>/d;/<\/TEXT>/,/<TEXT>/d' wsj7_001 | more
```

This deletes from line 1 through the first <TEXT> and then deletes ranges between an end </TEXT> and a <TEXT> inclusive (and also the tail of the file after the last </TEXT> -- see the sed man pages. In general a sed command consists of a line number range followed by a single letter command. The semicolon separates multiple commands given to sed. The command is given via the -e flag (actually optional), with filenames then following. A line number range can either be a number (which matches just that line number), or a regular expression, which matches all lines matching the regular expression, or a range with a beginning and end, separated with a comma. The beginning and end can be line numbers or regular expressions, and depending on which is used this will pick out one or potentially many text ranges. If a line number range is omitted (as in the substitution example above), then the command is applied to all lines. The three commands we have seen are s (substitute), d (delete), and p (print). There are some others, which you can find out about in the man page, or from a helpful site such as:

```
http://www.dbnet.ece.ntua.gr/~george/sed/sedtut_1.html
```

However, these 3 commands are all you need for what we will do.