Assignment 3 - Filter scripts and Parallel

Objectives: Gain experience building a command line pipeline with your own filters. Make filters executable and use them from the notebook. Demonstrate understanding of data-parallel processing by replacing part of your pipeline with parallel processing.

Grading criteria: The tasks should all be completed, and questions should all be answered. Results should be correct, relative to the implementation choices made - not everyone will have the same results. The narrative surrounding code should inform the reader about the process, from sources of data through any decisions made about techniques and tools. The notebook itself should be reproducible; from start to finish, another person should be able to use the same code to obtain the same data and same results that you obtain, or at least similar data and similar results, should the data sources chosen vary over time.

Part 3 is **optional**. I encourage you to give it a try.

Give your notebook a clear name like "`assignment-03-mylastname`". Acknowledge any assistance you received, download the notebook as a PDF, and zip your .ipynb, PDF, and filter script together. Upload your zipfile to blackboard by the deadline.

Deadline: Tuesday, September 29, 7pm (before class begins)

Part 1 - English stop words filter

In Information Retrieval, "stop words" are words so common that their high frequencies in arbitrary texts can affect the quality of text search results. It is typical for software systems to remove stop words during indexing and search to improve quality (measured in precision and recall).

Write your own filter script using Python (see the example provided in class during week 3) or bash (see the SW Carpentry lessons or the Data Science at the Command Line book). This filter should remove at least 25 common English stop words from incoming text. You can find common stop word lists easily online, or use texts from Project Gutenberg to generate your own list.

Write the filter in a single, self-contained file. Give it a clear, descriptive name, and add comments that describe the expected input, transformation(s), and output. Make it executable, and within your notebook, describe the filter. Demonstrate its use in a word count pipeline, showing a pipeline without your filter first, then with your filter; the results of a word count should make the success of your filter obvious.

You may use existing command line tools (e.g. `grep -v`) in a pipeline to verify the correctness of your results, but to get full points, your filter must be invoked as a standalone script.

Repeat this demonstration with at least three different English book texts from Project Gutenberg. This will help verify that your filter works with different source data files.

Part 2 - Parallel processing

Download at least 10 separate English language texts from Project Gutenberg. Use a data-parallel processing tool like `parallel` (see the example provided in class during week 3) to build a pipeline to count the top words across all the texts. Remove stop words within this pipeline by using your filter from Part 1. If you use `parallel`, note that you can use it in with the `--eta` option inside a bash shell to see the jobs proceed across your available cores. You probably don't want to use this option within a notebook, though.

For more information about `parallel`, see its web site and the tutorial. The video included in this writeup may also be helpful (*note*: `parallel` is already installed on your VM; you don't need to download it or install it yourself; if you're on OSX, it's easy to install with homebrew).

Here is an example of using `parallel` to convert a sound file from one format to another. To convert `*.wav` to `*.mp3` using LAME running one process per CPU core run:

```
parallel lame {} -o {.}.mp3 ::: *.wav
```

You can find this example and many more on the parallel man page.

Summarize your results, and describe how you changed your pipeline from the previous assignment. How does stopword removal change the results? How does using a data-parallel technique change the way you think about building up a pipeline?

Part 3 - Image processing in parallel (Bonus: 25 points)

The graphicsmagick package is installed in your VM, providing simple command-line tools for working with image data. In addition, aliases for graphicsmagick that make it work like imagemagick are also installed, so you can use the convert and montage commands as described in the imagemagick docs.

Download at least a dozen images from Wikimedia Commons. You can find a helpful bash script for automating this here.

Experiment with the options for using `convert` to convert the images you downloaded to a smaller, thumbnail size. Then repeat the process, using `parallel` in a pipeline.

Experiment with the options for using `montage` and create a montage of the files you downloaded. Try creating it from the original files you downloaded, then repeating the process to create another montage from the thumbnail files you converted. Time the process both times using `time`. What are the main differences between the two approaches?

Note: you can display images within the bash notebook using the `display` command, e.g.:

```
display < myimage.jpg
```

Or if you prefer to use pipes:

```
cat myimage.jpg | display
```