A Word Frequency Pipeline

Due at 8:00pm on Monday, 12 October 2020

You are tasked with writing a pipeline using standard Linux programs to generate a word frequency list for a file.

There are 4 parts to this project.

Exercise 2.6 on page 38 of the textbook shows you that the following pipeline will produce the word[1] frequency in a file named "`document`":

```
tr –s '[:blank:]' '\n' <document | sort | uniq –c      (0)
```

You should consult the man pages for `tr`, `sort`, and `uniq` if you are unsure why this pipeline works.

In fact, as described in section 2.6.4 of the textbook, if we create a script file named `part0` that consists of the following single line:

```
tr –s '[:blank:]' '\n' | sort | uniq –c
```

we can achieve the same execution as pipeline (0) above by giving `bash` the following command:

```
sh part0 <document
```

1 Make the word list be case-insensitive

As discussed in Section 2.4.7.3 of the textbook, `uniq` is case sensitive, such that "this" and "This" would be considered different words by pipeline (0) above. Add another invocation of `tr` to the pipeline in `part0` to make the word count case-insensitive.

Your modified pipeline should be in a script file named `part1`.

2 Change the output order

The pipeline in `part0` outputs the results according to the sort order of the words. It is more likely that you wish to see the words by frequency, from high to low. Add another invocation of `sort` to the pipeline in `part0` to present the output by frequency, from high to low.

Your modified pipeline should be in a script file named `part2`.

[1] A word is a sequence of non-whitespace characters, separated from other words by one or more blanks, tabs, or end of line characters.

3 Punctuation and white space delimit words

Our definition of a word means that punctuation is included in a word – e.g., the last word in "I must go to the store." is "store.". Add another invocation of `tr` to the pipeline in `part0` to also use punctuation to delimit words.

Your modified pipeline should be in a script file named `part3`.

4 All together now

Finally, we want case-insensitive, punctuation and white space delimited words, with the final output sorted by frequency, from highest to lowest.

Your modified pipeline should be in a script file named `part4`.

5 Starting files

In Canvas, in Files/Projects, you will find a gzipped tar archive named `P1start.tgz`; this file contains the following files:

- `input` – a file that contains a lot of words, both with and without upper case letters and punctuation.
- `part0` – a file containing pipeline 0 above; **note that it does not contain the `<document` term shown in pipeline (0). Your `part1`, `part2`, part3, and `part4` files should NOT contain the `<document` term, either**.
- `part0.out` – this is how the output should look if you execute pipeline (0) using the file named `input`.
- `part1.out` – this is how your output should look if you have correctly modified pipeline (0) for `part1` and applied to the file `input`.
- `part2.out` – this is how your output should look if you have correctly modified pipeline (0) for `part2` and applied to the file `input`.
- `part3.out` – this is how your output should look if you have correctly modified pipeline (0) for `part3` and applied to the file `input`.
- `part4.out` – this is how your output should look if you have correctly modified pipeline (0) for `part4` and applied to the file `input`.
- `tscript` – a bash script file that performs a number of tests of your `part?` files using `input` and `part*.out`; invoking

  ```
  ./tscript
  ```

  will execute all of your pipelines, comparing their outputs against the corresponding output files.

6 Checking that your pipelines work correctly

As stated earlier, we can execute the pipeline in `part0` with the following command[2]:

```
sh part0 <input
```

This will cause the pipeline to be executed, taking its input from the file named `input`.

To check if the output is correct, you can execute the following pipeline:

```
sh part0 <input | diff - part0.out
```

This pipeline tells `diff` to compare standard input (denoted by `'-'`) to `part0.out`; if the correct output is generated by `part0`, nothing should be printed, and you will see the next prompt from `bash`.

You can use the same technique to check the output from `part[1-4]` against `part[1-4].out`.

The `sort` utility on Linux and Unix-like systems is dependent upon the current locale defined for your system[3]. The shell script I have provided, `tscript`, which can be used to execute the above tests, makes sure that you are using the correct locale; if one or more arguments are specified, `tscript` will only perform those specific tests; if no arguments are specified, all of the tests are performed. Each test is an invocation of your `part?` script and a comparison of your output against the correct output.

The tests that `tscript` performs are:

0	Tests that the `part0` script provided to you works correctly.
1	Tests that your `part1` script works correctly.
2	Tests that your `part2` script works correctly.
3	Tests that your `part3` script works correctly.
4	Tests that your `part4` script works correctly.

For each test, `tscript` prints out "Testing that part<digit> works correctly", where <digit> is one of {0, 1, 2, 3, 4}; it then prints a line consisting of "====="; it then executes the corresponding script applied to input; finally, it prints "===== Stopping test <digit>".

If there is any output between the "=====" line and the Stopping line, the test has failed.

---

[2] While "`sh part0 <input`" will cause `part0` to be executed by the shell, we can make `part0` executable using the following command:

```
chmod +x part0
```

after doing so, we can execute the pipeline contained in `part0` with the following command:

```
./part0 <input
```

[3] See https://man7.org/linux/man-pages/man1/locale.1.html if you want more information about locales.

7 Submission[4]

You will submit your solutions electronically by uploading a gzipped tar archive[5] via Canvas.

Your TGZ archive should be named `<duckid>-project1.tgz`, where `<duckid>` is the bit of your UO email address before the @. The archive should contain your files `part1`, `part2`, `part3`, and `part4`; it should also contain a file named **`report.txt`**; this file should contain your name, duckid, the names of any classmates who helped you and what type of help each provided, and the current state of your submission.

8 A professional way to create your TGZ archives

As described in section 2.7.3.1 of the textbook, you can create your TGZ archive using a command of the form:

```
tar –zcvf duckid-project1.tgz report.txt part1 part2 part3 part4
```

Of course `'duckid'` above has to be replaced by **your** DuckID. This a lot to type, and future assignments might require that you put 10-15 files into an archive. By creating a manifest file, one can simplify the creation of the TGZ file, and avoid including extraneous files.

A **manifest**, customs **manifest,** or cargo document is a document listing the cargo, passengers, and crew of a ship, aircraft, or vehicle, for the use of customs and other officials.[6] In software terms, a manifest is a list of files that should be included in an archive.

We recommend that you put all of your files for each project in a separate directory in your Arch Linux image; `project1` would seem to be a good name for this project, but you may name it anything you wish. You should probably put the contents of `P1start.tgz` from Canvas in this directory, and should create your `part1`, `part2`, `part3`, `part4`, and `report.txt` files there, as well.

Since the submitted archive has to contain files named `part1`, `part2`, `part3`, `part4`, and `report.txt`, you should create a file in this directory named `manifest` that contains the following lines:

```
report.txt
part1
part2
part3
part4
```

Armed with the `manifest`, `report.txt`, `part1`, `part2`, `part3`, and `part4` files, creation of the TGZ file becomes

[4] ***A 25% penalty will be assessed if you do not follow these submission instructions.***

[5] *See section 2.7.3.1 on page 42 of the textbook for instructions on how to create a gzipped tar archive.*

[6] This definition is taken from Wikipedia.

```
tar –zcvf duckid-project1.tgz $(cat manifest)
```

The `$(cat manifest)` expression causes the shell to invoke `cat` on the file named `manifest`; instead of displaying the contents on the standard output, the output from `cat` is captured by the shell, and it merges all of the lines into a sequence of blank separated words; this merged list then provides the additional arguments given to `tar`.

Grading Rubric

Your submission will be marked on a 50 point scale. Substantial emphasis is placed upon **WORKING** submissions, and you will note that a large fraction of the points are reserved for this aspect. It is to your advantage to ensure that whatever you submit compiles, links, and runs correctly. The information returned to you will indicate the number of points awarded for the submission.

You must be sure that your code works correctly on the virtual machine under VirtualBox, regardless of which platform you use for development and testing. Leave enough time in your development to fully test on the virtual machine before submission.

The marking scheme is as follows:

Points	Description
5	Your report – honestly describes the state of your submission
4	Your pipeline in `part1` works correctly against the provided `input` file.
4	Your pipeline in `part2` works correctly against the provided `input` file.
4	Your pipeline in `part3` works correctly against the provided `input` file.
4	Your pipeline in `part4` works correctly against the provided `input` file.
7	Your pipeline in `part1` works correctly against an unseen file of words.
7	Your pipeline in `part2` works correctly against an unseen file of words.
7	Your pipeline in `part3` works correctly against an unseen file of words.
8	Your pipeline in `part4` works correctly against an unseen file of words.

Note that:

- Your report needs to be honest. Stating that everything works and then finding that it doesn't is offensive. The 5 points associated with the report are probably the easiest 5 points you will ever earn as long as you are honest.