wordgrep utility's specifications

Benoît Dunand-Laisin

October 13, 2012

1 Introduction

2 Specifications

Write a utility with the following syntax:
wordgrep dictfile [inputfile ...]
wordgrep parses a dictionary file as follows:

- a line with a single word defines a word to be found in the input stream
- empty lines and line starting with # are ignored

wordgrep then filters (to the standard output) its input stream, the concatenation of all input files specified on the command line or standard input if none is given. wordgrep outputs any line from the input streams that contains one of the words from the dictfile. Matching is performed on a full word basis. Words are delimited by white space.

Implementation should be simple and efficient, dictionary size should be limited only by memory,

An arbitrary limit on line length is OK for the dictionary, but should be unnecessary on input stream.

3 Existing alternatives

Why not try this: grep -wf dictfile [inputfile ...]

4 Test cases

4.1 Without any parameter

Command line	wordgrep
dictfile content	n/a
file(s) content	n/a
Error output	Without a dictionnary file, there is nothing to do
Standard output	No output

4.2 With an empty dictfile and one file

Command line	wordgrep dictfileEmpty fileA
dictfile content	
file(s) content	Never mind
Error output	Dictionnary file is empty, there is nothing to do
Standard output	No output

4.3 With a dictfile and one file

Command line	wordgrep dictfile fileA
dictfile content	Toto TITI
file(s) content	I like Toto's joke. However, some on them are not funny. Toto + Toto = Toto's head Funny isn't it?
Error output	No error
Standard output	Toto + Toto = Toto's head

Note: First line doesn't appear in output because the word is Toto's and not Toto.

4.4 With a dictfile and more than one file

Command line	wordgrep dictfile fileB fileC
dictfile content	Toto TITI
file(s) content	**fileB** I like Toto's joke. However, some on them are not funny. **fileC** Toto + Toto = Toto's head Funny isn't it?
Error output	No error
Standard output	Toto + Toto = Toto's head

4.5 With a dictfile and no file

Command line	wordgrep dictfile
dictfile content	Toto TITI
file(s) content	n/a
Error output	No error
Standard output	????

4.6 With a missing dictfile and one file

Command line	wordgrep dictfile2 fileA
dictfile content	n/a
file(s) content	Never mind
Error output	The given dictfile is missing.
Standard output	No output

4.7 With a dictfile and one missing file

Command line	wordgrep dictfile fileAA
dictfile content	Never mind
file(s) content	Never mind
Error output	fileAA is missing
Standard output	No output

4.8 With a dictfile, one missing file and at least one existing file

Command line	wordgrep dictfile fileAA fileB
dictfile content	Toto TITI
file(s) content	**fileAA missing** **fileB** Toto + Toto = Toto's head Funny isn't it?
Error output	fileAA is missing
Standard output	Toto + Toto = Toto's head

4.9 Parsing validation

Command line	wordgrep dictfileB fileD
dictfile content	Toto TITI some World is mine # Funny averybigwordsosolongthatitswillbeignored don't
file(s) content	I like Toto's joke. However, some on them are not funny. Toto + Toto = Toto's head Funny isn't it? World is mine !!! I don't think so.
Error output	The following dictfile entries are ignored: World is mine averybigwordsosolongthatitswillbeignored
Standard output	However, some on them are not funny. Toto + Toto = Toto's head I don't think so.

4.10 Big data testing

To complete...

5 Implementation

5.1 function parseDictionary

This function is used to parse and load the dictionary content in memory.

5.1.1 Words storage

Words are stored in memory, concatened, and separed by a pipe. That way permit to use the strstr function on the words list.
Alternative is to manage an array of char*. However the grep operation would be more time consumming (but it permits to use the full memory capacity).
Memory is allocated first for a "big" size. and then reallocated each time it's necessary for a defined size (that doesn't equal to the size of the word to store).
Alternative is to reallocate the memory for each word, but this lead to poor performances and a worst memory fragmentation.

5.1.2 Dictionnary file reading and filtering

Dictionnary file is read line by line. The variable used to store the line is size limited abitrary. This will be a known limitation.
Lines are left and right trimed before tested.
Empty lines are ignored.
Lines beginning with a "#" are ignored.
Lines containing a white space are ignored with a warning message.

5.1.3 Dictionnary context

The context of the dictionnary contains:

- The number of words stored
- The size of the biggest word

5.2 function openInput

5.3 function readInput

This function extracts data from input flux. The data is read character by character (getc function) The character is stored in the current word. When a white space is read then If the current line doesn't contain a dictionnary word, then call the grepWord function Clear the current word and begin a new one When a new line is read: If the current line contains a dictionnary word, then go back to the beginning of the line (fsetpos function) et write the line (fwrite) Store the position in the file (fgetpos function). When input flux is ended then exit the function

5.4 function grepWord

This function handle a single full word and check its existence in the dictionnary. The word is stored between two pipes and then, the function strstr is used. Returns "True" if word is found in the dictionnary, else "False".

6 Improvement

7 Packaging