Using GNU/Linux tools to perform lexical searches and statistics (work in progress)

Christophe Pallier

January 26, 2007

Contents

Abstract

This document is meant to demonstrate the use of some GNU tools[1]/Linux[2] command tools to manipulate text files, with special emphasis on regular expressions and the AWK scripting language.

We rely on the Lexique database[3] for many examples. Creating this database and putting it on-line has been made possible only by the free availability of the GNU tools and free software like Perl[4].

[1]http://www.gnu.org
[2]http://www.linux.org
[3]http://www.lexique.org
[4]http://www.perl.org/

Chapter 1

Using regular expressions to query the Lexique database

Lexique is a lexical database providing information for more than 100000 French words, such as, orthographic and phonetic forms, frequency of occurrence in several corpora, gender, number, grammatical category, . . . The database is described in [New2004] and is freely accessible at `http://www.lexique.org`. It can be queried on-line or downloaded for further manipulations.

1.1 Simple on-line searches

⋄ Open Lexique's main page[1] in a new window (It is highly recommended to view the Lexique web page and this documentation in two browser windows placed side-by-side) and browse the site.

⋄ In the left-hand menu, select "Use Lexique online[2]" (Or "Interroger Open Lexique[3]" on the French version[4] of the site).

In the "Word request" form, press "search" (*Recherche par mots*). Try other French words. Note that other lexical databases are available beyond 'Lexique 3'.

1.2 More complex on-line searches

For more complex queries (e.g., finding words that are 7 letters long, that start with an 'a' letter, which are nouns,...) you need to use the "Features request" (*Recherche par propriétés*) form.

⋄ In the "Features request" form, check "Lexique 3" and click "Search" (*Recherche*)

- Search for words such that "lexique3.ortho = arb*"
- Search for words such that "ortho=3 and nblettres=6"
- Search for words that are nouns and are also 7 letters long.
- Within the previous results, select only words for which "freqlivres > 100".
- Search words that are bisyllabic and have 6 letters.

Remark: You may get help from the Documentation page[5] (Lexique 3 Manual, section 6.1).

[1] http://www.lexique.org/english
[2] http://www.lexique.org/moteur/index.eng.php
[3] http://www.lexique.org/moteur
[4] http://www.lexique.org
[5] http://www.lexique.org/documentation.php

1.3 Introduction to regular expressions

Fine, but how to search for words that start with a vowel, contain a given string of consonnants...? To find words that have a given pattern (*motif*), you need to use ”regular expressions” (*expressions rationnelles*). You can think of a regular expression as a kind of ”filter” (like the wildcard pattern ’arb*’ used for the simple search, but much more powerful as you shall see).

Let us get started with a few fundamental notions about regular expressions (regex):

- A regex is a ’formula’ containing alphanumeric characters (letters and digits) and, optionnaly, symbols such as ’$’, ’+’, ’*’, ’?’, ’(’, ’)’, ...
- if the regex is a letter ’a’ or a string of letters ’abc’, it maches any word containing this letter or that string. Thus the pattern ’abc’ matches the word ’abcès’. Note that the pattern may occur anywhere within the word.
- The ’ˆ’ sign at the beginning of a regex matches the start of a word; the pattern ’ˆer’ matches words starting with the letters ’er’.
- The ’$’ sign matches an end. ’iop$’ matches words ending in ’iop’.
- A dot matches any character; the regex ’a..d’ matches ’abcd’, ’a00d’, ’artd’ but not ’afd’, ’afdrd’, ’adde’,...

Armed with this knowledge, you can now experiment:

◇ On the page ”Recherche par propriétés”, Check ”Expression régulières”.[6]

- Search for ”lexique3.ortho = arb”. Notice the difference with the simple search.
- Search for ”lexique3.ortho = ˆarb”.
- Search for ”lexique3.ortho = art$”.
- Search for ”lexique3.ortho = a.b.a”
- Search for words that have a CVCV phonetic frame.
- Search for words that start with the letter ’j’ and end with letter ’t’
- Search for words that start with the letter ’j’, end with the letter ’t’ *and* are 4 letters long.

More about regex:

Character classes
- [abc] matches any word containing a, b or c.
- [ˆabc] matches any word that does not contain a, b, or c
- Range of matches: [a-z] matches any letter between a and z, [0-6], any digit between 0 and 6, [a-zA-Z0-9] any alphanumeric character (but what about accented characters?)

Alternation (regex1 — regex2) matches word thats matches either regex1 or regex 2 (or both).

Repetitions
- a pattern followed by a ’+’ sign means that the pattern can be repeated several times. For example ’a+b’ matches ’ab’, ’aab’, ’aaab’, ... but not ’acb’.

  It is possible to use parentheses to delimit the regex to which the ’+’ operator applies. ’ba+t’ matches ’bat’, ’baat’, ’baaat’... while ’(ba)+t’ matches ’bat’, ’babat’, ’bababat’,...
- a pattern followed by a ’*’ sign means that the pattern can appear zero or more times (in other words it is optional). ’ab*a’ matches ’aa’, ’aba’, ’abba’, ...
- a pattern followed by a ’?’ sign means that the pattern can appear zero or one time.
- a pattern followed by $\{n, m\}$ means that the pattern can appear between n and m times.
- the vertical bar between two regex will match any string mathing one or the other regex (this is called an ”alternation” pattern). ’(ab|cd)’ will match ab, cd, but not ad, bc,...

[6]The corresponding English page does not work correctly at the time of this writing.

◇ Back to the Lexique interface:

- Search for words that contain 'aci' or 'acci'
- Search for words that start with a vowel (ortho)
- Search for words that contain three consonnants in a row (phono).
- Search for words that end with 3 consonnants.
- Search for words such that cvcv = ^CVC?VC$
- Search for words such that cvcv = ^CVC+VC$
- Search for words such that cvcv = ^CVC*VC$
- Search for words such that cvcv = ^(CV)+$

Table: Summary of operators for regular expressions

Symbol	Meaning	Example	Matching words
^	beginning of string	^a	arbre, arbuste
$	end of string	e$	tente, mare
.	any character	^a..e$	arme, acte
[xyz]	characters x, y or z	a[bc]	raccroché, abruti
[x-z]	La tranche de caractères de x à z	a[l-n]	amener, alourdi, anneau
[^xyz]	any character exceptt xyz	[^aeiyouéèïê]	Toutes les consonnes
*	maybe present zero or more times	m*	emmener, amender, entasser
+	maybe present one or more times	m+	emmener, amender
?	maybe present zero or one time at most	m?	amender, entasser
|	ou	(buv|parl)ant	buvant, parlant

There is much more to the topic of regular expressions. This tutorial[7] can be useful.

For a comprehensive overview of regex, see the book *Mastering Regular Expressions* [MRG].

[7] http://www.regular-expressions.info/tutorialcnt.html

Chapter 2

Grep and sed: Searching and Replacing text, from the command line

2.1 grep

Regular expressions can be useful in many situations, for example to validate data input on a form. Now we are going to play a little bit with the `grep` command line tool that searches files for patterns expressed as regular expression. The command *grep pattern file* reads *file* line by line and print each line that the pattern matches [GrepMan].

◇ Exercice.

- Create a text file ’aga.txt’ containing the following lines:

  ```
  alpha5
  kill 65 hours
  8 Juillet 1945
  45.6 F
  -4 C
  longueur: 45mm
  -798
  The answer is 42
  distance: 67km
  8pc
  babar@chezmoi.com
  jean.aymard@zut.com
  @@@@@@@@@@@@
  jim%lo@io$p
  "#45"
  toto Toto toTo
  titi
  toti
  ```

  (Hint: you can use the command ’cat > aga.txt’, cut & paste the preceding passage and press Ctrl-D)
  Now, using ’grep pattern aga.txt’
- find lines that contain an integer number
- find lines that contain a floating number
- find lines that contains a length measure
- find lines that contain an email address
- Write a bash script that reads a file line by line, and for each tells whether the line is an integer, a relative integer, a decimal number.
- Write a bash script that detects lines which contain an email address

- Computer connected to the Internet have IP numbers which consist of 4 numbers between 0 and 255 separated by dots. Can you write a regexp to detect if a string is a valid IP number?
- Find lines that contain a repeated two-letter pattern (e.g. titi). For this, you will need to learn about back-references in regular expressions (do a Web search)

2.2 sed

(Note: This passage was adapted from `http://www.faqs.org/docs/abs/HTML/sedawk.html`)

Sed is a *non-interactive* text editor [SedMan]. It performs certain operations on specified lines of the input, one line at a time, then outputs the result to the standard output.

Among others operations, sed allows to extract or delete certain lines, or to search and replace certain portions of text.

Operator	Effect
[address-range]p	Print [specified address range]
[address-range]d	Delete [specified address range]
[address-range]s/pattern1/pattern2/	Substitute pattern2 for every instance of pattern1 in a line, over address-range

The (optional) address-range is specified either by line number or by a regexp pattern, enclosed between '/', to match. For example, 3d signals sed to delete line 3 of the input, and /windows/d tells sed that you want every line of the input containing a match to "windows" deleted.

Examples:

8d	Delete 8th line of input.
/GUI/d	Delete all lines containing "GUI".
/^$/d	Delete all blank lines.
1,/^$/d	Delete from beginning of input up to, and including first blank line.
/Jones/p	Print only lines containing "Jones" (with -n option).
s/Windows/Linux/g	Substitute "Linux" for every instance of "Windows" found in each input line.
s/ *$//	Delete all spaces at the end of every line.
s/00*/0/g	Compress all consecutive sequences of zeroes into a single zero.

Many more examples are provided at `http://www.student.northpark.edu/pemente/sed/sed1line.txt`.

◇ A few exercices with sed:

- The files/alice30.txt and files/lglass19.txt come from the Gutenberg project.
  As all files from the project, they start with a long preamble that you may need to remove. How would you do that with sed?
- Download the web page at `http://www.w3.org/People/Berners-Lee/FAQ.html` and try to see if you can use sed to remove the HTML markup elements (enclosed between '¡' and '¿' characters.
- Write a bash script using sed to convert an ASCII file into Morse [Morse].

2.3 Note about text editors

Many text editors have search and replace functions that allow the use of regular expressions.

For example, in the "One True Editor" Emacs:

```
Alt-x isearch-forward-regexp # interactive search for a pattern (Ctrl-Alt-S)
```

```
Alt-x query-replace-regexp # search and replace
Alt-x replace-regexp # the same, without querying (dangerous!)
```

The documentation for regexp in Emacs is available at [egexps]section 20.5 of the Emacs manualhttp://www.gnu.org/software/emacs/ma
(also available in the local info manual available with 'Ctrl-h i')

◇ Use your favorite text editor (or Emacs if you do not have one) to:

- Detect all words starting with a uppercase letter.
- Detect strings of spaces and replace them with a single space
- convert text into Morse code (Alt-x morse-region ;-)

Chapter 3

Querying Lexique off-line, with awk

For any serious use of the Lexique database, rather than querying it on-line, you should download it on your local hard drive and perform the manipulations with installed software. This will afford you much more flexibility.

◇ Get the file Lexique3.utf8.txt[1].

(Note: This file was obtained by recoding the file Lexique3.txt included in the Lexique3 distribution (Zip from `http://www.lexique.org/telecharger.php`) from iso-8859-1 into utf8 using the command `recode`.)

- What size is the file, how many lines does it have? (see chapter 5 if you do not know how to do that.)
- View the file with the command 'less -S' (type 'q' to exit the viewer).
- Open the file with the emacs text editor.
  To view the file better:

  ```
  Alt-x set-variable RET truncate-lines RET T
  Alt-x set-variable RET tab-width RET 20
  ```

  Remark: Emacs is powerful but definitely not easy to apprehend; if you want to learn it, you will find plenty of documentation on the web, including this introduction to emacs[2]. If you are allergic to it, use nano, gedit or nedit, or any other among the many text file editors available under Linux.
- The file is very large and not easy to handle. It will be more convenient to work with a reduced version. In a terminal, run the following command:

  ```
  cut -f 1-3,10 Lexique3.utf8.txt >base.txt
  less -S base.txt
  ```

  What is the size of the new file?

You could use 'grep' to search 'base.txt', but you will quickly realise that is it not a very good idea. That is because 'base.txt' is organised in a tabular format. If you try to use 'grep' to find words that contain the vowel 'a' in their phonetic representation (that is, in the second field), you will match words that contain 'a' in the orthographic representation (first field).

3.0.1 Introduction to AWK

The right tool to manipulate data organized by rows and columns is AWK.[3]

[1] files/Lexique3.utf8.txt
[2] http://www.pallier.org//ressources/intro_emacs/intro_emacs.pdf
[3] Actually, AWK is also often the right tool when the input is not organized into columns.

The power of AWK stems from its ability to manipulate regular expressions and to store information in associative arrays.

For a complete coverage of Gawk, the GNU project's implementation of AWK, see [GawkMan]. If the Gawk documentation has been installed on your system, you should be able to access it from the command line with `info gawk`.

The basic function of AWK is to read the input line by line and, and for each line, save in it variable $0, split it into items associated to variables $1, $2, $3,... and execute a mini-program that is a series of:

```
condition {action}
condition {action}
...
```

If the condition is true, then the action is executed.

If no action is specified but the condition is true, the line is printed (this is equivalent to the action `print $0`).

The items in various columns are refered to as $1, $2, $3... Here are example of conditions:

```
$1 ~ /^arb/    # the first item matches the regex between slashes
$3 == 10       # the third item is equal to 10
($1 == "alabama") && ($3 >10) # example of a complex condition
```

◇ Run the following

```
awk -F'\t' '/^arb/' base.txt
```

The first argument tells awk that tabulations are used as column separators. It is followed by the awk program, between single quotes. Alternatively the script could be inside a file, e.g. script.awk, and `awk -f script.awk` would execute it (see [4]unning-gawk]How to Run awk Programs http://www.gnu.org/software/gawk/manual/gawk.html).

diamond Execute the following commands:

```
awk -
awk -F'\t' 'length($1)>14' base.txt
```

Two specials conditions exists: 'BEGIN' and 'END', which are true just before the program starts (ends) reading the input.

◇ Guess what the following program does:

```
BEGIN{ nl = 0}
{nl = nl + 1}
END{ print nl}
```

A variable 'NR' (number of records) keeps track of the number of lines processed.[5] For each line, total number of items (columns) detected in one line is save in a variable named 'NF' (number of fields).

◇ Guess what the following program does:

```
END{print NR}
```

[4]R

[5]AWK can actually break input into blocks that do not correspond to lines. This is useful for processing files with records spanning multiple lines.

And the following:

```
{ w = w + NF }
END{print w}
```

See http://www.pallier.org//ressources/awk_for_lex/awk_for_lex.html

◇ Exercices: Some rely on several commands listed in chapter 5

1. Find the 20 most frequent words.
2. Find the longest word(s). (Tip: use the function length in awk and pipe the output to command 'sort' and 'head')
3. Find the words that contains the most 'a'
4. Plot the distribution of word lengths.
5. Create a dictionnary where words are sorted by rhymes (for poets)
6. Find the orthographic palimdromes (words whose spelling can be reversed, such as, e.g., *radar*)
7. Find the phonetic palindromes
8. Find words which are composed of the same syllable repeated twice (e.g. *tchatcha*)
9. Find words whose orthography is the repetition of the same pattern (e.g. *couscous*)
10. Find words whose are composed of repeated letters
11. Find words whose inverted spelling is also a word.
12. Find all the possible groups of two or more vowels (in the phonetic transcription).
13. Find anagrams (e.g. *gymnaste – syntagme*)

Write an awk script that recodes text into [Morse], using the file Morse.code.txt.

3.0.2 Associative arrays

See [[6]rrays]Awk Arrayshttp://www.gnu.org/software/gawk/manual/gawk.html

[6]A

Chapter 4

Activities

4.1 Correlation between two frequencies estimors

Lexique3 provides two estimates of lexical frequency (9_freqfilms and 10_freqlivres) The first comes from a corpus of subtitles, the second from a corpus of books.

- Plot the relationship between the two (you can use graph (from plotutils), gnuplot, R, or any other means)
- Find the words for which the discrepancy is maximal.

4.2 Compute frequencies of letters and words in a text file

Given a text file, compute the frequencies (that is, number of occurrences) of words, and of letters.

http://www.gnu.org/software/gawk/manual/

4.3 Clean a corpus and compute frequency

◇ The files files/alice30.txt and files/lglass19.txt come form are from the Gutenberg project.

1. Clean them, removing the preambule (text until '*END*'). This can be done either by hand in a text editor, or with awk or sed (see the footnote for a solution[1])
2. Compute the lexical frequencies
3. Create a file providing, for each word, its orthography, it reversed orthography,

4.4 Checking Estoup-Zipf's law

J.-B. Estoup [Estoup1916] computed the frequencies of words (f) in a text, rank the words by frequencies, and plotted the ranks (r on the x axis and the frequency (f) on the y axis (on a log-log diagram). He noticed an interesting pattern.

This was later known as Zipf-law (Zipf proposed that (f $\times$ r = K).

[1] `sed "0,/\*END\*/d" alice30.txt`

◇ The files files/alice30.txt and files/lglass19.txt come form are from the Gutenberg project.

1. Create a Zipf plot

◇ Download other texts from the Gutenberg archive and create Zipf plots.

Mandelbrot noticed that if you take purely random text

For those who want to go further, I highly recommend the book *Word Frequency Distributions* [WFD]. The author, Harald Baayen, has made some software available at http://www.mpi.nl/world/persons/private/baayen/software.html:

1. Download lexstat-2.0.tar.gz[2]
2. Untar it

   ```
   tar xzf lexstat-2.0.tar.gz
   ```

3. Compile it (detailed instructions are in INSTALL).

   ```
   ./configure --prefix=~
   make
   make install
   ```

[2]http://www.mpi.nl/world/persons/private/baayen/doc/lexstats-2.0.tar.gz

Chapter 5

Useful tools

Most of these tools are documented in ’info coreutils’

du estimate file space usage

`du -h .` size of current directory

cat concatenate files and print on the standard output

`cat *.txt >everything`

less file viewer. Press ’?’ for help, ’q’ to quit.

wc print the number of newlines, words, and bytes in files

sort sort lines of text files

`du | sort -rk | less`

uniq report or omit repeated lines

`sort words.txt | uniq -c`

head output the first part of files

tail output the last part of files

cut extract columns from the input

`cut -f 10-12` extracts columns 10, 11 and 12 from a tab-delimited file.

grep print lines matching a pattern. See section 2.1

tr translate or delete characters

```
tr A-Z a-z file   # convert uppercase to lower case
tr '[:lower:]' '[:upper:]' # same thing (what about accented letters?)
```

sed perform basic text transformations on an input stream.

sed stream editor. See section 2.2

graph create graphics — see info plotutils

```
 awk 'BEGIN{for (i=1;i<10;i++) {print rand(),rand()}}' | sed s/,/./g | graph -T X  -C
```

unzip extract compressed files in a ZIP archive

`unzip -l archive.zip` lists the content of a zip file

recode converts files between various character sets

`recode utf8..latin1` convert from utf8 to latin1 charset

5.0.1 Perl or Python

When you reach the limits of awk (that is, when your program start to be longer than 30 lines), it may be time to learn real programming languages such as Perl[1] or Python[2].

[1]http://www.perl.org
[2]http://www.python.org

Chapter 6

Appendix

6.1 Installing UNIX || STAT

The ||STAT tools[1] is a suite of 30 data manipulation and analysis programs[2] which can be very handy. The data analysis tools comprise:

anova	multi-factor analysis of variance, plots
calc	interactive algebraic modeling calculator
contab	contingency tables and chi-square
desc	descriptions, histograms, frequency tables
dprime	signal detection d' and beta calculations
features	tabulate features of items
oneway	one-way anova/t-test, error-bar plots
pair	paired data statistics, regression, plots
rankind	independent conditions rank order analysis
rankrel	related conditions rank order analysis
regress	multiple linear regression and correlation
stats	simple summary statistics
ts	time series analysis, plots

Is is forbidden to redistribute the code. You must get it from the author following the instructions on the web page.

Once you have the stat.tar.Z package, you can install it under Linux in the following way:

```
mkdir -p ~/bin
tar xzf stat.tar.Z
cd stats
cd src
make -f makefile
cd ../bin
cp bin/* ~/bin
```

[1]http://www.acm.org/ perlman/stat/
[2]http://www.acm.org/ perlman/stat/doc/intro.htm

Chapter 7

References

Bibliography

[GrepMan] *Grep, print lines matching a pattern* http://www.gnu.org/software/grep/doc/grep.html 2.1

[SedAwk] Dougherty, D. and Robbins, A. *Sed and Awk*, 2nd Edition, O'Reilly and Associates, 1997 http://www.oreilly.com/catalog/sed2/

[SedMan] Pizzini, K. *Sed, a stream editor* http://www.gnu.org/software/sed/manual/sed.html 2.2

[GawkMan] *Gawk: Effective AWK Programming* http://www.gnu.org/software/gawk/manual/ 3.0.1

[AwkCheat] Cheat sheet for AWK at http://www.student.northpark.edu/pemente/awk/awkuse.txt

[MRG] Friedl, J.E.F. *Mastering Regular Expressions*, 3rd Edition, O'Reilly and Associates , 2006. http://www.regex.info/ 1.3

[New2004] New, B., Pallier, C., Brysbaert, M. & Ferrand, L. "Lexique 2 : A New French Lexical Database" *Behavior Research Methods, Instruments, & Computers*, 2004, 36, 516-52. http://www.pallier.org/papers/NewPallier.lexique2.bmric2004.pdf 1

[Morse] http://homepages.cwi.nl/~dik/english/codes/morse.html files/Morse.code.txt 2.2, 3.0.1

[WFD] Baayen, H. (2001) *Word Frequency Distributions*. Dordrecht: Kluwer Academic Publishers. 4.4

[Estoup1916] Estoup, J.-B. (1916) *Les Gammes Sténographiques* (Institut Sténographique de France, Paris. 4.4

[Zipf1935] Zipf, G. K. (1935) *Psycho-Biology of Languages* (Houghton-Mifflin)

[Pet1973] Petruszewycz, M. (1973) L'histoire de la loi d'Estoup-Zipf : documents. *Mathématiques et Sciences Humaines*, 44, p. 41-56 url:http://www.numdam.org/numdam-bin/fitem?id=MSH_1973__44__41_0