Introduction to `Bash`

`grep` and `awk`

Mauricio Sevilla

email= j.sevillam@uniandes.edu.co

02.01.19

We have been exploring several commands that allow us to use the OS via the `CLI`.

But some questions arise, basically search things seems a difficult task, but there are some very useful commands that help us to do these kind of things.

`grep`

The procipal use of `grep` is to search a specific *text* (string) on a particular set of files.

This is one of the most used programs on the shell,

```
grep "String to Search" group_of_files
```

And we'll get on the standard output the lines of the files which contain the specific string we are searching for.

Try out the first part of the Homework.

In [1]:
```
cat file1.txt
```

```
THIS LINE IS AN UPPER CASE LINE IN THIS FILE.
ESTA ES UNA LÍNEA EN MAYÚSCULAS.

this line is the a lower case line in this file.
esta es una línea en minúsculas.

This Line Has All Its First Character Of The Word With Upper Case.
Esta Línea Tiene La Primera Letra De Cada Palabra En Mayúsulas.

And this is the last line.
Esta es la última línea.
```

In [2]:
```
grep 'es' file1.txt
```

```
esta es una línea en minúsculas.
Esta es la última línea.
```

But, there are times, where we are not interested if we have or not, uppercase (lowercase) characters, so we can use the insensitive case

In [3]:
```
grep -i 'es' file1.txt
```

```
ESTA ES UNA LÍNEA EN MAYÚSCULAS.
esta es una línea en minúsculas.
Esta Línea Tiene La Primera Letra De Cada Palabra En Mayúsulas.
Esta es la última línea.
```

There is no difference if we use ' or "

In [4]:
```
grep -i "es" file1.txt
```

```
ESTA ES UNA LÍNEA EN MAYÚSCULAS.
esta es una línea en minúsculas.
Esta Línea Tiene La Primera Letra De Cada Palabra En Mayúsulas.
Esta es la última línea.
```

Some times we want to have exactly the word we are looking for, not the string to be part of a different word,

In [5]:
```
grep -iw "es" file1.txt
```

```
ESTA ES UNA LÍNEA EN MAYÚSCULAS.
esta es una línea en minúsculas.
Esta es la última línea.
```

Let see the manual!

In [6]:
```
man grep
```

```
GREP(1)                   BSD General Commands Manual                  GREP(1)

NAME
     grep, egrep, fgrep, zgrep, zegrep, zfgrep -- file pattern searcher

SYNOPSIS
     grep [-abcdDEFGHhIiJLlmnOopqRSsUVvwxZ] [-A num] [-B num] [-C[num]]
          [-e pattern] [-f file] [--binary-files=value] [--color[=when]]
          [--colour[=when]] [--context[=num]] [--label] [--line-buffered]
          [--null] [pattern] [file ...]

DESCRIPTION
     The grep utility searches any given input files, selecting lines that
     match one or more patterns.  By default, a pattern matches an input line
     if the regular expression (RE) in the pattern matches the input line
     without its trailing newline.  An empty expression matches every line.
     Each input line that matches at least one of the patterns is written to
     the standard output.

     grep is used for simple patterns and basic regular expressions (BREs);
     egrep can handle extended regular expressions (EREs).  See re_format(7)
     for more information on regular expressions.  fgrep is quicker than both
     grep and egrep, but can only handle fixed patterns (i.e. it does not
     interpret regular expressions).  Patterns may consist of one or more
     lines, allowing any of the pattern lines to match a portion of the input.

     zgrep, zegrep, and zfgrep act like grep, egrep, and fgrep, respectively,
     but accept input files compressed with the compress(1) or gzip(1) com-
     pression utilities.

     The following options are available:

     -A num, --after-context=num
             Print num lines of trailing context after each match.  See also
             the -B and -C options.
```

-a, --text
Treat all files as ASCII text. Normally grep will simply print ``Binary file ... matches'' if files contain binary characters. Use of this option forces grep to output lines matching the specified pattern.

-B num, --before-context=num
Print num lines of leading context before each match. See also the -A and -C options.

-b, --byte-offset
The offset in bytes of a matched pattern is displayed in front of the respective matched line.

-C[num, --context=num]
Print num lines of leading and trailing context surrounding each match. The default is 2 and is equivalent to -A 2 -B 2. Note: no whitespace may be given between the option and its argument.

-c, --count
Only a count of selected lines is written to standard output.

--colour=[when, --color=[when]]
Mark up the matching text with the expression stored in GREP_COLOR environment variable. The possible values of when can be `never', `always' or `auto'.

-D action, --devices=action
Specify the demanded action for devices, FIFOs and sockets. The default action is `read', which means, that they are read as if they were normal files. If the action is set to `skip', devices will be silently skipped.

-d action, --directories=action
Specify the demanded action for directories. It is `read' by default, which means that the directories are read in the same manner as normal files. Other possible values are `skip' to

silently ignore the directories, and `recurse' to read them recursively, which has the same effect as the -R and -r option.

-E, --extended-regexp
Interpret pattern as an extended regular expression (i.e. force grep to behave as egrep).

-e pattern, --regexp=pattern
Specify a pattern used during the search of the input: an input line is selected if it matches any of the specified patterns. This option is most useful when multiple -e options are used to specify multiple patterns, or when a pattern begins with a dash (`-').

--exclude
If specified, it excludes files matching the given filename pattern from the search. Note that --exclude patterns take priority over --include patterns, and if no --include pattern is specified, all files are searched that are not excluded. Patterns are matched to the full path specified, not only to the filename component.

--exclude-dir
If -R is specified, it excludes directories matching the given filename pattern from the search. Note that --exclude-dir patterns take priority over --include-dir patterns, and if no --include-dir pattern is specified, all directories are searched that are not excluded.

-F, --fixed-strings
Interpret pattern as a set of fixed strings (i.e. force grep to behave as fgrep).

-f file, --file=file
Read one or more newline separated patterns from file. Empty pattern lines match every input line. Newlines are not considered part of a pattern. If file is empty, nothing is matched.

-G, --basic-regexp
Interpret pattern as a basic regular expression (i.e. force grep to behave as traditional grep).

-H Always print filename headers with output lines.

-h, --no-filename
Never print filename headers (i.e. filenames) with output lines.

--help Print a brief help message.

-I Ignore binary files. This option is equivalent to --binary-file=without-match option.

-i, --ignore-case
Perform case insensitive matching. By default, grep is case sensitive.

--include
If specified, only files matching the given filename pattern are searched. Note that --exclude patterns take priority over --include patterns. Patterns are matched to the full path specified, not only to the filename component.

--include-dir
If -R is specified, only directories matching the given filename pattern are searched. Note that --exclude-dir patterns take priority over --include-dir patterns.

-J, --bz2decompress
Decompress the bzip2(1) compressed file before looking for the text.

-L, --files-without-match
Only the names of files not containing selected lines are written to standard output. Pathnames are listed once per file searched. If the standard input is searched, the string ``(standard input)'' is written.

-l, --files-with-matches
Only the names of files containing selected lines are written to standard output. grep will only search a file until a match has been found, making searches potentially less expensive. Pathnames are listed once per file searched. If the standard input is searched, the string ``(standard input)'' is written.

--mmap Use mmap(2) instead of read(2) to read input, which can result in better performance under some circumstances but can cause undefined behaviour.

-m num, --max-count=num
Stop reading the file after num matches.

-n, --line-number
Each output line is preceded by its relative line number in the file, starting at line 1. The line number counter is reset for each file processed. This option is ignored if -c, -L, -l, or -q is specified.

--null Prints a zero-byte after the file name.

-O If -R is specified, follow symbolic links only if they were explicitly listed on the command line. The default is not to follow symbolic links.

-o, --only-matching
Prints only the matching part of the lines.

-p If -R is specified, no symbolic links are followed. This is the default.

-q, --quiet, --silent
Quiet mode: suppress normal output. grep will only search a file until a match has been found, making searches potentially less expensive.

-R, -r, --recursive
Recursively search subdirectories listed.

-S If -R is specified, all symbolic links are followed. The default is not to follow symbolic links.

-s, --no-messages
Silent mode. Nonexistent and unreadable files are ignored (i.e. their error messages are suppressed).

-U, --binary
Search binary files, but do not attempt to print them.

-V, --version
Display version information and exit.

-v, --invert-match
Selected lines are those not matching any of the specified patterns.

-w, --word-regexp
The expression is searched for as a word (as if surrounded by `[[:<:]]' and `[[:>:]]'; see re_format(7)).

-x, --line-regexp
Only input lines selected against an entire fixed string or regular expression are considered to be matching lines.

-y Equivalent to -i. Obsoleted.

-Z, -z, --decompress
Force grep to behave as zgrep.

--binary-files=value
Controls searching and printing of binary files. Options are binary, the default: search binary files but do not print them; without-match: do not search binary files; and text: treat all files as text.

--context[=num]
Print num lines of leading and trailing context. The default is 2.

--line-buffered
Force output to be line buffered. By default, output is line buffered when standard output is a terminal and block buffered otherwise.

If no file arguments are specified, the standard input is used.

ENVIRONMENT

GREP_OPTIONS May be used to specify default options that will be placed at the beginning of the argument list. Backslash-escaping is not supported, unlike the behavior in GNU grep.

EXIT STATUS

The grep utility exits with one of the following values:

0 One or more lines were selected.
1 No lines were selected.
>1 An error occurred.

EXAMPLES

To find all occurrences of the word `patricia' in a file:

```
$ grep 'patricia' myfile
```

To find all occurrences of the pattern `.Pp' at the beginning of a line:

```
$ grep '^\.Pp' myfile
```

The apostrophes ensure the entire expression is evaluated by grep instead of by the user's shell. The caret `^' matches the null string at the beginning of a line, and the `\' escapes the `.', which would otherwise match any character.

To find all lines in a file which do not contain the words `foo' or `bar':

```
$ grep -v -e 'foo' -e 'bar' myfile
```

A simple example of an extended regular expression:

```
$ egrep '19|20|25' calendar
```

Peruses the file `calendar' looking for either 19, 20, or 25.

SEE ALSO

ed(1), ex(1), gzip(1), sed(1), re_format(7)

STANDARDS

The grep utility is compliant with the IEEE Std 1003.1-2008 (``POSIX.1'') specification.

The flags [-AaBbCDdGHhIJLmoPRSUVwZ] are extensions to that specification, and the behaviour of the -f flag when used with an empty pattern file is left undefined.

All long options are provided for compatibility with GNU versions of this utility.

Historic versions of the grep utility also supported the flags [-ruy]. This implementation supports those options; however, their use is strongly discouraged.

HISTORY

The grep command first appeared in Version 6 AT&T UNIX.

BUGS

The grep utility does not normalize Unicode input, so a pattern containing composed characters will not match decomposed input, and vice versa.

AWK

From **A**ho, **W**einberger and **K**ernighan, the names of its authors.

- Is a programming language which allows us to create variables, use conditionals and loops.

Very good when managing data on the terminal.

Generally, the syntax goes as follows

```
awk '/pattern1/ {What to do}
/pattern2/ {What to do}' Group_of_files
```

In [7]:
```
cat file1.txt
```

```
THIS LINE IS AN UPPER CASE LINE IN THIS FILE.
ESTA ES UNA LÍNEA EN MAYÚSCULAS.

this line is the a lower case line in this file.
esta es una línea en minúsculas.

This Line Has All Its First Character Of The Word With Upper Case.
Esta Línea Tiene La Primera Letra De Cada Palabra En Mayúsulas.

And this is the last line.
Esta es la última línea.
```

To print a complete file usinng `awk` , we use the instruction print

In [8]:
```
awk '{print;}' file1.txt
```

```
THIS LINE IS AN UPPER CASE LINE IN THIS FILE.
ESTA ES UNA LÍNEA EN MAYÚSCULAS.

this line is the a lower case line in this file.
esta es una línea en minúsculas.

This Line Has All Its First Character Of The Word With Upper Case.
Esta Línea Tiene La Primera Letra De Cada Palabra En Mayúsulas.

And this is the last line.
Esta es la última línea.
```

The patterns we are using can be used to search similarly as `grep`

In [9]:
```
awk '/This/' file1.txt
```

```
This Line Has All Its First Character Of The Word With Upper Case.
```

Is usual, in different programming languages that `$` means a column, or a numeration that is implicit, so it is the case.

In [10]:
```
awk '{print $2;}' file1.txt
```

```
LINE
ES

line
es

Line
Línea

this
es
```

And can be combined!

In [11]:
```
awk '{print $5,$2;}' file1.txt
```

```
UPPER LINE
EN ES

a line
en es

Its Line
Primera Línea

last this
línea. es
```

There are also some variables already defined,

In [12]:
```
awk '{print $5,$2,$NF;}' file1.txt
```

```
UPPER LINE FILE.
EN ES MAYÚSCULAS.

a line file.
en es minúsculas.

Its Line Case.
Primera Línea Mayúsulas.

last this line.
línea. es línea.
```

Let's use our own data,

In [13]:
```
awk '{print;}' DataHw1a.dat
```

```
1       jf.escobarr     José    17      Geociencias
2       asesquivel      Andrea  18      Fisica
3       df.rodriguezg   Diego   20      Fisica
4       l.garciae       Laura   18      Geociencias
5       aj.mendoza      Alberto 19      geociencias
6       ke.solano       Kevin   17      Geociencias
7       sm.morelli      Sebastian       18      Física
8       bj.anaya        Bryan   19      Física
9       sm.gutierrez    Sharol  19      Física
10      am.forero1      ana     20      Ingenieria Electronica y Geociencias
11      ac.melo Angie   17      Física
12      b.taborda       Brayan  18      Física
13      df.vegao        Daniel  18      Geociencias
14      lp.cardozo      Lina    18      Ingeniería de Sistema
15      s.pastrana      Santiago        21      Ingeniería Industrial
16      v.castilloc     Valeria 19      Fisica
17      f_ariasc        Fabio   18      Física
18      e.cayon Edgardo 20      Fisica
19      sp.joven        Susan   19      Ingeniería de Sistemas
20      e.gonzalezr     Emilio  20      Ingeniería de Sistemas
21      s.posadac       Sofia   18      Ingeniería Mecánica
22      js.velasco      Juan    18      Ingenieria Ambietal y Geocienicas
23      j.sevillam      Mauricio        80      Prof
```

Data can be filtred!!

- By number

In [14]:
```
awk '$4<19' DataHw1a.dat
```

```
1       jf.escobarr     José    17      Geociencias
2       asesquivel      Andrea  18      Fisica
4       l.garciae       Laura   18      Geociencias
6       ke.solano       Kevin   17      Geociencias
7       sm.morelli      Sebastian       18      Física
11      ac.melo Angie   17      Física
12      b.taborda       Brayan  18      Física
13      df.vegao        Daniel  18      Geociencias
14      lp.cardozo      Lina    18      Ingeniería de Sistema
17      f_ariasc        Fabio   18      Física
21      s.posadac       Sofia   18      Ingeniería Mecánica
22      js.velasco      Juan    18      Ingenieria Ambietal y Geocienicas
```

- Or by strings!

In [15]:
```
awk '$5 ~ /Ingeniería/' DataHw1a.dat
```

```
14      lp.cardozo      Lina    18      Ingeniería de Sistema
15      s.pastrana      Santiago        21      Ingeniería Industrial
19      sp.joven        Susan   19      Ingeniería de Sistemas
20      e.gonzalezr     Emilio  20      Ingeniería de Sistemas
21      s.posadac       Sofia   18      Ingeniería Mecánica
```

Let us explore the manual,

In [16]:
```
man awk
```

```
AWK(1)                                                    AWK(1)



awk

NAME
       awk - pattern-directed scanning and processing language

SYNOPSIS
       awk [ -F fs ] [ -v var=value ] [ 'prog' | -f progfile ] [ file ...  ]

DESCRIPTION
       Awk scans each input file for lines that match any of a set of patterns
       specified literally in prog or in one or more  files  specified  as  -f
       progfile.  With  each  pattern  there can be an associated action that
       will be performed when a line of a file matches the pattern.  Each line
       is  matched  against the pattern portion of every pattern-action state-
       ment; the associated action is performed for each matched pattern.  The
       file  name  - means the standard input.  Any file of the form var=value
       is treated as an assignment, not a filename, and  is  executed  at  the
       time  it  would  have been opened if it were a filename.  The option -v
       followed by var=value is an assignment to be done before prog  is  exe-
       cuted;  any  number  of  -v  options  may be present.  The -F fs option
       defines the input field separator to be the regular expression fs.

       An input line is normally made up of fields separated by  white  space,
       or by regular expression FS.  The fields are denoted $1, $2, ..., while
       $0 refers to the entire line.  If FS is null, the input line  is  split
       into one field per character.

       A pattern-action statement has the form

              pattern { action }
```

A missing { action } means print the line; a missing pattern always matches. Pattern-action statements are separated by newlines or semicolons.

An action is a sequence of statements. A statement can be one of the following:

```
if( expression ) statement [ else statement ]
while( expression ) statement
for( expression ; expression ; expression ) statement
for( var in array ) statement
do statement while( expression )
break
continue
{ [ statement ... ] }
expression                  # commonly var = expression
print [ expression-list ] [ > expression ]
printf format [ , expression-list ] [ > expression ]
return [ expression ]
next                        # skip remaining patterns on this input
line
nextfile                    # skip rest of this file, open next, sta
rt at top
delete array[ expression ]# delete an array element
delete array                # delete all elements of array
exit [ expression ]     # exit immediately; status is expression
```

Statements are terminated by semicolons, newlines or right braces. An empty expression-list stands for $0. String constants are quoted " ", with the usual C escapes recognized within. Expressions take on string or numeric values as appropriate, and are built using the operators + - * / % ^ (exponentiation), and concatenation (indicated by white space). The operators ! ++ -- += -= *= /= %= ^= > >= < <= == != ?: are also available in expressions. Variables may be scalars, array elements (denoted x[i]) or fields. Variables are initialized to the null string. Array subscripts may be any string, not necessarily numeric; this allows for a form of associative memory. Multiple subscripts such as [i,j,k] are permitted; the constituents are concatenated, separated

by the value of SUBSEP.

The print statement prints its arguments on the standard output (or on a file if >file or >>file is present or on a pipe if |cmd is present), separated by the current output field separator, and terminated by the output record separator. file and cmd may be literal names or parenthesized expressions; identical string values in different statements denote the same open file. The printf statement formats its expression list according to the format (see printf(3)). The built-in function close(expr) closes the file or pipe expr. The built-in function fflush(expr) flushes any buffered output for the file or pipe expr.

The mathematical functions exp, log, sqrt, sin, cos, and atan2 are built in. Other built-in functions:

length the length of its argument taken as a string, or of $0 if no argument.

rand random number on (0,1)

srand sets seed for rand and returns the previous seed.

int truncates to an integer value

substr(s, m, n)
the n-character substring of s that begins at position m counted from 1.

index(s, t)
the position in s where the string t occurs, or 0 if it does not.

match(s, r)
the position in s where the regular expression r occurs, or 0 if it does not. The variables RSTART and RLENGTH are set to the position and length of the matched string.

split(s, a, fs)

splits the string s into array elements a[1], a[2], ..., a[n], and returns n. The separation is done with the regular expression fs or with the field separator FS if fs is not given. An empty string as field separator splits the string into one array element per character.

sub(r, t, s)
substitutes t for the first occurrence of the regular expression r in the string s. If s is not given, $0 is used.

gsub same as sub except that all occurrences of the regular expression are replaced; sub and gsub return the number of replacements.

sprintf(fmt, expr, ...)
the string resulting from formatting expr ... according to the printf(3) format fmt

system(cmd)
executes cmd and returns its exit status

tolower(str)
returns a copy of str with all upper-case characters translated to their corresponding lower-case equivalents.

toupper(str)
returns a copy of str with all lower-case characters translated to their corresponding upper-case equivalents.

The ``function'' getline sets $0 to the next input record from the current input file; getline <file sets $0 to the next record from file. getline x sets variable x instead. Finally, cmd | getline pipes the output of cmd into getline; each call of getline returns the next line of output from cmd. In all cases, getline returns 1 for a successful input, 0 for end of file, and -1 for an error.

Patterns are arbitrary Boolean combinations (with ! || &&) of regular expressions and relational expressions. Regular expressions are as

defined in re_format(7). Isolated regular expressions in a pattern apply to the entire line. Regular expressions may also occur in relational expressions, using the operators ~ and !~. /re/ is a constant regular expression; any string (constant or variable) may be used as a regular expression, except in the position of an isolated regular expression in a pattern.

A pattern may consist of two patterns separated by a comma; in this case, the action is performed for all lines from an occurrence of the first pattern though an occurrence of the second.

A relational expression is one of the following:

```
expression matchop regular-expression
expression relop expression
expression in array-name
(expr,expr,...) in array-name
```

where a relop is any of the six relational operators in C, and a matchop is either ~ (matches) or !~ (does not match). A conditional is an arithmetic expression, a relational expression, or a Boolean combination of these.

The special patterns BEGIN and END may be used to capture control before the first input line is read and after the last. BEGIN and END do not combine with other patterns.

Variable names with special meanings:

CONVFMT
: conversion format used when converting numbers (default %.6g)

FS
: regular expression used to separate fields; also settable by option -Ffs.

NF
: number of fields in the current record

NR
: ordinal number of the current record

FNR ordinal number of the current record in the current file

FILENAME
the name of the current input file

RS input record separator (default newline)

OFS output field separator (default blank)

ORS output record separator (default newline)

OFMT output format for numbers (default %.6g)

SUBSEP separates multiple subscripts (default 034)

ARGC argument count, assignable

ARGV argument array, assignable; non-null members are taken as filenames

ENVIRON
array of environment variables; subscripts are names.

Functions may be defined (at the position of a pattern-action statement) thus:

```
function foo(a, b, c) { ...; return x }
```

Parameters are passed by value if scalar and by reference if array name; functions may be called recursively. Parameters are local to the function; all other variables are global. Thus local variables may be created by providing excess parameters in the function definition.

EXAMPLES

```
length($0) > 72
```
Print lines longer than 72 characters.

```
{ print $2, $1 }
```
Print first two fields in opposite order.

```
BEGIN { FS = ",[ \t]*|[ \t]+" }
      { print $2, $1 }
```
Same, with input fields separated by comma and/or blanks and tabs.

```
     { s += $1 }
END  { print "sum is", s, " average is", s/NR }
```
Add up first column, print sum and average.

```
/start/, /stop/
```
Print all lines between start/stop pairs.

```
BEGIN     {    # Simulate echo(1)
     for (i = 1; i < ARGC; i++) printf "%s ", ARGV[i]
     printf "\n"
     exit }
```

SEE ALSO

lex(1), sed(1)

A. V. Aho, B. W. Kernighan, P. J. Weinberger, The AWK Programming Language, Addison-Wesley, 1988. ISBN 0-201-07981-X

BUGS

There are no explicit conversions between numbers and strings. To force an expression to be treated as a number add 0 to it; to force it to be treated as a string concatenate "" to it.

The scope rules for variables in functions are a botch; the syntax is worse.