

Tutorial 4: The Unix programming environment

Georgios Gousios

`gousiosg@aueb.gr`

Department of Management Science and Technology

Athens University of Economics and Business

Revision : 1079

Contents

- Regular expressions
- Text stream processing
- Text editors
- The GNU compiler suite
- The GNU debugger

Regular expressions

- Used, in various dialects, by many text processing tools.
- Can be viewed as a recipe for extracting or otherwise manipulating text.
- Dialects
 - Traditional: Those supported by the `grep` and `sed` tools
 - Extended or `POSIX` compatible: Extensions to the above set to support, among others, *i18n*. Supported by `egrep` and `awk`
 - Perl-compatible: The superset of all extensions used in the Perl language

Regular expression syntax

Symbol	Action
.	Match any character
*	Match zero or more of the preceding characters
^	Match the beginning of a line
\$	Match the end of a line
[]	Match one from the set
[^]	Match everything not in the set
A-Z, a-z, 0-9	The range of capitals, small case letters and digits
\char	Don't attach special meaning to char
\n \t \ \e	Special characters for newline, tab, space, escape

Regular expression examples

RE	What matches
----	--------------

<code>te.*</code>	
-------------------	--

Regular expression examples

RE	What matches
----	--------------

<code>te.*</code>	Match anything starting with <code>te</code> and continuing with anything else, e.g. <code>test</code> , <code>eteeewt</code> , <code>te</code>
-------------------	---

<code>[A-Z0-9][a-z]*</code>	
-----------------------------	--

Regular expression examples

RE	What matches
<code>te.*</code>	Match anything starting with <code>te</code> and continuing with anything else, e.g. <code>test</code> , <code>eteeewt</code> , <code>te</code>
<code>[A-Z0-9][a-z]*</code>	Match anything starting with a capital letter or digit followed by any number of small characters, e.g. <code>London is huge</code> but not <code>london is huge</code> .
<code>^[a-z].*\\$</code>	

Regular expression examples

RE	What matches
<code>te.*</code>	Match anything starting with <code>te</code> and continuing with anything else, e.g. <code>test</code> , <code>eteeewt</code> , <code>te</code>
<code>[A-Z0-9][a-z]*</code>	Match anything starting with a capital letter or digit followed by any number of small characters, e.g. <code>London is huge</code> but not <code>london is huge</code> .
<code>^[a-z].*\\$\$</code>	Match a line that starts with either <code>a</code> , <code>z</code> or <code>e</code> , ends with a dollar sign and has anything else between e.g. <code>aad wd1* \$\$</code>

Extended regular expression syntax

Symbol	Action
+	Match one or more of the preceding
?	Match zero or one of the preceding
< >	Match the beginning or end of a word
()	Group RE and store matched string for later use
\1 \2	The first or second etc matched group
	Match either left group or right group
{num1, num2 }	Match the preceding at least num1 and at max num2 times. One of num1 or num2 can be omitted.

POSIX character classes

Class	Matches
<code>[:alnum:]</code>	Printable characters (includes whitespace)
<code>[:alpha:]</code>	Alphabetic characters
<code>[:blank:]</code>	Space and tab characters
<code>[:cntrl:]</code>	Control characters
<code>[:digit:]</code>	Numeric characters
<code>[:graph:]</code>	Printable and visible (non-space) characters
<code>[:lower:]</code>	Lowercase characters
<code>[:print:]</code>	Printable characters (includes whitespace)
<code>[:punct:]</code>	Punctuation characters
<code>[:space:]</code>	Whitespace characters
<code>[:upper:]</code>	Uppercase characters
<code>[:xdigit:]</code>	Hexadecimal digits

Extended Regular expression examples

- `(m|n|kn|r)ight`

Extended Regular expression examples



- `(m|n|kn|r)ight`
Match might, night, knight, right
- `a[n]? (simple|easy) problem`



Extended Regular expression examples



- `(m|n|kn|r)ight`

Match `might`, `night`, `knight`, `right`

- `a[n]? (simple|easy) problem`

Match `a`, possibly followed by `n`, followed by a space, followed by either `simple` or `easy` followed by a space and the word `problem`

- `80[23]?86|PENTIUM I{1,3}V?`



Extended Regular expression examples



- `(m|n|kn|r)ight`
Match might, night, knight, right
- `a[n]? (simple|easy) problem`
Match a, possibly followed by n, followed by a space, followed by either simple or easy followed by a space and the word problem
- `80[23]?86|PENTIUM I{1,3}V?`
Match the names of all PC processors
- `[[:alnum:]]*[[:space:]]{2,20}$`



Extended Regular expression examples



- `(m|n|kn|r)ight`
Match might, night, knight, right
- `a[n]? (simple|easy) problem`
Match a, possibly followed by n, followed by a space, followed by either simple or easy followed by a space and the word problem
- `80[23]?86|PENTIUM I{1,3}V?`
Match the names of all PC processors
- `[[:alnum:]]*[[:space:]]{2,20}$`
Match lines ending with at least 2 and at max 20 whitespace characters
- `[[:alnum:]]*[[:space:]]{2,20}$`



Extended Regular expression examples

- `(m|n|kn|r)ight`

Match might, night, knight, right

- `a[n]? (simple|easy) problem`

Match a, possibly followed by n, followed by a space, followed by either simple or easy followed by a space and the word problem

- `80[23]?86|PENTIUM I{1,3}V?`

Match the names of all PC processors

- `[[:alnum:]]* [[:space:]]{2,20}$`

Match lines ending with at least 2 and at max 20 whitespace characters

- `[[:alnum:]]* [[:space:]]{2,20}$`

Match lines ending with at least 2 and at max 20 whitespace characters

Extended Regular expression examples

- `([1-2]?[1-9]{1,2}){3,}[1-2]?[0-9]{1,2}`

Match lines containing an IP address

- Extreme example

```
([-\w. ]+)\s+([-\w ]+)\s+([-\w ]+)\s+\[(\d+)\]/(\w+)\]/(\d+)\:(\d+)\:(\d+)\[^\]]+?\]\s+"([-\w ]+)\s*([^\s]*)["]*?\]\s+(\d+)\s+([- \d ]+)\s+"([^\s]*)"\s+"([^\s]*)"\s*
```

Match (and extract info from) the Apache log file:

```
195.251.249.158- - [22/Dec/2004:13:57:21+0200]
"GET/~renegade/ps2005.rar HTTP 1.1"200 7797240"- "
"Mozilla/5.0(Windows;U;WindowsNT5.1;en-US; rv:1.7.5)
Gecko 20041107 Firefox/1.0"
```

Tools of the trade

- Already seen: `cat`, `less`, `more`, `head`, `tail`,
`wc`, `xargs`
- More text tools: `sort`, `uniq`, `cut`, `paste`, `tr`,
`printf`, `rev`, `diff`
- We will explain in detail: `find`, `[e]grep`, `sed`

Tools of the trade - `grep`

- `grep options REGEXP file` - print lines matching the REGEXP
 - `-r` recursively search directories for files
 - `-v` Inverse; print lines that don't match
 - `-i` case insensitive matching
 - `-l` only print files that match
 - `-c` count matching lines per file
 - `-L` only print files that match
 - `-n` print matching line number
- `egrep` - A version of `grep` that understands extended regular expressions
- `fgrep` - Uses input file as a source of patterns for matching

Tools of the trade - `find`

`find options path1...pathn expr` - traverse a directory hierarchy

`options` can be

- `-E` support extended RE
- `-s` traverse in lexicographical order

`expr` can be

- `-exec util` - exec the specified `util` for each file
- `-{i}regex` - print paths matched by the `regex`
- `[max,min]depth n` - decent at most or at least `n` directory levels
- `-newer file` - only display files newer than `file`

Many expressions can be combined with the logical `-or` and `-and` operators and be negated with the `-not` operator.

Tools of the trade - `sort`/`uniq`

- `sort <file>` - sort lines of text files. By default, sorts lines in ascending order according to the first word of each line
 - `+<num>` sort by column *num*
 - `-t<delimiter>` assume columns are separated by *<delimiter>*
 - `-r` Reverse results
 - `-n` Sort according to a number at the beginning of a line
- `uniq <file>` - filter out or report repeated lines in *file*
 - `-c` report number of occurrences
 - `-d` output repeated lines

```
cat file |grep -v "^[ \n\t]*$" |sort|uniq -c|sort -rn
```

Tools of the trade - cut / paste

- `cut` - extract portions from text lines
 - `-d` column separator character
 - `-f[num, ...]` field numbers to extract

```
cut -d: -f1,6 /etc/passwd
```

extract usernames and home directories from the user database

- `paste file1...n` - merge corresponding lines of files
 - `-d 'char'` separate fields by char
 - `-s` merge all lines of one file

Tools of the trade - printf

`printf format arguments,...` - Print arguments formatted according to *format*

The format string contains 3 types of objects: characters, which are copied verbatim, escaped characters (`\t` `\n` `\\` `\a`) and format specifications, which cause arguments to be printed in a specific format

Format specifications can be:

- `%d`, `%o`, `%X`: Print argument as decimal, octal or hexadecimal
- `%.numf`: Print argument using *num* precision points
- `%s`: Print argument as string
- `%c`: Print first character of argument

```
printf "Hello %s\n" $USER - Welcomes the user
```

```
ls -l `stat -f "%z" file` && printf "Size is: 0x%x" $s
```

Tools of the trade - diff

`diff file1 file2` - show differences in files

```
_____ file1 _____           _____ file2 _____  
int main(){                          1 #include <stdio.h>  
  printf("Hello world");              2  
                                       3 int main(){  
                                       4   printf("Hello dmst");  
                                       5 }
```

```
_____ Output _____  
Book$ diff lab4/1.txt lab4/2.txt  
a1,2  
#include <stdio.h>                   > = lines added  
  
c4  
printf("Hello world");                < = lines removed  
--                                     -- = line replacement  
printf("Hello dmst");
```

Tools of the trade - patch

The output of `diff` can be processed by the `patch` program in order to:

- transfer changes between source files on different computers
- get changes out of a versioning system

`patch < file`. *file* must be in the current directory

```
Book:~ george$ echo "test"> test
```

```
Book:~ george$ cp test test.new
```

```
Book:~ george$ echo "new line">>test.new
```

```
Book:~ george$ diff test.new test > patch
```

```
Book:~ george$ patch -p0 test < patch
```

Tools of the trade - `tr`

`tr from_char to_char` - Change input character(s) *from_char* to *to_char*

Important arguments:

- `-s` - 'Squeeze' multiple occurrences of *from_char*
- `-z` - Delete occurrences of *from_char*

`tr ' () ' ' { } ' - Change parentheses to braces`

`tr -s ' ' ' ' - Change multiple spaces to a single space`

`tr '[a-z]' '[A-Z]' - Change lowercase letters to capitals`

`tr -d '[0-9]' - Delete all digits`

Tools of the trade - sed

sed *'cmd' file* - stream editor

cmd = [addr1,addr2][!] function [arguments]

sed applies the editing commands specified by the `function`

argument to lines from `addr1` to `addr2`. If addresses are omitted,

all lines are considered.

Common commands

- `s/pattern/replstr/[g,p]` For each line replace matched `pattern` with `replstr`. Argument `g` forces all occurrences of `pattern` in a line to be replaced. `p` prints the changed line.
- `/pattern/=` Print line number for lines that matched `pattern`
- `y/abc/xyz/` Translate characters. Change every instance of `a` to `x`, `b` to `y`, `c` to `z`

Tools of the trade - sed

- `sed '5d'` - Delete line 5
- `sed '/[Tt]est/d'` - Delete lines containing the words test or Test
- `sed '1,10s/unix/UNIX/g'` - Change all occurrences of unix to UNIX in lines 1 to 10
- `sed 's/\(.*\) : \(.*\) / \2 : \1 /'` - Exchange columns in input

Text editors

- Historically, Unix featured single line editors (`ed`) and stream editors (`sed` and `awk`)
- The `vi` editor was introduced by Bill Joy as a full screen alternative.
- The `emacs` editor is a 'product' of the GNU project, initially written by Richard Stallman.
- Choosing between them is almost a matter of religion today.

The vi editor



- 2 modes of operation
 - Command mode (default), (ESC)
 - Insert mode, acts like a typewriter (i)

- General form of commands

(command)(number)(text object)

- *(text object)* Select a portion of text
 - *(number)(optional)* Multiplies the effect of the text object selection
 - *(command)(optional)* change, delete, or yank
- Start with `vi <filename>`



Cursor movement



- Basic movement
 - Up : `k` or Up Arrow
 - Down: `j` or Down Arrow
 - Left: `h` or Left Arrow
 - Right: `l` or Right Arrow
- A number *m* can precede the movement command to repeat the command *m*, e.g. the command `12j` moves the cursor 12 lines below
- `^`: Move to the beginning of the line
- `$`: Move to the end of the line



Movement in page



- `nG`: Move to line n
- `G`: Move to last line
- `ctrl+F`: Move one page down
- `ctrl+B`: Move one page up
- `w`: Move forward, one word
- `b`: Move backward, one word
- `e`: Move to the end of the current word



Basing editing

- To insert text: Move to the desired location, press `i` and write
- `c<what>`: Change text from cursor
`<what>` can be:
 - `w`: change a word
 - `c`: change a line
 - `~`: change case
- `r`: Change a letter
- `d<what>`: Delete `<what>`
 - `w`: delete a word
 - `W`: delete remaining sentence
 - `d`: delete a line
- `x`: Delete a character
- `~`: Repeat last edit command

Search and Replace

`vi` uses a syntax similar to `sed` for replacement commands

- `:s/old/new/` - Replace all occurrences of `old` with `new`
- `george`

Bibliography

References

- [1] Debra Cameron, James Elliott, and Marc Loy. *Learning GNU Emacs*. O'Reilly, 3rd 2004.
- [2] Dale Dougherty and Arnold Robbins. *sed and awk*. O'Reilly and Associates, 2nd edition, March 1997.
- [3] B. W. Kernighan and R. Pike. *The Unix Programming Environment*. Prentice-Hall, Inc., 1984.
- [4] B. W. Kernighan and R. Pike. *The Practice of Programming*. Addison-Wesley, 1999.
- [5] Stephen G. Kochan and Patrick Wood. *Unix Shell Programming*. Sams Publishing, 3rd edition, February 2003.
- [6] Linda Lamb and Arnold Robbins. *Learning the vi Editor*, volume The Unix CD bookshelf. O' Reilly and Associates, 6th edition, November 1998.
- [7] Eric S. Raymond. *The Art of Unix Programming*. Addison-Wesley, 2003.
<http://www.faqs.org/docs/artu/>.

[?] [?] [?] [?] [?] [?] [?]