TEXT MANIPULATIONS

http://linoxide.com/how-tos/linux-grep-command-find-strings/

String manipulations

RegEx is a language for describing **patterns** in strings

- **`grep`** finds lines containing a pattern, and outputs them

- **`sed`** (stream editor) applies transformation rules to each line of text based on a pattern

- **`awk`** powerful text processing language

Regular expressions

.	Matches any single character **a.c** matches **abc, acc, etc**.
[]	Matches a set. **[abc]** matches **a, b, or c**. **[a-zA-Z]** matches any letter. **[0-9]** matches any number. "**^**" negates a set, **[^abc]** matches **d, e, f,** etc.
^	Starting position anchor. **^abc** finds lines starting with **abc**
$	Ending position anchor. **xyz$** finds lines ending with **xyz**
\	Escape symbol, to find special characters. **\*** will find "*****". **\\n** matches new line character, **\\t** – tab character
*****	Match the preceding element zero or more times. **a*b** matches **ab, aab, aaab,** etc.
Extended regular expressions	
?	Matches the preceding element zero or one time. **a*b** matches **b, ab,** but not **aab**
+	Matches the preceding element one or more times. **a+b** matches **ab, aab,** etc.
|	OR operator. "**abc|def**" matches **abc** or **def**

https://en.wikipedia.org/wiki/Regular_expression#POSIX_basic_and_extended

grep usage

Basic syntax: **`grep ”pattern" <filename>`**

- **`cat README.md | grep "use"`**
  outputs lines containing the pattern ”use”, non-case-sensitive, prints line numbers

- **`ls | grep "^[w|b]"`**
  lists files/directorys starting with ”w” or ”b”

Fine-tuning your grep

`-v` inverts the match

`-i` matches case insensitively

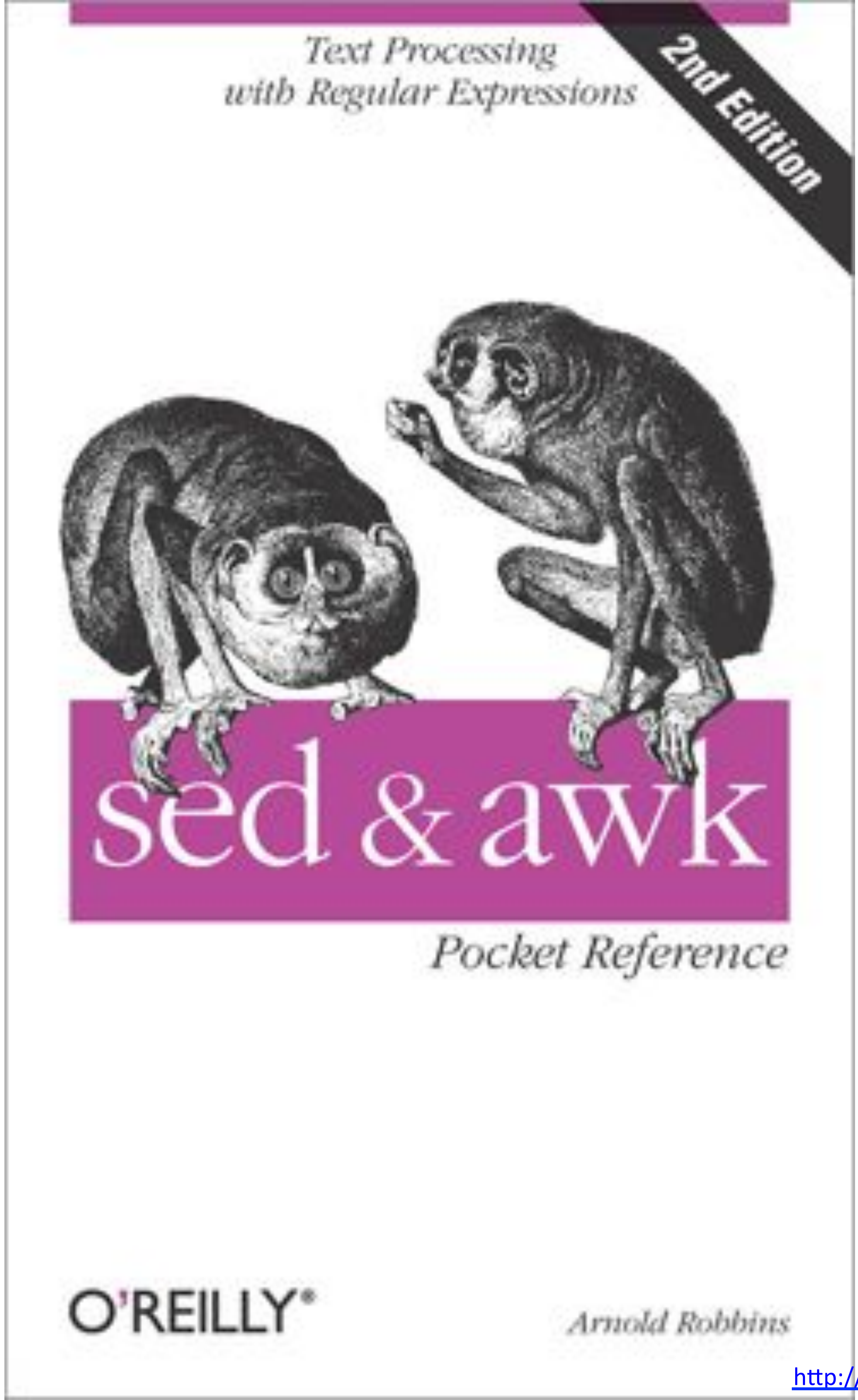
`-H` prints the matched filename

`-n` prints the line number

`-f <filename>` gets patterns from a file, each pattern on a new line

`-w` forces the pattern to match an entire word

`-x` forces patterns to match the whole line



http://shop.oreilly.com/product/9780596003524.do

sed - **s**tream **ed**itor

Most common usage – **substitute** a pattern with replacement. Basic syntax:

```
sed ‘s/pattern/replacement/’
```

-
  ```
  echo “The Internet is made of dogs” | sed ‘s/dogs/cats/’
  ```
  replaces “dogs” with cats, so the final output is “The Internet is made of cats”

-
  ```
  echo “dogs, dogs, dogs” | sed ‘s/dogs/cats/g’
  ```
  global substitution with “g” modifier. The final output is “cats, cats, cats”

DEMO

https://en.wikipedia.org/wiki/Cats_and_the_Internet

sed - **s**tream **ed**itor

- Special characters – escape with “\”

`echo "1*2*3" | sed 's/\*/-/g'` outputs “1-2-3”

- Regular expressions – use as in **`grep`**, with “-E” argument for extended regex

`echo "tic-tac-toe" | sed 's/[ia]/o/g' | sed 's/e$/c/'` **–** outputs “toc-toc-toc”

- Delete line(s) – **`sed ‘X[,Y]d’`** deletes line X through Y

`cat <filename> | sed ‘1d’` **–** deletes first line (e.g., header)

`cat <filename> | sed ‘10,37d’` **–** deletes lines from 10 through 37

awk

A more traditional programming language for text processing than sed. Awk stands for the names of its authors “Alfred **A**ho, Peter **W**einberger, and Brian **K**ernighan”

- Operates on “pieces” of a line = columns. A piece is defined as separated by space, tab, or pre-specified symbol (e.g., comma)

- Columns are enumerated, and can be addressed as **`$1, $2, $3`** ….
  `$0` represents the whole line

Conditional output with awk

Basic syntax: **`cat <filename> | awk ‘expression { action }’`**

- **`if (expression) {action} [ else {action} ]`**
- Boolean operators **`==, !=, >, >=, <, <=, &&, ||`**

- Print a line if the first column is “chr1”

```
awk '{if ($1 == "chr1”) print $0}’
awk '$1 == "chr1" {print $0}’
```

- Print columns 2 and 3, switched, if the 1st column is > 100

```
awk ‘{OFS=“\t”} $1 > 100 {print $3, $2}’
```

OFS – output field separator, “space” by default

awk goodies

- Arithmetics

```
awk ‘{print $1, $2+100, $3-100}’
```

prints first 3 columns, the 2nd numerical column is increased by 100, the 3rd is decreased by 100

- Number of columns

```
head <filename> | awk ‘{FS=“\t”} {print NF}’
```

using tab as a field separator, prints number of fields

- Sort files by the number of lines

```
wc -l *.bed | awk '{OFS="\t"} {print $2, $1}' |
sort -k2n
```

DEMO

man awk for more

http://www.pement.org/awk/awk1line.txt

Statistical command line goodies

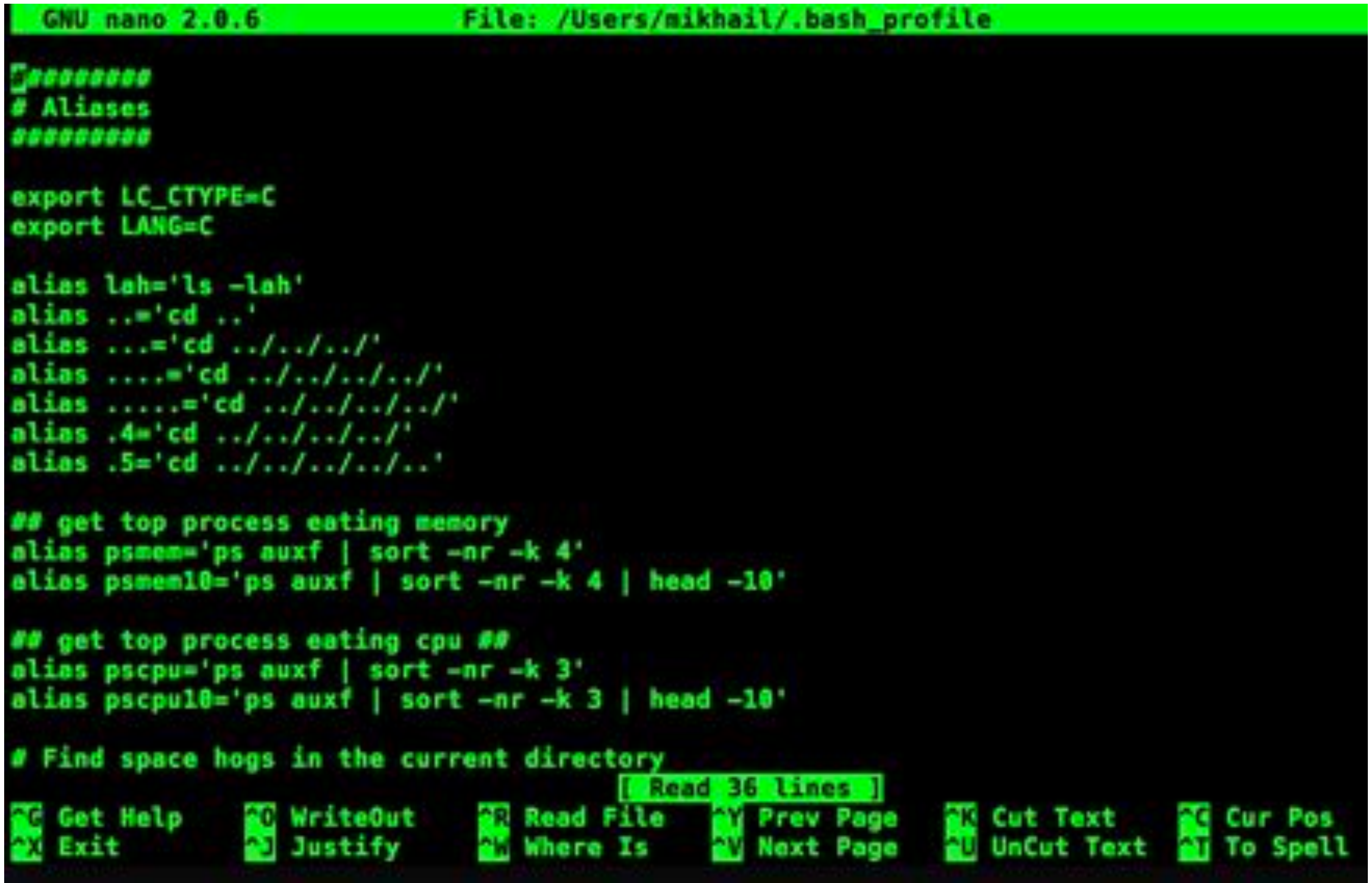
- **data_hacks**, https://github.com/bitly/data_hacks
  - Command line tools for data analysis
  - **`histogram.py`**
  - **`bar_chart.py`**
  - **`sample.py`**

- **datamash**, https://www.gnu.org/software/datamash/
  - summary statistics
  - transposing matrixes

KNOW YOUR TEXT EDITOR

Know your text editor

`nano` – default text editor of GNU operating systems



```
  GNU nano 2.0.6                File: /Users/mikhail/.bash_profile

##########
# Aliases
##########

export LC_CTYPE=C
export LANG=C

alias lah='ls -lah'
alias ..='cd ..'
alias ...='cd ../../../'
alias ....='cd ../../../../'
alias .....='cd ../../../../'
alias .4='cd ../../../../'
alias .5='cd ../../../../..'

## get top process eating memory
alias psmem='ps auxf | sort -nr -k 4'
alias psmem10='ps auxf | sort -nr -k 4 | head -10'

## get top process eating cpu ##
alias pscpu='ps auxf | sort -nr -k 3'
alias pscpu10='ps auxf | sort -nr -k 3 | head -10'

# Find space hogs in the current directory
                                   [ Read 36 lines ]
^G Get Help    ^O WriteOut    ^R Read File   ^Y Prev Page   ^K Cut Text    ^C Cur Pos
^X Exit        ^J Justify     ^W Where Is    ^V Next Page   ^U UnCut Text  ^T To Spell
```

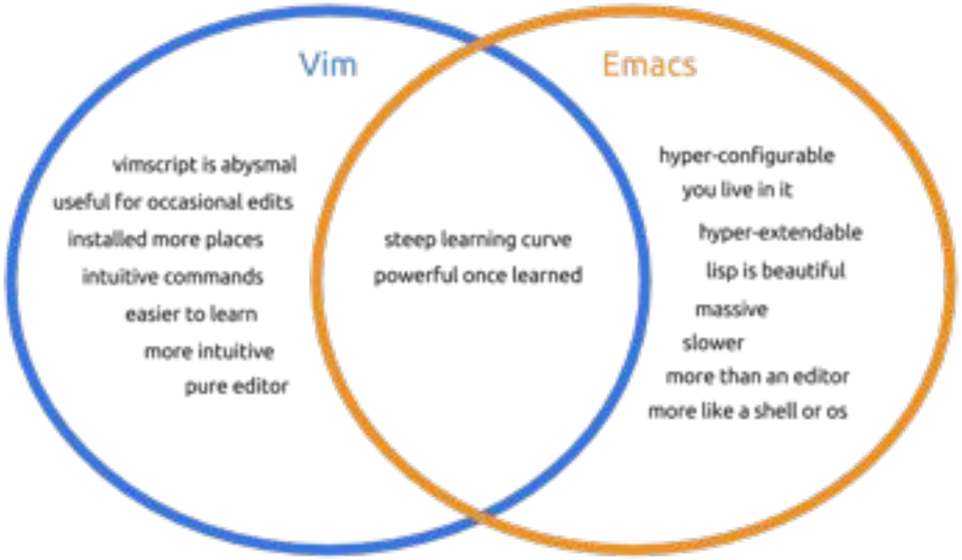
Vi, Vim, Emacs

Vi(m) Basics

- Created by Bill Joy, 1976
- **Advantages:** Supremely intuitive once basics are learned

Emacs basics

- Created by Richard Stallman, 1976
- **Advantages:** Unparalleled power and configuration



https://en.wikipedia.org/wiki/Vi

https://en.wikipedia.org/wiki/Emacs

https://danielmiessler.com/blog/differences-vi-vim-emacs/

vim basics

Start vim on a file: **`vim <filename>`**

Two modes:

- **`i`** editor mode, to type
- **`Esc`** command mode. Press "**`:`**" and enter a command

Commands:

- **`:w`** write changes
- **`:wq`** write changes and quit
- **`:q!`** force quit and ignore changes

Basic vim commands

`k, j, l, h`, or **`arrows`**	navigation
`v`	(visually) select characters
`V (shift-v)`	(visually) select whole lines
`d`	cut (delete) into clipboard
`dd`	cut the whole line
`y`	copy (yank) into clipboard
`P (shift-p)`	paste from clipboard
`u`	undo

Find and replace in vim

In command mode:

- **`/pattern`** search for pattern, "**`n`**" – next instance
- **`:s/pattern/replacement/g`** search and replace

- **`:help tutor`** learn more vim