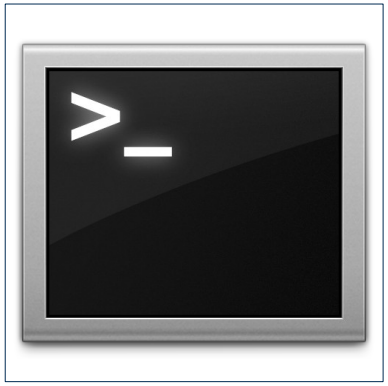
Linux Environment for Bioinformatics



Algoritmos Avançados de Bioinformática 2018

BioLinux as a Virtual Machine

Documentation for installation steps:

- http://environmentalomics.org/bio-linux-installation/

Running Bio-Linux as a VM with VirtualBox

- VirtualBox is a free and powerful cross-platform VM manager found at **http://virtualbox.org**

- Bio-Linux is a 64-bit operating system.
- Full documentation at: http://environmentalomics.org/bio-linux-installation/

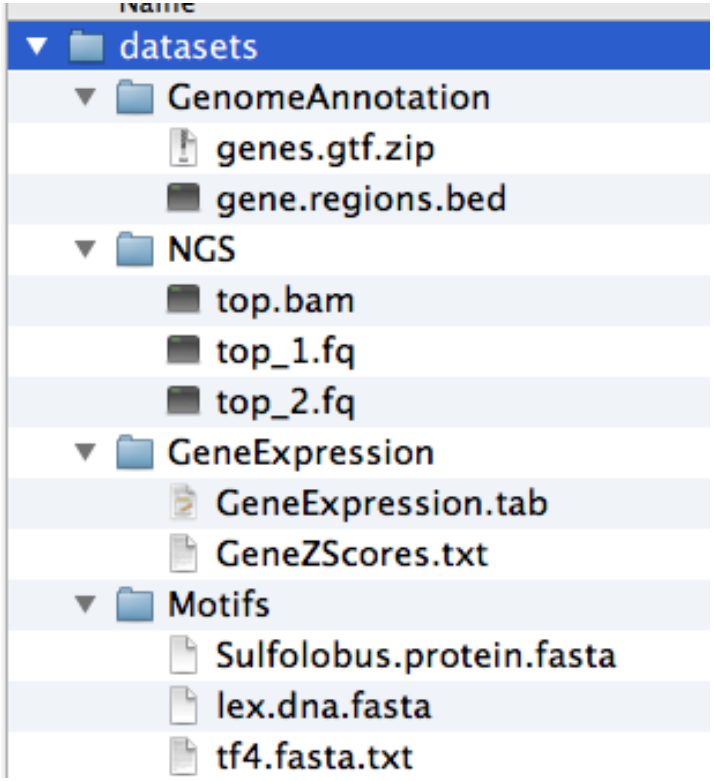
- Steps:
  1. > 40GB free disk space.
  2. Download and install the appropriate version of VirtualBox.
  3. Download the OVA file of Bio-linux.

  4. Start VirtualBox and select Import Appliance from the File menu and import the .ova file (don't worry that it says you need an OVF file).
  5. When importing the appliance, select the option to reinitialize the MAC addresses of network cards.
  6. Start the VM and if you see a log-in screen, log in as user manager with password also manager.
  7. Once this is working, you can delete the .ova file to save space.

Datasets for exercises

- Download zip file with example files to be used from here: https://www.dropbox.com/s/8pjb8skzkx7ruw3/datasets.zip?dl=0

Open firefox and in the url paste the link above. Data will be saved in /home/manager/Downloads/



Getting familiar with terminal/command lines

- Home directory is (typically) the starting point and is the root directory that for the structure that contains the user files, directories and programs.
- Basic managing linux operations includes: listing (ls), changing directory (cd) copying files (cp), removing file (rm), moving files (mv)

```
# open a terminal window
# obtaining the full path of current directory
$ pwd

# list the contents of the directory
$ ls
$ ls -la

# changing directory
$ cd Documents
$ cd        # changes to home folder
$ cd ..     # changes to the upper directory, note the space
$ cd /home/manager/Documents/
$ cd ~manager   # a shortcut for home directory

# create a directory
$ mkdir AAB2
$ cd AAB2
```

Getting familiar with terminal/command lines

```
# in folder AAB2 create a sub-folder
$ mkdir Exercises

# create an empty file
$ touch exercises.txt
$ echo "EXERCISES FROM COURSE AAB 2016/2017" >> exercises.txt
$ cat exercises.txt

# copy a file from one point to another
$ cp exercises.txt Exercises/

# if you are already in the target directory
$ cp ../exercises.txt .
# copy multiple files
$ cp ../* .

$ cd /home/manager/Documents/AAB2
$ cp /home/manager/Downloads/datasets.zip .
```

Getting familiar with terminal/command lines

```
# remove a file
$ rm exercises.txt

# move files and rename
$ mv /home/manager/Documents/AAB2/Exercises/exercises.txt
/home/pferreira/AAB2/exercises.txt
$ mv ex1.txt ex1a.txt


# clears the screen
$ clear

# history of commands
$ cat ~manager/.bash_history
```

Compression tools and formats

- lrzip, lbzip, gzip, bzip2, 7-zip

- Commonly used data formats in bioinformatics are gzip and bzip2.
  - gzip is faster and bzip compress more efficiently.
  - gzip receives a parameter from 1 to 9 for a compression that can be faster to more efficient.

```
# create a file with random data
$ head -10 /dev/urandom | hexdump > file.txt
$ cat file.txt

# compress with bzip2 and gzip
$ gzip -6 file.txt
$ bzip2 file.txt

# uncompress the files
$ cd /home/manager/Documents/AAB2
$ unzip datasets.zip
$ bunzip2 file.txt.bz2
$ gunzip file.txt.gz

# No need to uncompress the files to access its contents
$ zcat file.txt.gz | less
$ bzcat file.txt.bz2 | less
```

Bundle files in directories

- tar ("tape archive") is a linux tool that allows to bundle files and directories maintaining the respective structure. The resulting file is called the *tar ball*.
- tar can be used as a stand-alone tool.

```
# create a tar ball file
$ tar -cf test.tar file1 file2 dirA/

# extract contents from a tar ball
$ tar -xvf test.tar
# options
# c: creates a new tar file
# f: file name given to archive
# x: extract tar
# v: verbose output
```

- tar can be combined with gzip to create a compressed tar file.

```
# create compressed tar
$ tar cvzf test.tar.gz testFolder/
# or
$ tar cvzt test.tgz testFolder/
# option z: for compression

# Uncompress tar.gz archive file
$ tar -xvf test.gz
```

Symbolic Links

- Symbolic links create references to files and folders and allow this data to be accessed without duplication.
- To create symbolic links mode to the folder where you want to create the link and the use the *ln* command:

```
# move (cd) to folder where you want the link
$ ln –s fullPathToFileOrFolderToLink
```

- Links can be listed with *ls* as ordinary files;
- To delete symbolic links use *unlink*;

Other useful commands

- Regular expressions allow to target multiple files that follow in their names a specific pattern.
  - * Symbol matches on the respective position any string of variable length. e.g. s* 🖧 sun, Saturday, s1, …
  - **?** Symbol matches on the respective position one character. e.g. c?t 🖧 cut, cat, …
  - **[]** on the respective position matches all the characters inside the brackets. e.g. c[a,u] 🖧 cat, cut, …

```
# wildcards match everything
$ ls *.fastq

# check the storage used in the different available disks
$ df -h

# size of current directories
$ du -sh *
# option s: provides a human readable summary of the currents
directories

# size of all directories
$ du -h
```

Commands to improve productivity

```
# download data in the command line
$ wget url1; url2; url3; …
$ wget https://www.dropbox.com/s/db5nzrd2rn424g2/datasets.zip?dl=0

# hidden configuration file of the bash ~/.bashrc

# aliases allow to create shortcuts for different commands
# edit .bashrc and add some alias
alias ll='ls -lh'
alias la='ls -la'
# ask before removing or overwriting files
alias mv='mv -i'
alias cp='cp -i'
alias rm='rm -i'
alias dirsize='du -sh */'

# finding files
$ find -name test.sam
# find needs some options to work, including
# -name: search for the file name
# -type: search for the type: (f)ile, (d)irectory, (l)ink

# finds recursively in the current directory all fastq (fq) files
find . -name "*.fq"
```

Command evaluation in bash

- In the bash environment backticks `` have a special meaning. Everything typed between backticks is evaluated (executed) by the shell.

```
# the result of a command can be directly stored in a variable
$ getls=`ls -la`
$ echo $getls
```

- Bash scripting is a powerful tool to efficiently apply the same operation to multiple files.

```
# list all the fastq files and get the first one hundred lines;
create a subset file for each file

for file in `ls *.fq`;
do
    echo $file
done

for file in `ls *.fq`;
do
    cat $file | head -n 100 > $file".subset"
done
```

Chaining Command line tools

- A powerful feature of unix environment is to allow building data work flows by naturally integrating multiple tools;
- **Pipes** allow to combine different tools by chaining their input and output, i.e. the output of a program/tool is the input of another program.

Main elements:

- *stdin*: input channel - what is read by the program
- *stdout*: channel used to output the results
- *stderr*: channel used for error reporting

- >: indicates that the data is redirected to the following channel or file
- >>: appends the data to the following channel or file
- <: defines the input as the channel or file

```
$ echo "Hello" > text.txt
$ echo "World" >> text.txt
$ cat < text.txt
$ ls -la > tt.txt
$ less < tt.txt
# in alternative the last 2 commands can be combined in one
$ ls -la | less
# here the stdout of ls is redirected to stdin of less
```

Unix Power Tools

- Unix/Linux has an extensive and very efficient set of tools for text analysis. They handle very efficiently large text files which makes them very well suited for bioinformatics analysis.

Tools:
Reporting : wc
Extraction and filtering: head, tail, grep, uniq, awk, cut
Manipulation: dos2unix, sort, tr, sed
Comparison: diff

Counting and translating

wc is a general tool to count lines, words and characters

Options:

- c: number of characters
- w: number of words
- l: number lines

```
# unzip file in GenomeAnnotation folder
$ unzip genes.gtf.zip
# number of lines in a file
$ wc -l genes.gtf
# number of lines that match TP53
$ grep TP53 genes.gtf | wc -l
```

tr replaces characters

Syntax:

tr 'find' 'replace'

```
# number of lines in a file
$ echo "toupper" | tr '[a-z]' '[A-Z]'

# tr can also be used to delete characters: tr -d 'del'
$ tr -d 'chr' < input.txt > out.txt
```

filtering

grep extracts lines that match a string. If matched a string is outputted to *stdout*.

Syntax:
$ grep [options] string [file(s)]
Selected options:

- -i: ignore case
- -v: invert match
- -l: list only the files that contain the match
- --color: highlights the match
- -f: the patterns to be matched are listed in the input file

grep only handles fixed patterns. **egrep** allows to search for regular expressions.

```
# all lines that contain TP53
$ grep "TP53" genes.bed

# number of lines that match chr1, chr2, chr3
$ egrep "chr[1-3]\t" genes.gtf | wc -l
```

filtering

cut cuts out selected portions of each line of a file. Typically defined by the fields in the line.

Syntax:

cut [-d delim] -f <fields> [file(s)]

Options:

delim: field delimeter

fields: index number of the fields to be extracted

```
# extract gene names and strand
$ cut -d \t -f4,6 gene.regions.bed
# get the coordinates of genes
$ cut -f 1-3 gene.regions.bed
# extract the first three characters of each line
$ cut -c 1-3 gene.regions.bed
```

Selecting and getting unique elements

head displays the first lines of a file.
tail displays the last part of a file.
Selected Options:
-n : the number of lines to be extracted

```
# extract the first 10 lines
head -n 10 gene.regions.bed
#or
head -10 gene.regions.bed
# extract line at position 100
head -100 gene.regions.bed | tail -1
```

uniq reports or filters out repeated lines in a file.
Syntax:
uniq [options] [files(s)]
-c: precede each output line with the count of the number of times the line occurred in the input

```
# how many different chromossomes
cut -f1 gene.regions.bed | sort | uniq | wc -l
# count number of repeated occurrences per item
cut -f1 gene.regions.bed | sort | uniq -c
```

sorting

sort sorts lines of text files.

Syntax:

sort [options] [file(s)]

Selected Options:

-r: reverse the result of comparisons

-g: compare according to general numerical value

-t: define the field separator

-k: start and end index field

```
# sorts file first by chromossome then by start position and then
by end position
$ sort -k1,1 -k2,2g -k3,3g gene.regions.bed | less

# sort by gene names alphabetically in reverse order
$ sort -r -k4 gene.regions.bed
```

Differences and OS conversion

diff compares files line by line.
Syntax:
diff [options] [files(s)]

```
# create two similar files
$ head -10 gene.regions.bed > h1.txt
$ head -10 gene.regions.bed > h2.txt
# make some changes in the first
$ sed -i's/chr22/chrX/' h1.txt
# compare them line by line
$ diff h1.txt h2.txt
```

dos2unix is DOS/MAC to UNIX text file format converter. Specially important if files are created in different operating systems.
Syntax:
dos2unix [options] [-c convmode] [-n infile outfile]

Differences and OS conversion

dos2unix is not a native command from linux. To have it ready in your machine:

sudo apt-get install tofrodos

tofrodos == to and from dos

```
sudo ln –s /usr/bin/fromdos /usr/bin/dos2unix
sudo ln –s /usr/bin/todos /usr/bin/unix2dos
```

or in alternative the function of this tool can be achieved with:

```
tr –d '\r' file.txt > file.converted.txt

# or to generate repercute the changes in the original file
tr –d '\r' < file.txt > t
mv t file.txt
```

Stream editor

sed is a stream editor. It reads from the input channel (file or *stdin*) modifies the input as specified by a list of commands and then writes to standard output. Has been used in the development of programming languages.
sed is a very powerful and complete tool, we will only look into the substitute function.

General Sed syntax:
sed [options] [script] [inputfile]

Selected Options:

-e: sets the following commands to be run while processing the input

-f: defines the script-file where the commands to be executed are listed

-i: specifies that the changes are done directly in the input file and not send to stdout.

Stream editor

Substitution command syntax:

sed -e 's/r1/s1/' [file(s)]

- s: indicates the substitution operation
- r1: the regular expression to be replaced
- s1: text that will replace the regex match

```
# convert to upper chromossomes
$ sed –e 's/chr/CHR/' gene.regions.bed
# Note that sed will only replace the first match. option g is
used to do a global match
# remove the chr prefix
$ sed 's/chr//g' gene.regions.bed
# forcing matches at the start of the line
$ sed 's/^chr//' gene.regions.bed
# Trim whitespaces and tabulations at start and end of file
$ sed 's/^[ \t]*//;s/[ \t]*$//' file.txt
```

```
sed can also be used to delete files
# delete blank line
$ sed '/^$/d' file.txt
# delete the fourth line
$ sed '4d' h1.txt
# delete the first to the third line
$ sed '1,3d' h1.txt
```

Pattern Matching

awk is pattern-directed scanning and processing language. It is a high-level and flexible programming language.

awk command line Syntax:
awk [-F fs] [-v var=value] [-f progfile] [file]

-F: field separator
-v: parameters value to be passed to the awk script
-f: name of the awk script
file: input files.

Awk is a based on 'pattern ==> { action }' paradigm.

- It operates on a line-by-line basis.
- It can be used to do calculations and data manipulation.
- It has a structure: BEGIN{}{}END{}.
- It can be run in a file with the command awk -f yourscript.awk or in the command line using quotes as awk '{print $0}'.

Pattern Matching

- It operates on the concept of records (rows) and fields (columns). Both the input and output markers for records and fields can be defined by the user in the code of the program, typically in the command line or the BEGIN section.
    - FS: Field Separator, e.g. FS=";"
    - OFS: Output Field Separator, e.g. OFS=","
    - RS: Record Separator, e.g. RS="\t"
    - ORS: Output Record Separator, e.g. RS="\n”

    - NF and NR: number of records and fields
    - $x: indicates the field number
    - $0: entire line
    - $1: field number 1
    - $NF: last field

Pattern Matching

Awk as a filtering tool.

Syntax:

awk -F delim '/pattern/{print}'

Options:

delim: field delimeter

pattern: regular expression with pattern to be matched

The part of the code in brackets is executed if the line has a match with the regular expression match.

Awk Examples

```
# extract the gene coordinates in fields 1, 4 and 5
awk '{print "chr"$1,$4,$5}' genes.gtf

# print lines with exact match to the first field
awk '{if($1 == "chr1"){print}}' gene.regions.bed

# print lines that do not have exact match to the first field
awk '{if($1 != "chr22"){print}}' gene.regions.bed

# print lines whose 7th field matches the regular expression
awk '$7  ~ /^[a-f]/' text.txt
# or
awk '{if($7 ~ /^[a-f]/){print $0}}' text.txt

# print lines whose 7th field does not match the regular expression
awk '$7  !~ /^[a-f]/' text.txt

# print genes that have coordinates in a certain chromossomic
region
awk '{if($1=="chr1" && $2 > 100000 && $3< 200000){print $0}}'
gene.regions.bed

# print genes that are either chr1 or chr3
awk '{if($1=="chr1" || $1=="chr3"){print $0}}' gene.regions.bed
```

Awk Examples

```
# Sum of all elements of field 1
awk '{sum+=$1} END {print sum}' file.txt

# Compute the mean of field 2
awk '{x+=$2}END{print x/NR}' file.txt

# Compute the mean of field 2 with an initial value
awk 'BEGIN{x=100}{x+=$2}END{print x/NR}' file.txt

# convert gtf to bed format
awk '{print "chr"$1"\t"$4"\t"$5"\t"$10"\t"$7}' genes.gtf  | sed
's/"//' | sed 's/";//' > genes.bed

# output the list of unique chromosome ids
awk '{chrlist[$1]=1;}END{for(c in chrlist){print c}}' gene.regions.bed

# count the number of occurrences of each chromosome
awk '{chrlist[$1]+=1;}END{for(c in chrlist){print c"\t"chrlist[c]}}'
gene.regions.bed
```

Exercises

1) Calculate the sum of column 2 and 3 and put it at the end of a row (GeneExpression/GeneExpression.tab)

2) Calculate the sum of column 2 and 3 and put it at the end of the file (GeneExpression/GeneExpression.tab)

3) Output sequence name and its length for every sequence within a fasta file (Motifs/lex.dna.fasta)

4) Sort genes by their increasing length. Output gene name and respective length (gene.regions.bed)

https://drive.google.com/file/d/1jzddejVPOeW08dLnIeIXoI9W3WUjnIp2/view?usp=sharing