Introduction to Mac and Linux Command Line for Bioinformatics

INBRE/COBRE Joint Bioinformatics core

Hawaii State
INBRE II

IDeA Network
of Biomedical Research
Excellence

Pacific Center for Emerging
Infectious Diseases
Research

Introduction to the Mac and Linux Command Line

INBRE/COBRE Bioinformatics Workshops Series

Mahdi Belcaid[1]

[1] INBRE/COBRE Joint Bioinformatics Core

December 7, 2010

Notes

Notes

BLAST thrargh command line for 454 data

Control: More ways than through the GUI

Speed: Certain commands require many GUI step to do equivalent

Availability of applications: Bioinfo tools and powerful system applications

Because (some) bioinformaticians are Lazy!

Notes

Why Mac and Linux?

- Mac is Unix based and Linux is Unix-like
  - They are both "Unix" compatible and possess a comparable set of core system-applications
- A wide range of applications available for both
- Platform of choice for bioinformaticians
  - More and more biologists embracing it (Mac more than Linux)
- There are ways to emulate Linux on Windows
  - Easier to just run Linux from a Virtual Machine

Notes

What You Should Know

- During this workshop, we will interact with the OS only through the Command Line Interface (CLI)
- The CLI is interactive, i.e. takes commands, executes specific tasks and returns (or not) an output
  - Each command accomplishes a specific task
- Case-sensitive: Remember that commands, files, folders, behaviors, . . . , are all case sensitive

Notes

Tips for a "Happy Interaction"

- History: The shell keeps a history of a set of typed commands, uses arrow to access them or to type **history** to see the full list
- Proprietary formats: Avoid working with annotated files. (Ex: Word, Excel, pdf, etc...)
- Avoid Spaces: Rename any files/folder that contains spaces in its name.
- To view the contents of a file, use the **less** program

```
less <file_name>
```

  - To quit the **less** viewer, type "**q**"
- Aborting commands: You can press **Ctrl+c** to stop a running command

Notes

tab button fills in command

"history"

I.E.[1] Some Shortcuts

- Command completion: Use tab to complete a partially typed command, a file or a directory name
- **Ctrl+A** Takes you back to the beginning of the line
- **Ctrl+E** Takes you back to the End of the line
- **Ctrl+delete** delete one word back
- **Ctrl+→** jump one word to the right
- **Ctrl+←** jump one word to the left
- To get help, use "`-help`" option or **man** command
  - type "**q**" to exit **man** or **less**
- More useful commands on the cheat sheet

[1] I.E. Interactive Example

Notes

man first for assistance, then help

Unix Command Line Structure

- Typical structure[2]:

```
command [options] [arguments]
```

- Options alter the behavior of the command
  - ...and in most cases, the output
- Arguments are the operands of the command
  Example[3]:

```
mahdi@laptop$ grep -i ACGT myInutFile.fasta
```

- Command line structure across Unix-like systems is similar

[2] Code in red boxes is not executable
[3] Code in green boxes is executable

Notes

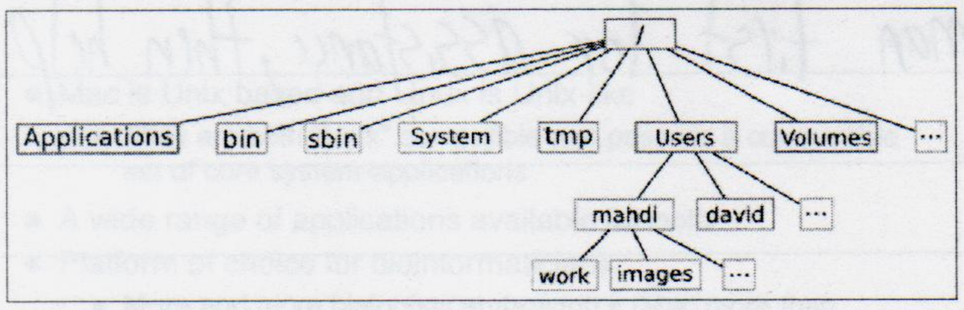
Unix-like File System Hierarchy

- One of the major differences between Windows and Linux/OS X
- Unix-like systems follow a "Quasi-standardized" hierarchical structure
- The root of the hierarchy is: /
  - Similar to **C:** or **D:** under Windows
- All files, folders, applications, drives, etc..., are mounted (placed by the OS) under /

```
Windows path    C:\Folder\subfolder\file.txt
Unix-like path  /Folder/subfolder/file.txt
```


Notes

OS X File System Structure



/Applications	User apps
/bin	Essential programs (date, ls, ...)
/sbin	Essential system admin utils
/System	Critical OS files (...)
/tmp	Temporary files, caches, ...
/Users	All user homes on the machine
/Volumes	Mounted devices & volumes (hard disks, DVD's, DMG mounts, ...)
...	Other directory partitions (/dev, /etc, /usr, ...)

Notes

bin = binary binary = program
sbin = admin, system binary

tmp: deletes on restart

Gathering-Information

- List the content of a directory:

```
ls [directory_path]
```

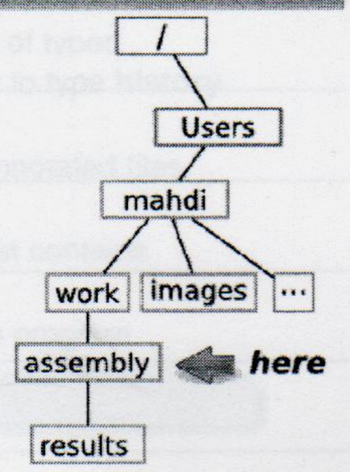
```
mahdi@laptop$ ls /home/mahdi
 Applications  Dropbox  Music  Photos
 work
```

- To know where you are, use the **pwd** command (**p**rint **w**orking **d**irectory)

```
mahdi@laptop$ pwd
/Users/mahdi/
```


Notes

Navigating the Hierarchy



- You are in:
  `/Users/mahdi/work/assembly`

Notes

Navigating the Hierarchy: *cd*

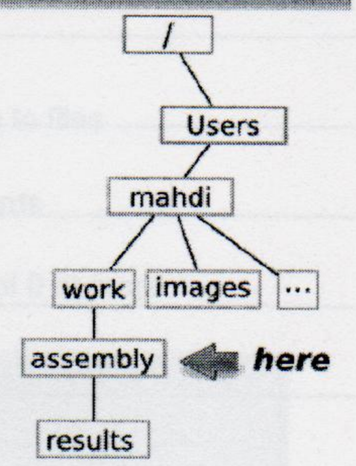
- Changing directory: Going up and down the hierarchy

```
cd [directory_path]
```

- directory_path: The *absolute* or *relative* path
  1. *Absolute path*: Starts at the root and spell full path
  2. *Relative path*: with regards to your current location use .. to specify a parent directory

Notes

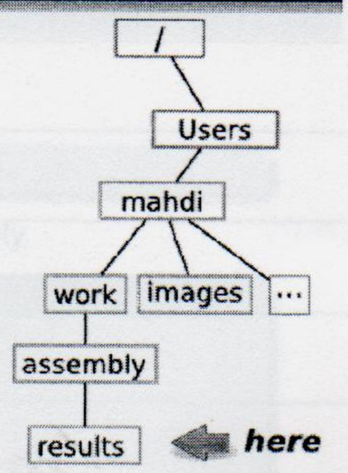
Navigating the Hierarchy: Going-down



- You are in:
  `/Users/mahdi/work/assembly`

Notes

Navigating the Hierarchy: Going-down



- You are in:
  `/Users/mahdi/work/assembly`
- You need to go into the directory `results`, located in `assembly`

Notes

```
mahdi@laptop$ pwd
/Users/mahdi/work/assembly
```

- Using the *absolute* path:

```
....$ cd /Users/mahdi/work/assembly/results
```

- Using the *relative* path:

```
mahdi@laptop$ cd results
```


Notes

- You are in:
  `/Users/mahdi/work/assembly/results`

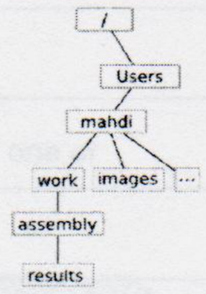
Notes

- You are in:
  `/Users/mahdi/work/assembly/results`
- To go into the directory
  `/Users/mahdi` (user's home directory)

Notes

Going-Up: Example

```
mahdi@laptop$ pwd
/Users/mahdi/work/assembly/results
```



- Using the *absolute* path:

```
mahdi@laptop$ cd /Users/mahdi
```

- Using *relative* path:

```
mahdi@laptop$ cd ../../../
```

Or use a shortcut

```
mahdi@laptop$ cd
```


Notes

CLI wildcards: *

- They are used by the CLI when referring to files
  - These are not regular expressions!
- There are various command line wild-cards
  - We will only look at: *
- * is interpreter by the CLI as any string of 0 or more characters.

```
file*   Any file name that starts with the prefix "file".
        Ex:  file, file128, file_A, ...
*.txt   Any file that ends with the entension ".txt"
```


wildcards

Notes

*

file names started with . are invisible

Creating or Deleting Directories
mkdir and **rmdir**

- You can create or delete directory with:

```
mkdir dir_name rmdir dir_name
```

- For **rmdir**, the directory needs to be empty.

Example

```
1 mahdi@laptop$ mkdir /Users/mahdi/work/assembly/results/tempDir1
2 mahdi@laptop$ mkdir /Users/mahdi/work/assembly/results/tempDir1/tempDir2
3 mahdi@laptop$ rmdir /Users/mahdi/work/assembly/results/tempDir1
4 rmdir:  failed to remove
  '/Users/mahdi/work/assembly/results/tempDir1':  Directory
  not empty
```


Notes

rmdir → file needs to be empty

Copying Files: **cp**

- Done through the **cp** command:

```
cp file1 file2
```

```
cp file1 file2 file3 ...filen dir
```

```
cp -r dir1 dir2
```

- Paths of arguments can be relative or absolute

Notes

-r → recussive, needed for copying directory w/ contents

mv command used to rename files

Deleting Files

- Done through the **rm** command:

```
rm file1 file2 file3 ...filen
```

- You can remove directories with **rm**
  - Consult the **man** page of **rm**

```
mahdi@laptop$ rm -r dirPath
```

- Be extra careful with **rm**... Especially when using it in **-r** (recursive) mode or with the "*" wild-card
  - No way to undo **rm**

Notes

Exercise 1

- Download the following zip archive:
  `http://dl.dropbox.com/u/28270/work.zip`
- Move the work.zip file to your home directory
  - Example in my case it is: `/Users/mahdi/`
- unzip the work.zip archive

```
mahdi@laptop$ unzip work.zip
```

  - This will create a new directory `work` in your home
- Create a directory in `work` called `exercise_1`
- Copy all the fasta files (.fasta) in `work/temp` to `exercise_1`
- Delete the directory `temp`

Notes

Counting File Characteristics: *wc*

- Generates the word, line and character counts for one or more inputs
- Three useful options:

```
wc -l [file1, ..., filen]   #counts the number of lines
wc -m [file1, ..., filen ]  #counts the number of characters
wc -w [file1, ..., filen ]  #counts the number of words
```


Notes

Sorting a File: *sort*

- The command **sort** sorts the content of a file passed as a parameter
- Default behavior is to sort alphabetically, in ascending order
- Most used options are:

```
-n  Sort using numerical order
-r  Sort in reverse(descending) order
```


Notes

commonly used

Removing Redundant Lines: *uniq*

- Deletes the duplicated lines in a file
- File needs to be sorted
- Most used options are

```
-c  prefix the unique lines by the number of occurrences
-d  print only the repeated lines
```


Notes

uniq tells you one of each thing you have

Output Redirection: >

- The ">" is used to redirect **standard output** (what gets printed to the screen)
- You can send the output to another program (**pipe**), or **redirect** it to a file
- Redirection is selected through the ">" character

```
program [option] [arguments] > results_file.txt
```

```
mahdi@laptop$ ls /Users/mahdi/work/viruses > listing.txt
```

- `listing.txt` contains all the names of files and directories located in `/Users/mahdi/work/viruses`

Notes

Unix Pipes: |

- A pipe is a way of chaining two or more commands in Unix
  - The standard output of the first command becomes the input of the next command
  - The output is put in a "virtual" files for the subsequent program to use
- Example: Since **uniq** requires sorted files, we can easily pipe the output of **sort** into **uniq**

```
mahdi@laptop$ sort genes.ids | uniq
```

- There is no limit on the number of pipes you can do

Notes

I.E. Using Pipes

- Counting the number of unique blast hits:

`/Users/mahdi/work/blasts/contigs_replicase_all.csv`
contains the blast results of a contigs file against **nt**

`/Users/mahdi/work/blasts/genes.ids`
contains all the hits from the blasts files (8*th* column)

```
mahdi@laptop$ sort genes.ids | uniq | wc -l
```

This counts the number of unique ids in the files genes.ids

Notes

Exercise 2

- How many fasta files are there in the exercise_1 directory you created earlier?
- Print a sorted listing of of the `assembly/libraries` directory into a file named "`temp`"
  - Make sure the file is sorted alphabetically
- How many distinct libraries are in `assembly/library/data`

ls wc -l

→ sort ./libraries/* > temp.txt

Notes

Inclusion, Exclusion Comparison: **comm**

- **comm** is a way of comparing 2 files
  - Same logic as a Venn diagram
- The syntax of the program is:

```
comm [-1, -2, -3] file_1 file_2
 -1  Supresses lines unique to file_1
 -2  Supresses lines unique to file_2
 -3  Supresse lines common to both files
```

Notes

I.E. Comparing two files

- Directory `/Users/mahdi/work/examples/comm` contains two files
  - file_1 and file_2

```
mahdi@laptop$ comm -13 file_1 file_2
```

Prints lines unique to file 2

```
mahdi@laptop$ comm -3 file_1 file_2
```

Prints lines that are not shared by file_1 and file_2

Notes

Exercise 3

- Compare the singletons rate between two alternative assemblies `assembly1_singletons.ids` and `assembly2_singletons.ids` in the `results` folder
  - How many ids are singletons in assembly1_singletons.ids?
  - How many ids are considered singletons in both assemblies?

Notes

What is a Regular Expression?

- "A regular expression, abbreviated as regex or regexp, is a string that describes or matches a set of strings, according to certain syntax rules." *Wikipedia*
- A regex is a pattern used to give a concise description of a set, without listing all its elements
- An exact string or one literal character is a trivial example of a regular expression
  - Describes exactly one element
  - Example: `A, or ACGT, etc...`

Notes

How Does a Regular Expression Work?

- Uses meta-characters, meta-symbols to better describe a pattern
- Each meta-(character|symbol) has a special meaning
- Meaning is expanded during search phase to represent a class
- Suppose we need to search for exact occurrences of poly-tails in a file
  - We start with `A`

Notes

Using Anchors

- To enforce matching to either the beginning or the end of a string, anchors are used:

```
^   Forces matching at beginning of line
$   Forces matching at end of line
```

```
^A  Searches for all the lines that start with A
A$  Searches for all the lines that end with A
```

- However, we know that a poly-As contain more that one A.
  - How do we define the multiplicity of a character?

Notes

Using Quantifiers

- The minimum, as well as the maximum number of matches can be set using quantifiers
- Quantifiers apply to the object they follow
- Quantifiers can be explicit:

```
A{5}       find exact occurrences of AAAAA (5 As)
A{5, 10}   Minimum 5 consecutive As, maximum 10
A{5, }     5 consecutive As with no maximum
```

- Or they can be implicit (using a meta-character)

```
A*  match A, 0 or more times
A+  match A, 1 or more times.  Example:  A, AA, AAA
    equivalent to A{1,}
A?  match A, 0 or one time
```


Notes

Using Alternation

- We know that poly-tails can be comprised of `T`s as well
  - How can we define alternative patterns?
- To describe alternative options to match, use the | meta-character

```
A|T              match an A or a T
(A{5,}|T{5,}     match 5 consecutive As or Ts
```

or options

- The regular expression engine tries to match any occurrence of at least 5 consecutive `A`s with no maximum
- If it fails, it tries to match any occurrence of at least 5 consecutive `A`s with no maximum

Notes

Other Useful Meta-Characters: "."

- The . represents any character
  - Depending on the OS, the context, the applications, this previous statement might be false
  - It is safe to assume that is any alphanumeric character + space can be represented by .
- Considering the DNA alphabet

```
A.T will match AAT, ACT, AGT, ATT, or A T
```


Notes

Other Useful Meta-Characters:
Character Classes

- [] are used to represent a class of characters
- Match one character of the class

```
[ACGT] represent either A or C or G or T
```

```
[A-M] represent either, A, B, ..., through M
```

```
[0-9] represent any one digit character
```

- "." and "[]" can be combined with any of the previous quantifiers

```
[0-9]+ a number of one or more digits
```


Notes

Quick Review

- To describe a pattern, you can use a combination of the meta-(characters|symbols)
- You need at least one literal, or a character class, such as "." or "[0-9]"
- Use quantification to define multiplicity of the occurrence
- Separate patterns with alternation symbol "|"

Examples

```
AGT*    Matches AG, AGT, AGTT, AGTTT ...
N{4,}   Matches any occurrence of four
        or more consecutive Ns
^>.+    Matches id lines from a fasta files
```


Notes

0 or more times

FASTA start with these

Starts with > with one or more characters

grep: An Intro

- Very powerful utility program used to search input for lines matching a given pattern
- The syntax of **grep** is the following:

```
grep [OPTIONS] PATTERN [File_1, ..., File_n]
```

- To find all the lines containing the pattern ACGT in the file `sequences.fasta`

```
mahdi@laptop$ grep ACGT sequences.fasta
ATGCTAGTGATCGTGTAGCTGTGAACGT
ACTGTGTAGACGTATGTGTAGTCGTGAT
```

- Patterns in **grep** are case sensitive

Notes

grep — kind of like find

More with grep

- The pattern used in **grep** can be a regular expression
- Use the -E option for more flexibility

```
mahdi@laptop$ grep -E ">.+" input.fasta

>tempFT13I length=103 xy=2275_2188 region=3 run=R_2010_09_22_21_17_58
>tempHH7RK length=74 xy=2960_3134 region=3 run=R_2010_09_22_21_17_58
>tempGB02L length=178 xy=2856_0539 region=3 run=R_2010_09_22_21_17_58
>tempF5IB1 length=93 xy=2409_1839 region=3 run=R_2010_09_22_21_17_58
...
```


Notes

grep ">" Kbay.fna | wc -l

grep -c ">" Kbay.fna

often used to count # of sequences

Useful grep Options

- Some of the most used options

```
-c    count the number of occurrences of the pattern
-i    ignore case of the pattern
-n    print the line number containing hit
-v    print lines that DO NOT contain pattern
-AN   print N lines after the pattern
-BN   print N lines before the pattern
```

- Many other options...

I.E. Counting Sequences in Fasta Files

- Count the number of sequences in a fasta file

```
mahdi@laptop$ grep -c -E "^>.+" reads.fasta
331
```


Notes

ls | grep -i "marine"

I.E. **grep** in a pipe chain

- Find all the files containing a specific pattern
  - The results of the first program becomes the input of the second

```
mahdi@laptop$ ls | grep -i marine
Marine_birnavirus
Marine_RNA_virus_SOG
Unknown_marine_virus
```

- Lists all the files in the current directory and runs **grep** on the output to find which files contain the case-insensitive word marines

Notes

Exercise 4
Comparing Results From Two Assemblies

- ***newbler*** generates, among other results, `/Users/mahdi/work/assembly/results/454ReadStatus.txt`
- Two columns of interest in the file are:
  1. First col. is the read id
  2. Second col. is the status of the read (Assembled, Singleton, etc...)
- You want to:
  1. Know what are the distinct statuses (Col. 2) that are in your results file
  2. Count the number of reads, in `454ReadStatus.txt` that are "TooShort"
  3. Extract the reads that have been sequenced on the 9/25, 9/24 or 9/23
     - `run=R_2010_09_25_2X`
  4. How many sequences were NOT run on 09/22?

Notes

grep -v "0f.22" reads.fasta |

grep -c ">"

sed: An Introduction

find & replace

- **sed** (Stream editor) is used to perform a set of editing operations on a file
  - Reads the input, one line at a time, and executes the edit operation specified
  - We will only be looking at **sed** for making substitutions
  - Substitute a pattern occurrence with a string
- By default the modified file is printed to the standard output (Screen)

Notes

sed Basic Format

- The basic command line structure is:

```
sed [options] 's/PATTERN/STRING/[g]' file
```

Example

```
mahdi@laptop$ sed 's/ACGGCA/TGCCGT/'
sequences.fasta
```

1. Replaces the first occurrence of 'ACGGCA' (if any) on each line, with the string 'TGCCGT'
2. Outputs the contents of `sequences.fasta` to `stdout`

Notes

Useful sed Options

- By default, only the first string is replaced
  - You can change the behavior by running **sed** in greedy mode

g = global

```
mahdi@laptop$ sed 's/ACGGCA/TGCCGT/g'
sequences.fasta
```

- Also, you can instruct **sed** to make the changes directly to the input file

```
mahdi@laptop$ sed -i .backup 's/ACGGCA/TGCCGT/g'
sequences.fasta
```

- **sed** make the modification directly to sequences.fasta, but...
- Creates a backup of the original file with the ".backup" extension

Notes

with no g, it will replace only the first instance per line

s/abc//g ← replaces abc w/nothing

Regular Expressions With **sed**

- **sed**'s strength comes from the fact that you can use it with regular expressions
- To instruct **sed** to expect an extended regular expression, the "**-E**" option is used
  - or "**-r**" under Linux

```
...$ sed 's/\.[0-9]+//g' accession.fasta >
accession_no_version.fasta
```

- Remove the trailing version number for a NCBI accession.
  - Replaced with empty string "//"
  - We need to escape wild-cards chars to give them their literal meaning

used to get duplicate accession numbers to have same tag

Notes

Exercise 5

- In the file:
  `/Users/mahdi/work/blasts/genes.ids`
- Remove the dd identifier in the beginning of the name. (`gb|, dbj|, ...`)
- Remove the version from NCBI accessions names
- Remove spaces at the beginning or end of line

ref| sp| embl

Notes

this character makes you take literal meaning of bar

```
sed -E 's/\.[0-9]+//' genes.ids
sed -E 's/\.[0-9]+//' genes.ids
```


Working with **awk**

- **awk** is an excellent tool for working with column delimited files
  - Ex: Comma delimited, tab delimited, etc...
- Complete, but often complicated, reporting language
- We will only look at some of the useful functionalities
- Basic command-line structure:

```
awk [options] 'program text...program text' input_file
```


Notes

How **awk** Addresses Columns

- By default, **awk** splits columns based on * (one or more spaces)
- Behavior can be changed in the program text
- Each column is assigned an "automatic" variable name
  1. Column 1 is assigned variable name $1
  2. Column 2 is assigned variable name $2
  3. Column n is assigned variable name $n
  4. The whole line is assigned variable name $0
- In the `program text`, to print a column, use the following syntax:

```
print column_a"SEPARATOR"column_n,...
```


Notes

Example: Extracting Columns from a Tab Delimited Blast Report

- Extract, for each input, the accession and description

```
mahdi@laptop$ awk '{print $1"\"$8}'
contigs_viruses_all.csv
```

- Prints the sequence id, followed by a tab, followed by the hits accession
- How do we capture the description (Col 9), which contains spaces?

```
mahdi@laptop$ awk '{print $9}' contigs_viruses_all.csv
Marine
Rhizosolenia
...
```


Notes

Changing the Field Separator

- You can change the field separator using the -F option
- To avoid confusion, double quote the separator:

```
mahdi@laptop$ awk -F ":" {print $1"\"$3}'
```

- **awk** can be applied to the output of another program
  - You can pipe into **awk**

Notes

I.E. Tying **awk** with Other Programs

- Extracting gene information from NCBI genbank file `/Users/mahdi/work/proteins/protein.gbk`
- Extract gene name from texttt/gene="gene name"
- Ignore `/gene_synonym`
- Remove formatting (= and ")

awk -F"\t" ⇒ tab as delimitor

Notes

Conditionals in **awk**

- We will make a slight exception to our "No Programming" rule
- We can code conditionals in awk

```
awk '{ if (<condition>) {execute this block}
else{ execute this other block} }' input_file
```


Notes

I.E. Extracting Specific Hits from a Blast Report

- Conditionals are useful if you want to extract a subset to the data
  - Split a file into different files
- Example: to extract all the hits that are above 1e-05

```
awk '{if($3>1e-05){print $1,$8}}'
contigs_viruses_all.csv
```


Notes

if statement e must have number before it otherwise it thinks that it is a string.

Blast Databases

- Blast requires specifically formatted database to run
- **nt**, **nr**, **envxx** are available, preformatted, from the NCBI ftp site
- For this example, we will format our own database:
- The basic syntax for formatting a blastdb is:

```
formatdb -i <fasta_inut> -p <F|T> -o
 -i <input>   Input file to be formatted (Mostly Fasta)
 -p F|T       The file type.  protein(T) or nucleotide(F)
 -o           Created indexes for fastacmd
```


Notes

Running Blast from the Command Line

- The basic Blast command line is:

```
blastall -p <program_name> -i <input_file> -d <database>
-o output_name [other options]
```

- Blast has many params that are not available on public web interfaces
- For more details, see **man** page or type:

```
mahdi@laptop$ blastall -help
```


Notes

Some Blast Parameters

- There is no consensus on the most used blast options
  - Depends on the task
- The following params are very useful

```
-m INT    changes the output type (-m 8 for table)
-e eval   set the e-value threshold
          (usually 1e-05 for nucleotide blast.)
-F F|T    filer sequence for low complexity
          (T by default)
-W INT    changes the word size, for small input sequence
```


Notes

Parsing Sequences from a Blastdb

- **fastacmd** is a program from the blastall suite
- Used to extract sequences from a blast formatted database
  - Extract one or more sequences from the DB
- Syntax of the program is:

```
fastacmd -d <database> -s <ids> [-o output_file]
 -s  comma delimited list of ids to extract
 -i  instead of -s, you can provide a file
     containing a list of ids
```


Notes

I.E. Working with Blast

- Formatting the `/Users/mahdi/work/blasts/blastdb/kbay.fas` file
- Blasting `test.fas` against `kbay.fas` in tabular format
- Extracting hit ids with evalues > 1e-50
- Extracting the ids form the original fasta file using fastacmd

Notes

Exercise 6

- Use the blast report:
  `/Users/mahdi/work/blasts/contigs_viruses_all.csv`
- Extract all the ids whose hit description contains the keyword *Rhizosolenia*
  - Make sure you have no redundancies
- Extract the corresponding sequences using **fastacmd**
  - The original sequences are in `/Users/mahdi/work/blasts/blastdb/kbay.fas`
  - Remember you have to **formatdb** before running **fastacmd**
- Once you have your input, blast it (**blastn**) against the already formatted db:
  `/Users/mahdi/work/blasts/blastdb/marine/marine.fas`
  - Generate a table-formatted output (`-m 8`)

Notes

Feedback Time

- Any thoughts?
- Suggestions for next workshops?
- Subjects suggestions?

Notes

Notes

Notes

Introduction to Command-Line for Bioinformatics Cheat Sheet

Some section

CLI Keyboard Shortcuts

Ctrl+a	Go to beginning of line
Ctrl+e	Go to end of line
Alt+→	Go to the right one word
Alt+←	Go to the left one word
Ctrl+k	Delete from cursor to end of line
Esc+delete	Delete one word back (backspace)
Esc+d	Delete one word forward (backspace)

Important OS X Directories

/Applications	User applications
/bin	Esential programs (date, ls, ...)
/sbin	Essential system admin utilities
/System	Critical OS files
/tmp	Temporary files, caches,...
/Users	All user homes on the machine

Getting Help

To get help, try man first `man command`
If manual pages are not available for command, try:
`command --help` or `command -h` or

Miscellaneous symbols

>	Redirection symbol
|	Piping symbol

Elementary Commands

`cd [dir_path]`	Change directory
`mkdir name`	Creat new directory
`rmdir <dir>`	Remove empty directory
`cp file1 file2`	Create a copy of file 1
`cp file1 file2`	Copy files 1 through n to dir
`...file_n dir`	
`cp -r dir1 dir2`	Copy directory1 to directory2
`rm file1 file2`	delete files 1 through n
`...file_n`	
`rm -r dir`	Recursively remove directory

Notes

- `../` references one level up the hierarchy
- Paths can be *relative* or *absolute*.
  - Absolute paths starts at the root `/`
  - Relative paths start at the *current working directory*
- `rmdir` requires `dir` to be empty

File Summary Commands

`wc`	Count Characteristics of a file
`-l`	Count lines
`-m`	Count character
`-w`	Count words
_____	Sort a file
`sort`	
`-n`	Use numerical order
`-r`	Use descending order

Regular Expressions

Anchors

X^	Match X at the beginning of a line
X$	Match X at the end of a regex

Quantifiers

Explicit

X{N}	Match X exactly N times
X{N,M}	Match X at least N times No more than M times
X{N, }	Match X at least N times

Implicit

X*	Match X, 0 or more times
X+	Match X, 1 or more times
X+	Match X, 0 or 1 times

Alternation

regex_A|regex_B	Match regex_A or regex_B

Character Classes

$[XYZ]$	Match any one letter between []. (X or Y or Z)
$[A-M]$	Match any one letter in range A to M
$[0-9]$	Match any digit
.	Match any character

Usually, ^ and $ are respectively first and last elements of a

regex

grep, sed and awk

`grep`	Search for a pattern in one or more files
`grep [OPTIONS] PATTERN [File_1, ..., File_n]`	
`-c`	Count the number of occurrences of the pattern
`-i`	Ignore case of the pattern
`-n`	Print the line number containing hit
`-v`	Print lines that DO NOT contain pattern
`-AN`	Print N lines after the pattern
`-BN`	Print N lines before the pattern
_____	
`sed`	Perform editing operations on a file
`sed [options] s/PATTERN/STRING/[g] file`	
`-E`	Give a regex as a pattern (**-r** under linux)
`-i postfix`	backup a file and edit it directly
_____	
`awk`	Parse and process pattern delimited files
awk [options] '*program text*' input_file	
`program text`	Contains the logic for processing your input file Ex. column_a"SEPARATOR"column_n,...
$	Prefix column of the input
$1,$2,$3, ...	Column 1, 2, 3, etc ...
`awk '{ if (<condition>) {execute this block}'`	
`else{execute this other block} }`	
<condition>	Implements a conditional test. The format is: (COL > | < | = | != Value)
block to execute	Another print statement

Blastall Suite

`formatdb`	Formats a database for blastall or fastacmd
`formatdb -i <input file> -p <type> -o`	
`type`	T for protein, F for nucleotides
_____	
`fastacmd`	Extracts fasta sequences from a blast formatted DB
`fastacmd -d <database> -s <ids> [-o output_file]`	
`-s`	Comma delimited list of ids to extract
`-i`	Instead of -s, you can provide a file containing a list of ids
_____	
`blastall`	blasts, using a specific program (blastn, blastx, ...) an input file on a database
`blastall -p <program> -i <input> -d <database>`	
`-o output [other options]`	
`-p`	Program name (blastn, blastx, tblastx, ...)
`-i`	Program name (Fasta input)
`-d`	blast formatted DB
`-o`	The output file
`-m`	Changes the output type. Use -m 8 for table output
`-e`	Set the e-value threshold (usually 1e-05 for nucleotide blast.)
`-F`	(F|T) Filter sequences of low complexity (T by default)
`-W`	changes the word size, for small input sequence

Blast Notes

- Make sure you provide blast program with the appropriate blast db
  - Ex. You cannot use a protein db with blastn

_____
Mahdi Belcaid
mahdi<at>hawaii<dot>edu

history

pwd - print working directory, shows what directory you are at

cd .. goes up a directory