Lydia Becker

EU-US Training in Marine Bioinformatics

Linux Command Line Introduction and BASH Scripting

Christian Quast, Jan Gerken

18. Juni 2012

cp - copy

ls = list

ls - = list in different format

apropos - appropriate commands

- - help

what is - displays a brief descript

-l is long format

1 Command Line Introduction

Book Chapters: 3, 4, 5(, 19 tar)
Directory:

Section 6 of this books explains how to get help for different commands that will be used throughout these exercises.

Searching in Help Man pages as well as the less command have a set of useful controls that can be used to navigate the displayed text. Basic navigation is done using the arraow keys. Pressing `enter` will advance one line, and `space` will skip one page. Forthure usefull shortcuts are:

q Quit.

g Go to the beginning of the file

G Jump to the end of the file

/ followed by TEXT: search for TEXT in the opened file.

n Next search result

N Previous search result.

1.1 Basic Navigation

1. Open a Shell (Konsole).
2. Type `pwd`, what do you see? name of current working directory
3. List the content of the current directory. /people/home/baker
4. List the content of the current directory (long listing format). What is the difference to the previous exercise? gives content of baker, also ~~[illegible]~~ permission
5. List the content of the parent directory without changing into the directory. ~~[illegible]~~ ls /home or ls ../
6. List the content of the top level directory without changing into the directory. ls /home people
7. List the content of the directory `/var/tmp` without changing into the directory. ls /var/tmp
8. Type the commands `cd .`, `cd ..`, `cd ..`, `cd .`
9. Where are you in the file system tree? no A, change back, A bit word
10. Type the command `cd .` again, where are you? same place
11. Explain what the command `cd .` does. is relative path in tree, where the working directory is
12. Type `cd`, where are you now? home
13. Type `cd ~`, where are you now? changes it to ~~[illegible]~~ home directory

14. Type cd ~LOGIN, where are you now (Substitute LOGIN with your user login)? takes me to user directory (how to get to other users no we directory)
15. Type cd tmp What happens and why? changes to temporary directory (tmp)
16. Type cd /tmp What happens and why? nothing finds the directory called tmp
17. Change back to your home directory. cd ~
18. Change into the following directories, list their content, change one directory up, and again list the directories content.
    - /home, /bin, /usr/local, /.
19. Change back to your home directory.

1.2 File Opterations

1. Create new directories with the names: mkdir
    - temp, Sub, sub, subsub, and inside directory sub, a directory called dirname with spaces. if you put a space creates 2 dirs name or with a "
2. List the content of the current directory (long listing format).
3. Does the output looks as expected?
4. Change into directory Sub.
5. Create an empty file called test.txt. touch test.txt
6. Copy the file into the directory temp. cp test.txt ../temp) ok
7. Rename file test.txt to renamed.txt. cp then rm mv test.txt renamed.txt
8. Change to the parent directory.
9. Remove the file called renamed.txt without changing into the directory Sub. rm -r
10. Remove directroy Sub using the command rmdir.
11. Delete the directory sub using the command rmdir
12. Delete the directory temp using the command rmdir

1.3 Looking at Files

1. From the home directory of user cquast copy the file called course.tar.bz2 found in the subdirectory eu us course files.
2. Unpack the archive into a subdirectory calld course.
3. Change into the directory called course.
4. Print the content of file README using the commands cat and less. What is the difference betweens these two commands? opens
   - new page I can't figure how to get out of
   - but lets you navigate that pg
5. List the content of the current directory

2 I/O Redirection

Book Chapters: 7
Directory: course/exercises/linux_introduction/io redirection

2.1 Operators '>' and '>>'

1. Create a file named `test.fasta` that cotaincs the linc `>hello world` sequence. Use the `echo` BASH built-in
2. Print the content of the file to stdout.
3. Add a new line to the end of the file containing any number of 'a', 'c', 'g', 't' and 'n' using the >> operator.
4. Print the content of the file to stdout.
5. Add a new line ('this is the last line') using the operator >.
6. Print the content of the file to stdout.
7. What is the difference between operators > and >>.

2.2 Operator '|'

Sorting

1. Sort the file `unsorted lines` in alphanumeric order, printing only unique lines.
2. Print only the first / last five lines of the sorted lines.
3. Print the number of times each line is contained in the file.

More Complex Use of the '|' Operator

1. The command `ps afxu` prints a list of all processes currently running on the computer. The first column is the name of the user who started the process. Additionally to the commands introduced in section 7. use the command `cut` to delete unwanted *fields* from the output. By default `cut` uses the tab character as field separator.
   - Sort the list according to the username.
   - One of the lines starts with the word 'USER'. Use `grep` and the pipe operator '|' to remove this line from the output. The pattern for grep is `^USER`.
   - List only those processes started by your user.
   - Print a unique list of users running process on your computer (print only the user name).
   - How many processes is each user running.

djvu → pdf

gp - ... 2 print

while <something> ; do <whatever> ; done

bash course_script.txt

3 Globbing / Pattern Expansion

Book Chapters: 8
Directory: course/exercises/linux_introduction/wildcards

1. List the content of the current directory (long listing format). ll
2. Use the commands ls and wc to count the number of files with the extensions: ls -l | grep .txt | wc -l
   - bla, blb, blubba. grr, txt.
3. Delete all files with the extension blubba. Use only one command to do this. rm *blubba
4. Delete all files with the extensions bla and blb. Use only one command to do this. rm *bla *blb rm -i first
   ask individ before remove
5. Delete all files with the extensions grr and txt. Use only one command to do this.
6. Go to the parent directory and delete the directory wildcards. rmdir wildcards

| head -n -1 gets rids of the last line

head -n+4 / gives from 4th position

all files with 1,2 or 5

ls -p file[125].txt

ls -p file[^3].txt

all except 3

Sed 's/what/that before/what want after/'

4 Search for / in Files

Book Chapters: 18, 20, 21

1. Remove all spaces from the names of directories that contain any.

4.1 Text Processing

Directory: course/exercises/linux_introduction/search in files

1. How many lines does the file `lines.txt` have.
2. Use the command grep to only print lines containing the word `blubba`.
3. How many lines are these. Be aware that some these lines may contain words containing `blubbaaa`. These lines are not to be counted.
4. How many different lines are in file `linex.txt` and how many times is each line repeated.
5. For each of the following words find the number of files that contain the word:

   - `bla`, `abc`, `def`

   Be aware that some files may have multiple lines containing the word to search.
6. Translate the content of file `text70.txt` from lowercase to uppercase.
7. In each file containing the word `bla`, replace this word by the word `new word`.
8. In subdirectory `statistics` there are some files containing length distributions for different rDNA samples. Join all these files into a multi column file. The first column of the new file should contain the legnth of the sequence. All following columns should contain the counts for the different samples. The fields in the files are tab delimited.

   Be aware that not all files have the same number of lines.

4.2 Search for Files

Directory: course

1. Use find to find all directories in this directory and all subdiretories.
2. How many subdirectories are these.
3. How many files are in these directories.
4. How many files with extension `txt` are in these directories.
5. Use find to delete all files whoes filename contain the numer `100`.

5 Reciprocal Best Match (Simplified)

Book Chapters: all of the previous and `Part 2`

The goal is to write a script that finds the *reciprocal best matches* between two Genomes. Reciprocal best matches are: two proteins from different genomes that are each others best BLASTp match[1]. For this exercise, comparing all genes found in the first genome to all predicted genes in the second genome shall be enough.

Additionally, the matches must suffice minimal quality standards. For this exercise, let this be an *identity* of at least 60% and a *bit score* of at least 100 as reported by BLASTp. These thresholds are arbitrarily chosen.

> The Basic Local Alignment Search Tool (BLAST) finds regions of local similarity between sequences. The program compares nucleotide or protein sequences to sequence databases and calculates the statistical significance of matches. It can be used to infer functional and evolutionary relationships between sequences as well as help identify members of gene families. [2]

5.1 Requirements

- Write a script that accepts two files, each containing a genome in the FASTA Format as input, on the command line.
- use metagene[3] to predict ORFs.
- extract the sequence for each predicted ORF from the genomes,
- convert the extracted DNA sequences to protein (amino acid) sequences,
- build a BLASTp database containing all protein sequences of the first genome and a second database for the ORFs of the second genome,
- compare all genes found in the first genome to all genes predicted in the second genome (blastall) – *note:* use option -m8 to manipulate the output format,
- extract a list of matches from the results that suffice the afore mentioned quality thresholds,
- compare these candidate genes to genes in the SwissProt database to get hints about their biological function, store these results.
- Document Your Code :)

[1] `http://asap.ahabs.wisc.edu/~genetics875/lectures/Lec5-Homology2007.ppt`
[2] `http://blast.ncbi.nlm.nih.gov/Blast.cgi`
[3] `/usr/local/software/metagene`

5.2 Support

Gene Prediction Open reading frames (ORFs) are potential genes that describe a subregion of the genomic sequence between a start and stop codon. ORFs can be found on the leading (forward) and lagging (backward) strand of the genome. Because DNA sequences are translated to protein sequences based on triplets each strand has three reading frames. The first starting at the first position of the sequences (offset 0), the second at the second position (offset 1), and the third at the third position (offset 2).

Be sure to understand the output format of the *metagene* program. While parsing the result, only accept those lines that describe a valid ORF prediction, e.g. those lines that have the correct format (are all fields set?). Do the fields have the correct format? Note, that ORFs may be predicted in both forward (+) and backward (-) directions. What does this mean for extracting the DNA sequence from the genome? Also note, ORFs may only be predicted partially. In these cases, the reading frame of the ORF is not defined by '*start module* 3'. Again, what does this mean for extracting the DNA sequence from the genome?

Tools to work with FASTA Files Genpak (GP)[4] is a set of small utilities written in ANSI C to manipulate DNA sequences in a Unix fashion. This scripts can easily be used in larger shell scripts. Each script takes a FASTA file as input and creates a new file as output. If the output file is omitted the results will be printed to stdout. You can use the output of one command as the input for another program by using the pipe operator '|'. man pages are provided for each tool. Additionally, a man page called 'Genpak' is provided that briefly describes the purpose of this tool collection and it lists all programs part of the collection.

A known bug in the GP tool collection is that some of the tools only work with all Upper case sequence characters. You can use the *tr* command to convert a lower case string to an upper case string.

Sequence Searches The tool *blastall* can be used to search a single ORF or a set of ORFs against a database containing known genes. When using the blastall tool you have to specify the type of search you want to run: BLAST vs. a nucleotide database, BLAST vs. protein database, a six frame translation of a query sequence against a protein database.

Custom databases to be used with the blastall command can be created with the *formatdb* command. Make sure that you use the correct set of options to create a database suitable for the type of search you want to run.

The SwissProt database, that should be used to identify the function of a candidate ORF, can be found in `/local/biodb/blast`.

5.3 Hints

Divide and Conquer is your friend: solve parts of the problem on the commandline before you put them in your script. Divide your code into functions and start by writting the function to predict ORFs. Then continue with writing the functions in the order given above. Once a function works, save the result produced by that function to a 'backup' directory and start with the next function.

[4] `http://www.bioinformatics.org/genpak/index.html#gp`

Disable the previous function and use the results to test the new function. If your function does not work, you still have a backup of the results of the previous step (copy back to a working directory and try again). Store the results of the different functions in different subdirectories.

Simple CSV Parser The simplest method to parse a white space separated file in BASH is to use commands **cat**, **while**, and **read**. The command **read** is a BASH builtin command[5] that reads a line from the standard input, or from the file descriptor fd supplied as an argument to the -u option, and the first word is assigned to the first variable, the second word to the second variable, and so forth. Make sure you understand the synopsis of the the *read* builtin. Otherwise you may get unexpected results. An example may look like this:

```
1 #!/bin/bash
2 cat SOMEFILE | while read FIRSTFIELD SECONDFIELD ...; do
3     COMMAND...
4 done
```

The Solution

There are many solutions how to solve this exercise. You may use the programs and hints provided in this manual or you may solve the problem a different way. It's up to you but don't forget to explain your code.

Happy Coding... ;)

[5]Search in the BASH man page for '^SHELL BUILTIN' to get a complete description of the **read** command

A Basic Commands

Command	Short Description
`pwd`	Prints the (absolute) path of the current directory.
`cd [dir]`	Change current directory. The command will change to the users home directory if the directory name is omitted.
`ls [dir]`	List directory content. If no directory name is given, list the content of the current directory. If a file is specified instead of a directory show only the named file.
`touch FILE ...`	Update the access and modification times of each FILE to the current time. If FILE does not exists then an empty file is created.
`grep "bla" FILE ...`	Grep searches the named input FILEs (or standard input if no files are named, or the file name - is given) for lines containing a match to the given PATTERN (*bla*).
`cat FILE ...`	Prints the content of the given files to `stdout`
`less FILE`	Like `cat`. Additionally, `less` allows to navigate within the file using the arrow keys. Use `q` to quit and `h` to show a help screen.
`cp a b`	Create a copy of file 'a' with the name 'b'.
`rm a`	Delete file 'a'
`mv b a`	Move (rename) files. If 'a' is a directory then file 'b' will be moved into it. Otherwise 'b' will be renamed to 'a'.
`mkdir c`	Create a new directory with name 'c'.
`rmdir c`	Delete empty directory 'c'. To delete a directory including its content use `rm -r c`.
`locate expr`	Find a file or directory containing 'expr' within the complete filesystem.
`find [dir] [expr]`	Search for files in a directory hierarchy starting in 'dir'. If no directory is specified and no expression is given then the command prints all files within the current directory and below.
`wc FILE ...`	Print newline, word, and byte counts for each file.

Tabelle 1: Table of commonly used commands

B Globbing

Wildcard	Description
?	Any character.
*	Any characters any number of times including zero. The expressinon 'ls `*.txt`' lists all text files in the current directory. Hidden files (those starting with '.') are not matched.
h*l	Matches any file starting with charater 'h' and ending with 'l', e.g: 'hl', 'hel', or 'h1123afasfasdvb224234'.
.*	Matches all hidden files.
{*,.[!.]*}	Matches all regular files as well as all hidden files.
.	Matches all files containing at least one period in their name.
abc*.*	Matches all files starting with 'abc' which contain at least one period.
[abc]	Matches any of the characters 'a', 'b', or 'c'.
[o-u]	Matches any of the characters 'o' through 'u'.
[A-Z]	Matches any upper case character.
[A-Z]7	Matches all files starting with an upper case character followed by the number '7'.
[A-Za-z]dat	Matches all files starting with an upper case character followed by the string 'dat'.
h[ea]llo	Matches the files 'hallo' and 'hello'.
[!x] or [^x]	The ! or ^ exclude the specified characters.
h[!ae]llo	Matches any file starting with 'h' and ending with 'llo' but not 'hallo' and 'hello'.
*.{jpeg,jpg}	Matches all JPEG images.
image_of_{moon,sun}.jpeg	Matches 'image_of_moon.jpeg' as well as 'image_of_sun.jpeg'.

Tabelle 2: Shell Globbing and pattern expansion