# The Scientific Communication of Comparative Genomics

Teacher: Prof. David W. Ussery
Assistant teacher and exercises: Tammi Vesth

December 6, 2011

# Unix

## Introduction to Unix

### Shell

There is a special type of window called *shell* or *terminalwindow*. Terminal windows are the principal vehicle of interaction with a UNIX machine. Their function is to perform the commands typed into them. An active terminal window will display a prompt and pause waiting for a command. The prompt can look like this:

```
genome[phd14]:/home/people/phd14/alignment>
```

It means that you have logged in to the machine called *genome*, your username is *phd*14 and you are in the directory (=folder) *alignment* in the directory *phd*14 in the directory *people* in the directory *home* at the highest level of the file hierarchy. The commands are submitted by typing them after the prompt and then hitting the *RETURN* key. The command you have typed is not submitted until you hit the *RETURN* key. You can move back and forth in the command string using the *LEFT* and *RIGHT*arrow keys and correct mistakes quietly. A command line may be longer than a line on the screen, just keep typing! As soon as the *RETURN* key has been hit the execution starts. Do not be alarmed if nothing happens at once; sometimes it takes a while to load and activate a program.

### File system navigation

In this guide the greater-than and less-than signs around something, for example like this <something> means: you type ÒsomethingÓ here. It will be obvious in the actual situation, what you should write. Example: <filename> should be replaced with the actual name of the file: MyFile.txt File and directory (folder) names should not include spaces, Danish or other outlandish letters etc. The following characters are safe "*abcdefghijklmnopqrstuvxwz*", "*ABCDEFGHIJKLMNOPQRSTUVXWZ*", "0123456789" and ". – _". Case matters in UNIX. Capital letters are not the same as small letters.

```
Example: MyFile.txt is not the same as myfile.txt.
```

File extensions (or the .doc, .txt things added to the end of files) do not matter in UNIX, but they are needed for other systems. It is a good idea to use them so you know what sort of file you are using eg a file with just text in should have a .txt extension, and other extensions you use should be consistent and relevant to the file.

File permissions is a difficult subject, but important to know in UNIX! Every file and directory is equipped with some controls over who are allowed to do what with it. In UNIX terminology there are three different groups of people; Owner, ownergroups, and everyone else. The groups have three kinds of access rights to the file; *read*, *write* and *execute* permissions. Read access means that you can read the file, write means that you can write or change the file, execute means that the file is a program, which you can run or execute. Here is an example:(you get this output on the screen by using the $ls - l$ command which lists the access status of the file);

```
-rwxr--r--  1 root  sys 113520 Sep 5 14:15 goodProgram
```

This file is named *goodProgram*, the size is 113520 bytes, it was changed/created at *Sep 5 14:15*, the owner is named *root*, and the group is *sys*. The file permissions are $rwxr--r--$, which means that the owner can read, write (change), and execute the file (it is a program). The group, *sys*, can read the file, and so can everyone else. So the owner, *root*, is the only one who can change and execute the program, but he/she does not mind if everyone else looks at what he/she is doing. File permissions are simply three triplet of *rwx*. Now if the file really is a directory, then permissions looks like this:

```
drwxr-xr-x  6 gorm  user    72 Jun 26 23:50 gorm_directory
```

Notice the *d* in the beginning of the permissions, it means that this is a directory. The permissions on directories have a slightly different meaning than on files. Write permission means right to create and delete files in the directory (but not change existing files, THAT is governed by the file permissions). Execute permission means right to list the content of the directory. If the execute permission is not set, then the directory will look empty to you.

## Directories (=folders)

The contents of the current directory (=folder) can be examined by typing *ls* (*list*). It can look like this:

```
genome[phd14]:/home/people/phd14/alignment> ls
file1    file2 dir1 dir2
```

To get more information than just the file names, use $ls - l$ (*list long*). This gives you the permissions, ownership, size, and last modification time of all the files. You can change the file permissions to allow other to read or you to execute a file:

```
chmod <options> <filename>
chmod u+x <filename>    # Making a file executable
chmod ga+r <filename>   # Making a file readable by everyone
chmod ga+rx <directoryname> # Making a directory readable by everyone
```

You can change to a directory in the current directory with the command *cd*:

```
genome[phd14]:/home/people/phd14/alignment> cd dir2
genome[phd14]:/home/people/phd14/alignment/dir2>
```

Notice the the prompt changes as you go to another directory. To go up one level in the hierarchy, use *cd..*. To go to your home directory, use *cd* with no arguments:

```
genome[phd14]:/home/people/phd14/alignment/dir2> cd ..
genome[phd14]:/home/people/phd14/alignment> cd
genome[phd14]:/home/people/phd14>
```

Wherever you are, *cd* with no arguments will always take you to your home directory.

```
mkdir <directoryname>   # Creating a directory (folder)
rmdir <directoryname>   # Removing a directory. Only empty directories may be removed.
cd <directoryname>  # Changing directory.
cd ..   # Going up (back) i the directory hierarchy
pwd # What is the current directory (print working directory).
```

## Simple file commands

To copy one or more file(s), use *cp*:

```
cp file newfile
cp file1 file2 etc... <directory>
```

To rename a file or move one or more file(s), use *mv*:

```
mv file newfile
mv file1 file2 etc... directory
```

To delete (remove) one or more file(s), use *rm*:

```
rm file1 file2 etc...
```

To count the number of lines in a file is called *word counting*:

```
wc <filename>       # Counting lines in a file.
# Typical output looks like this:
17078   38712   939401 link.test
# The numbers are lines, words and bytes in the file.
```

It is possible to search a file for a specific pattern or word, this is called *grep*:

```
grep <word> <filename>
grep LOCUS <filename>   # Remember that Unix is case sensitive
grep "<" <filename> # When searching for signs they must be put in quotation marks
```

This is an incredibly useful command, very versatile. The expression *grep promoter genome.txt*, lists all lines with the word promoter in the file *genome.txt*.
Extracting columns of data from a file. Extremely useful command for getting data out of tabular files.

```
# Input file type:
SwissProt GenBank   Cat1 Cat2 Cat3
K6PL_HUMAN X15573.CDS.1 0.6284 0.9183 -2.9719
G6PI_HUMAN K03515.CDS.1 1.0377 1.9856 -1.1401
G6PD_HUMAN L44140.CDS.10    3.1217 6.8903 -4.3967

cut <options> <filename>
# If you are only interested in Category 2 numbers (Cat2), you get them like this;
cut -f 4 <filename>
```

The parameter $-f$ stands for $field$ and you can write $-f2-5$ (fields 2 to 5) or $-f2,4$ (2 and 4). *cut* has a weakness, it operates best on tab delimited files.
Merging files. Concentrates corresponding lines of the given input files $file1$, $file2$ and so on. It treats each file as a column or columns of a table and pastes them together horizontally (parallel merging).

```
paste <filename1> <filename2>
```

See the manual pages for details:

```
man cp
man mv
man rm
man cut
```

## Viewing and editing files

**Text files** A very short text file can be typed on the screen with the command

```
cat file
```

(where you should substitute file with the actual name of the file). Larger files may be viewed with a pager:

```
less file
```

which shows you one screenful at a time. When the viewing session starts you are shown the top of the file. You move around in the file as follows:

```
SPACE   one screenful forward
b       one screenful backward
RETURN  one line forward
k       one line backward
g       top (beginning) of file
G       bottom (end) of file
q       leave the session
```

If you want to create/modify a text file type

```
mousepad file
```

A new window will appear and the file named file will be shown in it ready for editing. If file does not exist you will get an empty window in which to type.

**Spreadsheet files** Xubuntu has a build in spreadsheet editor similar to *Excel* or $Numbers$ from Windows and Mac OSX. The editor is called *Gnumeric*.

**Other files** Files containing graphics can be viewed with many different tools. The choice of tool depends on the format of the file in question. Most often it is mentioned in the exercise manual. Your browser is configured so that it in most cases will launch an appropriate tool when you view a file (select *Open file* in the $File$ menu).

**Viewing and saving output from commands** Often, a UNIX command will produce much more output than there is room for on the the screen at one time. In this case, there are two things you can do: Pipe the output of the command through a pager:

```
command | less
```

In general, the construct *command*1|*command*2 (known as a pipe) means that the output from *command*1 is used as input to *command*2. You will see examples of this in the exercises. Save (redirect) the output to a file:

```
command > file
```

Then you can examine the file as described above, edit it, print it, or whatever you like.

**Manual pages** Most UNIX commands have manual pages which are viewed with the command man, e.g.

```
man align
```

The manual page for the command will automatically be piped to less (see above).

## Exercises

1. Use *mousepad* to create a file *mycommands.txt* where you write all commands and observations you do in the following exercises.
2. Note: There are more standard text editors than *mousepad*. Examples are emacs, xemacs, vi, vim, and pico.
3. First list the files in the directory.
4. Copy *ex*1*.acc* to *myfile.acc*.
5. Look at the content of both files to ensure they are identical.
6. Copy *ex*1*.dat* to *myfile.acc*.
7. Check that the content of *myfile.acc* changed.
8. Delete *myfile.acc*.
9. Make a directory *test* and move the three files to it.
10. Make a directory *data* and move the three files to that instead.
11. Remove *test* directory.
12. Change directory to *data* and confirm that you succeeded. Go back to the home directory afterwards.
13. Make three new directories *newtest* - one inside the other, like a russian doll.
14. Move the data directory to the innermost *newtest* directory.
15. Confirm that the three files are moved along with the data directory.
16. Copy the three files to your home (your top directory).
17. Remove all *newtest* directories and data in the with a single command. There may be a lot of confirmations. These are not considered part of the command. They are annoyances.
18. Count the lines in *ex*1*.acc* and *ex*1*.dat*.

19. Concatenate $ex1.acc$ and $ex1.dat$ in the file $ex1.tot$, i.e. copy the content of two files into one new file. Verify that all gene IDs comes first followed by numerical data.
20. Paste $ex1.acc$ and $ex1.dat$ together in $ex1.tot$, thus destroying the old file. Verify that corresponding gene IDs and numerical data are put on the same line. as the data.
21. Extract (cut) SwissProt ID and 1st and 3nd numerical data (column 1,3,5) from $ex1.tot$. Put results into a file $ex1.res$.
22. Extract GenBank ID and 1st numerical data (column 2,3) from $ex1.tot$. Put results into a file $ex1.res$.
23. Find the lines (using grep) in *orphans.sp* which contain a GenBank accession number. There are 85, verify this. Note: An accession number is one or two capital letters and looks like this $AB000114.CDS.1$, the $.CDS.$ part is kind of optional.
24. How many human genes with SwissProt IDs in *orphans.sp* exist? How many of those are hypothetical? (11)
25. How many genes belong to the rat, and how many of those are precursors? (9) Note: A Swissprot ID looks like $PARG_{H}UMAN$ or $TF1A_{M}OUSE$, with the gene being before the underscore and the organism after the underscore.
26. This little exercise will require that *man* is used for help on *grep*. From the file $ex1.res$ find the lines with positive numbers and put then into $ex1.pos$. The lines with negative number go into $ex1.neg$.

```
Hint: try some of these options:
man grep    # explanations can be found here
grep -v - ex1.res
grep -e - ex1.res
grep - ex1.res)
```

27. Write a shell script that solves exercise 19-24, with the exercises clearly separated in both the script and the output. This should be straight forward (but long), especially since you took notes (exercise 1).
28. Mail your *mycommands.txt* file to the teacher for comments.