Unix tutorial, session 2

H. Seitz (IGH)

December 5, 2013

Contents

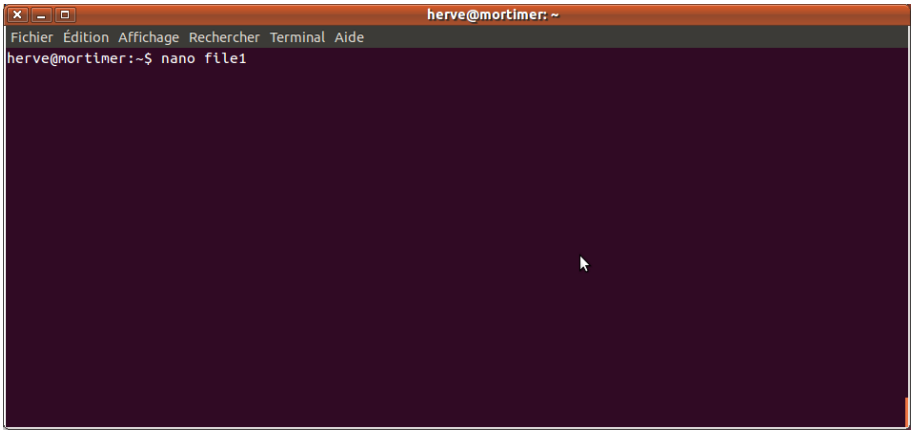
1 A text editor: nano

We learned how to move, rename and delete files, then we learned how to view their content (first session), but we have not yet learned to modify them. For this purpose, we will need a "text editor" (*i.e.*: a program which allows you to type text in a file).

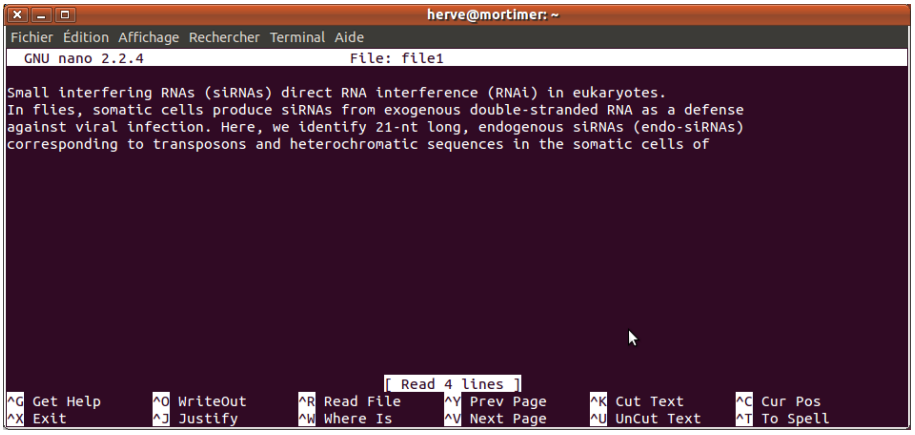
There are many text editors; some of them can do many things (like `emacs`; some people say it even does too much ...), some are very easy to use (like `joe` and `nano`), and some are very powerful, but somewhat hard to learn (like `vim`).

Apparently `nano` is installed by default on most systems (including MacOS X systems): that's the one we will use.

To open a file (to edit it) with `nano`, type `nano` followed by the name of the file (then press Enter):



then after pressing Enter:



You can then navigate in the text using the arrows, insert text by typing it, delete it with "Backspace" or "Delete", and exit (and save) by typing "Ctrl-x" (*NB*: a few basic commands are displayed on the two bottom lines of the `nano` window; the carret "^" symbol means "Ctrl").

2 Using scripts

2.1 Executable files

Combining several Unix commands (piping them together with "|") can achieve complicated tasks (things like "count all the sequences in a fasta file, whose name contains the word 'kinase' but not the word 'creatine', and whose second nucleotide is a 'T' while the third one is neither an 'A' nor a 'G' " ...), and this, in a single line. But of course, that line can be very long (with multiple steps piped to each other), and it can be very long to actually invent, and type, the command. Hence it would be nice to find a way to save these commands, so that they can be re-used later.

One obvious way, would be to copy-paste the line of commands into a text file, save that file, then (when needed), open that file and copy-paste the command into a terminal. There is a simpler way to do that: saving the command as an executable file.

In Unix, every file can be made executable, and used as a program. If that file contained a single line like:
`ls | grep -v '\.jpg$'`, then executing that file will execute that command; if it contained other things (for example, a real text, like "Small interfering RNAs (siRNAs) direct RNA interference (RNAi) in eukaryotes. In flies, somatic cells produce [...]"), then executing it will fail, and you will get an error message.

When a file is executable, it becomes a program that can be run, and (if it contains commands that can modify your data), this is potentially harmful to your computer. So it is important to control the permission of executability of your files (if your computer is connected to the Internet, you might want to allow the entire world to view that file, but not to execute it, for example). You can make a file executable *by yourself* (and yourself only) by typing:
`chmod u+x name_of_file`
("chmod" stands for "change mode"; "u" stands for "user": that's you, and nobody else; "+" means that you add a permission, and "x" means: executability). You can make a file executable only to a group of users (these users have to have an account on your computer, and to be grouped accordingly by the administrator; but I guess that, on your laptops, there is a single account, hence a single user: yourself), by typing `chmod g+x name_of_file`, and you can make it executable by everybody, by typing: `chmod o+x name_of_file` ("g" stands for "group", and "o", for "others").

Similarly, you can make a file readable to yourself only, or to your group only, or to everybody, with the same command, but replacing the "x" (executability), by "r"; and you can make a file modifiable by the same people, replacing that "x" by "w" (like: "write"). You can remove a permission, by replacing "+" by "-" in that command (for example, `chmod o-x name_of_file` will remove the permission, made to others, to execute that file).

Exercise

What would be the consequence of typing:
`chmod u+w sequences.fa`
?

2.2 Executing a file

Once a file has been made executable, you can run it by invoking its name, preceded by "./" (if the file is in the current directory; if that executable file is in the directory just upstream your current directory, you will type "../", and so on: this thing describes in what directory the program is).

For example, if I write a very simple program, that I will call 'list', made of a single line:

```
ls
```

(that program will just run the command `ls`), I can run this program by typing:

```
./list
```

in the directory where that file is stored. If I forgot to make that file executable first, I will get an error message, stating that I don't have the permission to execute that file.

The output of that command `./list` will be exactly the same than the output of the command `ls`, because `list` does nothing else than calling `ls`.

2.3 Application

Such an executable file is usually called a "script". It's a series of commands, that you don't type in a terminal: you put them in a file, and execute the file.

It is preferrable to write, as the first line of your script, this line:

```
#!/bin/sh
```

It is often not necessary (as we saw above: just writing the commands in the file works perfectly fine), but it will make a difference in some specific cases – so it's a good habit to add that line at the top of each of your scripts (unless you explicitly want them to behave in another, specific manner).

So a typical script would look like:

```
#!/bin/sh
name_of_command_1
name_of_command_2
name_of_command_3
```

2.4 Variables

You can make your script interactive by using variables: the following script:

```
#!/bin/sh
echo 'Hi, what is your name?'
read name
echo 'Your name is '$name
```

will ask you for your name, then repeat it (the command `echo` displays its argument – which can be quoted with single or double quotes – and the command `read` assigns a value to a variable). A variable's content (here: the user's name) is symbolized by the variable name, preceded by a dollar sign.

Exercise

Write the above script, and execute it. What happens if you omit the dollar sign in the last line?

3 Sed, the stream editor

3.1 Substituting characters on the fly

`sed` is a command that allows you to do a "search-and-replace" operation on the content of a text file. The syntax is:
`sed 's|A|a|' name_of_file`
(first, `sed`, which is the name of the command; then a space, then, between quotes: the character "s" – for: "substitution" – then a pipe, then the pattern to be found – here, a capital "A" – then another pipe, then the character string by which you want to replace that pattern – here, a lower case "a" – then a pipe again; then the input file name)

That command will display (on your terminal) the content of the input file, except that capital A's will be changed to lower case a's. The pipe is a delimiter: it is used to separate the "s" command from the searched pattern, from the replacing character string, and from the end of the command. The delimiter can actually be any character (except a quote, which is used here to isolate the content of the command): typing
`sed 's?A?a?'  name_of_file`
or
`sed 's/A/a/' name_of_file`
or
`sed 'sbAbab' name_of_file`
or ...
would do the exact same job – the only important thing, is that this delimiter must be absent from the searched pattern and the replacing character string (so that there are, in total, only three occurences of the delimiter in the whole command).

The above command will not replace every "A" by an "a": by default, it will only replace the first "A" of each line. If you want the modification to apply to every "A", you have to make the substitution "global", by adding a "g" at the end of the command:

```
sed 's|A|a|g' name_of_file
```

So, so far, everything looks like what you can do with a "Search-and-replace" in Word (except that, here, the output of that command is printed on the screen, it is not a modification of the input file: after typing that command, the input file is not modified, it still contains its capital A's – just the displayed output contained lower case a's instead; but of course, you can redirect the output of that `sed` command to an output file). A major difference with the "Search-and-replace" from a text editor, is that `sed` can be piped with other commands – so, instead of applying to an input file, it will apply to the output of the previous command, substituting the pattern by the replacing string character on the fly: the command
`grep -v '[abcd]' input_file | sed 's/et/ET/g'`
will extract all the lines that do not contain any of the characters {a,b,c,d}, then, in the remaining lines, it will replace lower case "et"'s by capital "ET"'s.

Exercise

What will be the ouput of the command:
`echo ‘‘Good night, sleep tight, don’t let the bed bugs bite.’’ | sed ’s|be|ba|’`
?

3.2 Restricting the replacement to specific lines

You can tell `sed` to apply its substitutions only to some of the lines in the imput (whether this is an input file, or the output of a piped command), by adding special instructions at the beginning of the `sed` command.

If you want the substitution to apply only to the second line of the input, type:
`sed ’2 s|A|a|g’ name_of_file`
(the number “2”, between the first quote and the “s” command, specifies the line number where the command shall apply; there is a space between the number “2” and the “s” character).

If you want the substitution to apply only to the lines 3 to 5, type:
`sed ’3,5 s|A|a|g’ name_of_file`
(the range of modified lines is indicated by the two line numbers, separated by a comma).

If you want the substitution to apply only to the last line, type:
`sed ’$ s|A|a|g’ name_of_file`
(here, the dollar sign means “last line of the input”).

You can also restrict the `sed` command to lines matching a given pattern: that pattern has to be delimited with slashes. For example:
`sed ’/>/ s|A|a|g’ name_of_file`
will replace capital A’s by lower case a’s only in the lines that contain the “>” character.

You can also revert the line selection, with an exclamation mark just before the “s” command: it will tell `sed` to apply the command *everywhere except* at the specified lines (whether these lines are specified by a line number, or by a pattern to be matched). For example:
`sed ’/>/ !s|A|a|g’ name_of_file`
will replace capital A’s by lower case a’s in every line, except the ones that contain a “>” character; and
`sed ’3 !s|A|a|g’ name_of_file`
will replace capital A’s by lower case a’s in every line, except on the third line.

3.3 Combining several sed commands

If you need to combine several `sed` commands (for example, you want to replace capital A’s by lower case a’s, capital B’s by lower case b’s, and so on), you can, of course, pipe several invocations of `sed`:
`sed ’s|A|a|g’ name_of_file | sed ’s|B|b|g’`
but there is a simpler way to do it: you can use several commands with a single invocation of sed, provided that each command is preceded by “-e”. Hence, simply typing:
`sed -e ’s|A|a|g’ -e ’s|B|b|g’ name_of_file`
will do both replacements (note that the input file name is written at the very end, after all the `sed` commands).

These replacements will occur sequentially: first, all the A’s will be replaced by a’s, then the B’s will be replaced by b’s. This is important, as the order of the commands can change the output

(for example, `sed -e 's|Are|are|g' -e 's|A|B|g' name_of_file` will first replace the "Are"'s by "are"'s, then replace the remaining A's by B's – if you had first replaced the A's by B's, then all the "Are"'s will have been replaced by "Bre"'s, hence the `'s|Are|are|g'` substitution would not find its pattern, and would not do any replacement).

3.4 Everything about sed

A very useful `sed` tutorial can be found at: `http://www.grymoire.com/Unix/Sed.html` (that's my bible for `sed` ... it is easy to find: that's the first Google hit when you search "sed"). You can do many more things with `sed`, than what I described above – and that webpage explains all that, and provides useful examples for every command.

As a webpage is never eternal, it is always good to save the important ones locally: it could be a good idea to save that HTML file in your archives for future use.

Exercise

Write a program that will compute the reverse-complement of the sequences in an input file (assuming that the sequences in that input file fit on single lines – but your program will have to display a warning message to state that). The reverse-complemented sequences will be written in the fasta format, in a file whose name will be: the name of the input file, preceded by the prefix "ANTI".

Hints: you will need the commands `echo`, `read`, `grep`, `sed`, `tr` and `rev`.