

Advancing with the Shell

Angelika Merkel (Head of Bioinformatics Unit IJC)
11/06/2024

<http://www.carrerasresearch.org/en/units/bioinformatics>

The IJC Bioinformatics Unit



Angelika Merkel
Unit Manager
Bioinformatician



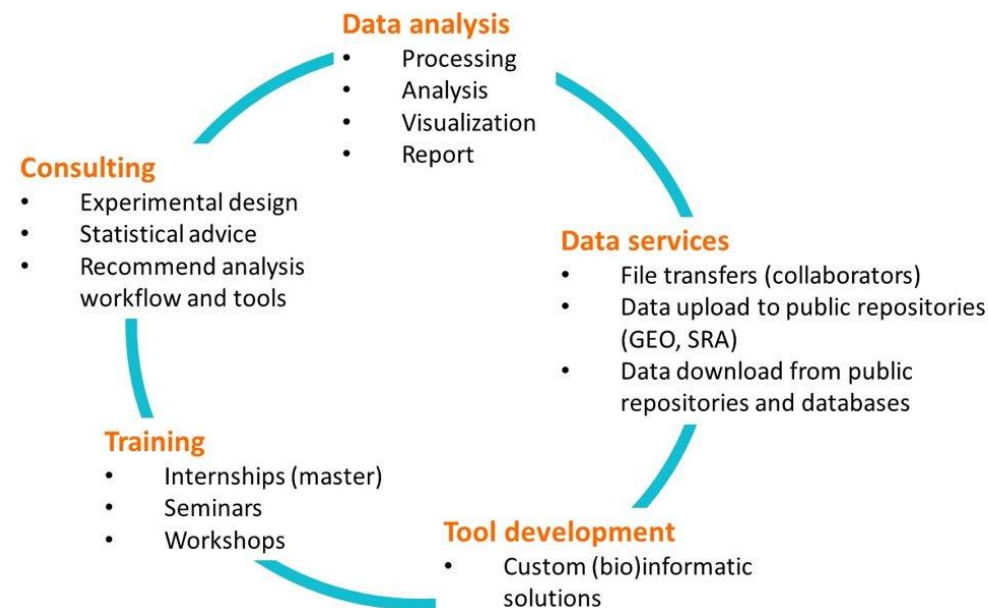
Emilio Lario
Software Engineer



Marina Vilardell
Bioinformatician



Marta Meroño
Bioinformatician
(Student)



Office: 1st floor; phone: 4300

<https://ijcbit.eu>

<https://www.carrerasresearch.org/en/bioinformatics-unit>



Workshop overview

Content:

1. Bash: Expansions and substitutions
2. Egrep and regular expression
3. File manipulations with sed and awk
4. Customizing your Shell

Workshop materials:

<https://ijcbit.github.io/Workshops/>

Linux course server:

<https://vpn.carrerasresearch.org>

Bash: Expansions and substitutions

Syntax elements in the command line can be interpreted or taken literally.

Bash splits the command line into tokens(words), scans for special elements and interprets them, resulting in a changed command line:

-> the elements are said to be **expanded** to or **substituted** to **new text and maybe new tokens** (words).

Pathname expansion (wildcards)

<code>*.txt</code>	# any number of any character(s)
<code>page_1?.html</code>	# any one particular character
<code>[SK][0-9].fastq</code>	# 'S' or 'K' followed by any number between 0 and 9

Example:

<code>\$ ls *.txt</code>	# list all files that end with '.txt'
--------------------------	---------------------------------------

Command substitution

```
$( COMMAND )    # substitute COMMAND with the output of that command  
`COMMAND`      # same as above
```

Example:

```
$ wc -w $(ls)
```

*E1

Arithmetic expansion

`$((EXPRESSION))` # evaluates the expression in a numeric context

Example:

`$((5+1))` # addition

`$((5*6))` # multiplication

`$((6/2))` # division

`$((++number))` # pre-increment

`$((--number))` # pre-decrement

Brace expansion

```
{X,Y,Z}          # individual elements  
{X..Y}          # sequence from..to  
{X..Y..Z}       # sequence from..to..by
```

Example:

```
$ echo a{d,c,b}e
```

```
ade ace abe
```

*E2

Parameter expansion

`$PARAMETER` same as `${PARAMETER}` # expand parameter value
`${STUFF...}` # expand value with modifications

Use '#' or '%' to expand only a part of a parameter's value, given a pattern to describe what to remove

`${PARAMETER#PATTERN}` # from the beginning, remove the shortest text matching the pattern
`${PARAMETER##PATTERN}` # from the beginning, remove the longest text matching the pattern

Example:

`$ MYSTRING="Be liberal in what you accept, and conservative in what you send"`

Syntax	Result
<code>\${MYSTRING#in}</code>	Be liberal in what you accept, and conservative in what you send
<code>\${MYSTRING##in}</code>	Be liberal in what you accept, and conservative in what you send

Parameter expansion

Use '#' or '%' to expand only a part of a parameter's value, given a pattern to describe what to remove

`${PARAMETER%PATTERN}` # from the end, remove the shortest text matching the pattern

`${PARAMETER%%PATTERN}` # from the end, remove the longest text matching the pattern

Example:

```
$ MYSTRING="Be liberal in what you accept, and conservative in what you send"
```

Syntax	Result
<code>\${MYSTRING%in}</code>	Be liberal in what you accept, and conservative in what you send
<code>\${MYSTRING%%in}</code>	Be liberal in what you accept, and conservative in what you send

Extract sample names:

```
$ BEDFILE="SAMPLENAME.bed"
```

```
$ echo ${BEDFILE%.bed}
```

*E3

Egrep and regular expression

egrep = print all lines matching a regular expression

egrep [command line options] <pattern> [path]

- v # print lines that do not match the regular expression
- w # match only complete words
- c # print a count of matching lines
- A N # after each matching line, print the next N lines from this file
- B N # before each matching line, print the next N lines from this file

Example:

```
$ grep -c 'chr1' peaks.bed
```

Regular expressions

Expression	Description
.(dot)	a single character.
?	the preceding character matches 0 or 1 times only.
*	the preceding character matches 0 or more times.
+	the preceding character matches 1 or more times.
{n}	the preceding character matches exactly n times.
{n,m}	the preceding character matches at least n times and not more than m times.
[agd]	the character is one of those included within the square brackets.
[^agd]	the character is not one of those included within the square brackets.
[c-f]	the dash within the square brackets operates as a range. In this case it means either the letters c, d, e or f.
()	allows us to group several characters to behave as one.
(pipe symbol)	the logical OR operation.
^	matches the beginning of the line.
\$	matches the end of the line.

Example: ***E4**

\$ egrep '^chr' peaks.bed

File manipulations with 'sed' (stream editor)

Sed = pattern-matching engine that can perform manipulations on lines of text

Examples

- `$ sed 's/red/hat/g' myfile` # print file and substitute all occurrences of "red" with "hat"
- `$ sed '1,10d' myfile` # print file with the first 10 lines removed
- `$ sed 's/^\s*//' file.txt` # remove all white spaces from the beginning of each line
- `$ sed '5 s/old/new/' file.txt` # Replace a string only on the 5th line

<https://quickref.me/sed.html>

*E5

File manipulations with awk

Awk/gawk/nawk = scripting language with pattern matching

```
# print first and fifth field; FS= input Field Separator, OFS= Output Field Separator *  
$ awk '{print $1, $5}' FS=";", OFS="\t" myfile
```

```
# filter first and fifth field by strings/values  
$ awk '{if($1== "chr1" && $5 >100){print}}' myfile
```

```
# filter first field for matches of 'chr1'  
$ awk '{if($1 ~ /chr1/){print}}' myfile
```

```
# for the first 5 rows print the number of field; NR= Number of rows, NF= Number of fields *  
$ awk '{if(NR<=5){print NF}}'  
myfile
```

* build-in variables to awk

<https://www.gnu.org/software/gawk/manual>

File manipulations with awk

```
# Split function (string, array, pattern))
```

```
$ awk '{ split($2, string_elements_array, " "); print $1, string_elements_array[2]}' myfile
```

```
## Code structure (BEGIN{}, {}, END{})
```

```
# Sum values of a field and print the final
```

```
awk '{ SUM+=$1 } END { print SUM }' myfile
```

```
# Sum values of a field and print file header, and print the final sum
```

```
awk '
```

```
BEGIN{ print "My header" }
```

```
{ SUM+=$1; print $1 }
```

```
END {print SUM }' myfile
```

*E6

Configure your Shell

Bash start-up files (configuration files): `.bashrc` & `.bash_profile`

- `~/.bashrc` executed on interactive non-login terminal windows
 - Use for commands that are executed every time you open a terminal (e.g. aliases, prompt)
- `/etc/profile` , `~/.bash_profile`, `~/.bash_login`, `~/.profile` executed on interactive login terminal windows
 - Use for commands that are executed once (e.g. environmental variables),
 - `.bash_profile` typically sources `.bashrc`

```
if [ -f ~/.bashrc ]; then
    . ~/.bashrc
fi
```


Environment variable and aliases

Set the path for your executable files

```
export PATH=$PATH:/place/with/the/file
```

Create command short cuts using an alias:

```
alias alias_name="command_to_run"
```

Example:

```
$ alias ll="ls -lrth"
```

Customize your prompt

Current prompt:

```
[amerkel@IJC20571R ~]$      # Displays user, host, base working directory, with brackets and '$'
```

Change via the environmental variable PS1:

```
export PS1="\u >"           # Display user and '>'
export PS1="\u@\h >"       # Display user, hostname (separated by '@') and '>'
export PS1="\W > "         # Export basename working directory and '>'
export PS1="\w > "         # Export working directory and '>'
```

Change the color of your prompt

```
export PS1="\e[0;32m[\u@\h \W]\$ \e[0m"
```

- `\e[` – Begin color changes
- `0;32m` – Specify the color code
- `[\u@\h \W]\$` – code for text BASH prompt (username@hostname Workingdirectory \$)
- `\e[0m` – Exit color-change mode

Customize your prompt (text format)

The first number in the color code specifies the typeface:

- **0** – Normal
- **1** – Bold (bright)
- **2** – Dim
- **4** – Underline

The second number indicates the color:

- **30** – Black
- **31** – Red
- **32** – Green
- **33** – Brown
- **34** – Blue
- **35** – Purple
- **36** – Cyan
- **37** – Light gray

<https://phoenixnap.com/kb/change-bash-prompt-linux>

Customize the color scheme of 'ls'

Use Linux utility 'dircolors'

```
$ dircolors --print-database > ~/.dir_colors
```

```
$ less ~/.dir_colors
```

```
TERM vt100
TERM xterm*
# Below are the color init strings for the basic file types. A color init
# string consists of one or more of the following numeric codes:
# Attribute codes:
# 00=none 01=bold 04=underscore 05=blink 07=reverse 08=concealed
# Text color codes:
# 30=black 31=red 32=green 33=yellow 34=blue 35=magenta 36=cyan 37=white
# Background color codes:
# 40=black 41=red 42=green 43=yellow 44=blue 45=magenta 46=cyan 47=white
#NORMAL 00 # no color code at all
#FILE 00 # regular file: use no color at all
RESET 0 # reset to "normal" color
DIR 01;34 # directory
```

```
$ eval $(dircolors ~/.dir_colors)
```

Add color support of ls to your .bashrc

```
if [ -x /usr/bin/dircolors ]; then
    test -r ~/.dircolors && eval "$(dircolors -b ~/.dircolors)" || eval "$(dircolors -b)":
fi
```

Questions?

Thank you!