Be in shell

Introduction

Pavel Fibich

pavel.fibich@prf.jcu.cz
dep. Botany - Na Zlaté stoce 1

02-2016

Přírodovědecká
fakulta
Faculty
of Science

Course motivation

Why to have such course in the bioinformatics study program?

- bioinformaticians are working with big and complex data, e.g. one pair read of tick genome in fasta sanger 1.9 format has 250 Gb
- there is need to easy manipulate, filter, sort, ... process big data

Linux shell is

- powerful, there are many tools for files, text and other manipulations
- reusable and fast, writting script avoids repeating Excel clicking
- widespread as platform, many bioinformatics toos are written for linux
- common in computational centers where you can use parallel processing of data

Be in shell allows you to *program on steroids* (fast writting of fast and powerful tools).

Course introduction

Goals of the course:

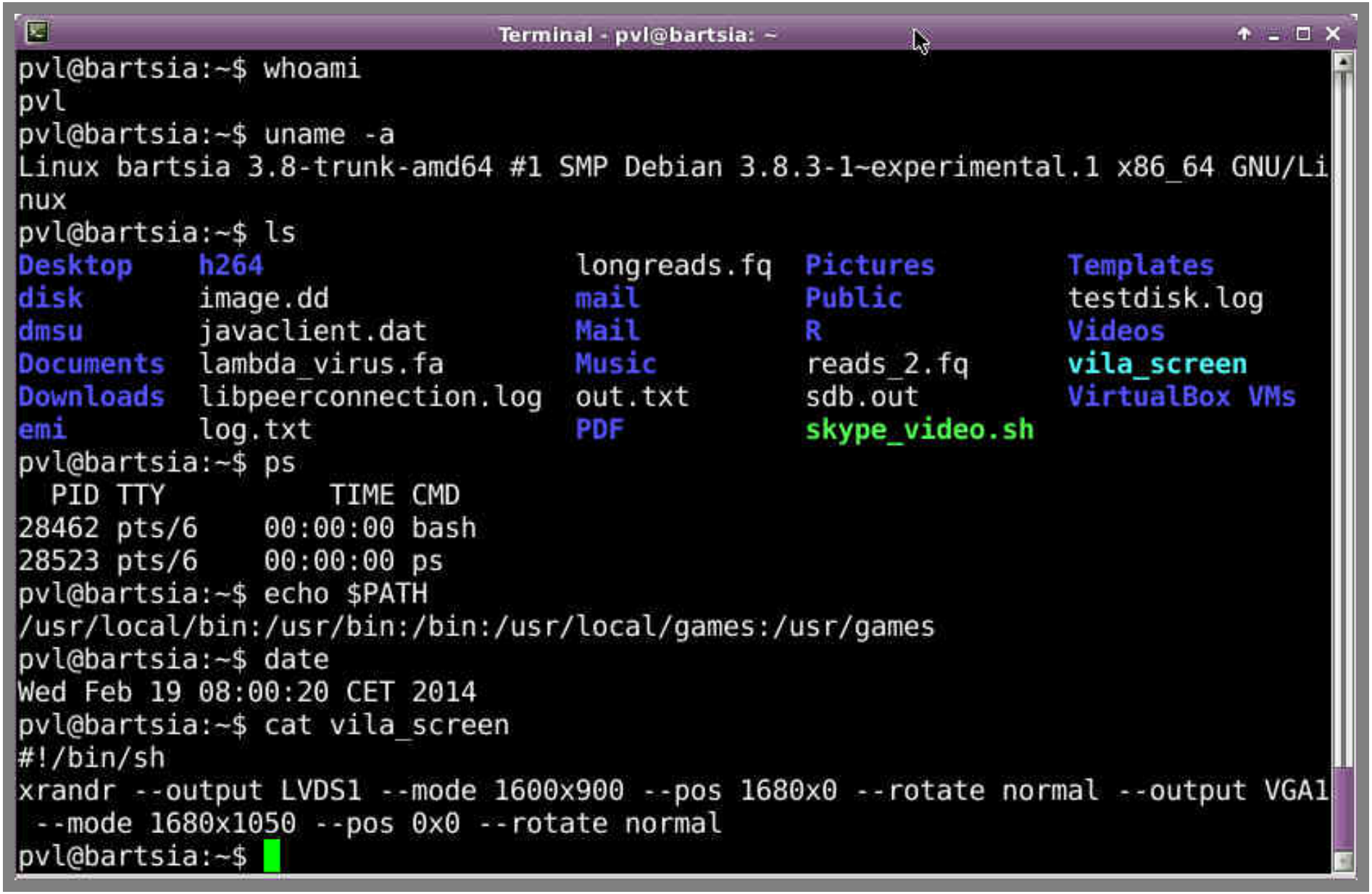
- be able to work under linux shell (i.e. bash) and use its power
- know and do not hesitate to use bash scripting, sed, AWK, GNUplot, R, python or perl
- BUT NOT to make linux administrators from you

How to make it:

- lectures every second week on Wed 13:15 – 15:30 at BB-7 room
- combination of theoretical and practical parts
- few practical homeworks influence final score
- final exam as discussion

No attendence checking!

Be in shell


Terminal - pvl@bartsia: ~
pvl@bartsia:~$ whoami
pvl
pvl@bartsia:~$ uname -a
Linux bartsia 3.8-trunk-amd64 #1 SMP Debian 3.8.3-1~experimental.1 x86_64 GNU/Li
nux
pvl@bartsia:~$ ls
Desktop h264 longreads.fq Pictures Templates
disk image.dd mail Public testdisk.log
dmsu javaclient.dat Mail R Videos
Documents lambda_virus.fa Music reads_2.fq vila_screen
Downloads libpeerconnection.log out.txt sdb.out VirtualBox VMs
emi log.txt PDF skype_video.sh
pvl@bartsia:~$ ps
PID TTY TIME CMD
28462 pts/6 00:00:00 bash
28523 pts/6 00:00:00 ps
pvl@bartsia:~$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
pvl@bartsia:~$ date
Wed Feb 19 08:00:20 CET 2014
pvl@bartsia:~$ cat vila_screen
#!/bin/sh
xrandr --output LVDS1 --mode 1600x900 --pos 1680x0 --rotate normal --output VGA1
--mode 1680x1050 --pos 0x0 --rotate normal
pvl@bartsia:~$

Materials

Shell scripting as ”Be in shell”

- Course materials at ‣ http://botanika.prf.jcu.cz/fibich/teaching.html
- Mostly all books about linux have some chapter about shell scripting
- Many online tutorials, just look for it
- books
  - Blum R. (2008) Linux Command Line and Shell Scripting Bible. Wiley.
  - Burtch K.O. (2004) Linux Shell Scripting with Bash. Sams Publishing.
  - Sobell M.G. (2009) Practical Guide to Linux Commands, Editors, and Shell Programming. Prentice Hall.

Program – working version

- 17.2. Introduction, work space, shells.
  Files, series of commands, output.
- 2.3. Variables, arithmetic expansion.
  Script basics, compound commands, if-else
- 16.3. *Training on exercises*
  Compound commands, loops, signals, **Homework 1**
- 30.3. Text files manipulations, regex, sed
  AWK, **Homework 2**
- 13.4. *Big exercise*
  GNUplot
- 27.4. R
  Python, Perl, *Big exercise 2*, **Homework 3**
- 11.5. *Big exercise 2*
  Final exam - the first and recommended try.

Keywords

Unix is operating system, has many variants

POSIX family of standards

Linux is Unix-like and POSIX-compliant operating system (Debian, Ubuntu, ...)

shell is user interface for accessing services of operating system (CLI or GUI)

sh is Bourne shell default in Unix systems

bash is Unix shell command line processor, default in Linux and MAC OS X, free replacement of sh

csh is C shell, syntax closer to C language

linux console (tty) is single user way how get/send infromations/commands from/to linux kernel

terminal is program that runs shell

xterm is terminal emulator for X window system

For the beginning, shell $\sim$ bash $\sim$ terminal assume as the same.

Bash in general

▸ www.gnu.org/software/bash

bash means Bourne-again shell

- command language interpreter written by Brian Fox, first released 1989
- widely distributed, even ported for MS Windows and cygwin
- typically runs in text window
- sh-compatible shell that incorporates useful features from the Korn shell (ksh) and C shell (csh)
- supports filename wildcarding, piping, command substitution, variables and control structures for condition-testing and iteration
- keywords, syntax and other basic features of the language were all copied from sh
- GNU GPL version 3

Workspace - playground

```
$ pwd
?
```

Ideally, to have own linux machine, or virtual one, with bash.

If you affraid of linux, use application VirtualBox (▸ www.virtualbox.org), create virtual machine and install of Debian or Ubuntu (or use some live linux, eg. Ubuntu). Mostly you need to get iso image with linux installation and attach it to the virtual machine to boot from it.

Where to start?

Log in linux OS and run terminal. You will get to command line, it often ends with $. After $, you can write commands, e.g. `whoami`

```
pvl@bartsia:~$ whoami  # what is my user name
pvl
pvl@bartsia:~$
```

- part before $ is mostly omitted (often USERNAME@COMPUTERNAME:ACTUALFOLDER, where ~ is alias for HOME directory)
- after $, there are commands written
- line without $ is the output of previous command, e.g. value of variable, content of file, ...
- after #, there are comments written, they are not interpreted (run) by shell

Few starting commands

```
$ date # actual date
Mon Jan 27 14:17:28 CET 2014
$ ls # list of files in the current folder
myfile
$ cat myfile # print content of file myfile
Nice day!
Nice shell!
$ grep day myfile # print lines having word day in file myfile
Nice day!
$ env # print list of all actual variables
...
$ echo $SHELL # print content of variable SHELL
/bin/bash
$ ps # snapshot of processes of current shell
  PID TTY          TIME CMD
21356 pts/0    00:00:00 bash
21399 pts/0    00:00:00 ps
```

Manuals - where to find help

We should start with `man` command for manual

```
$ man man  # manual pages of command man
$ man grep  # manual pages of command grep
$ apropos grep  # search for keyword grep
$ info grep  # different informations about grep
$ grep --help  # the most of commands have --help option
```

Mans have their options too (e.g. manual sections of man command, see `man man`)

```
$ man passwd  # passwd command changes password
$ man 5 passwd  # man about password file
```

Task

Always look at the man pages of the commands you are using (e.g. `man grep`), until you will get used to use them. `man bash` is quite compressive *long winter nights* article about what we will use.

Linux file system - paths

- Paths of folders and files are in linux separated by / (forward slash).
- Absolute path is full path from root (like in Windows from C:), e.g.
  `/etc/profile`
  `/home/pvl/bash/myFolder`
- Relative path starts from the actual folder (denoted by ., you can get it by `pwd` command), e.g.

```
./myFolder/myfile  # . is actual folder
../pvl/myfile  # .. is folder above the actual folder
```

- If no path with / or . is specified, than actual folder is assumed
- We will often work in some folder in home directory (denoted by ∼) of the user

```
pvl@bartsia:~/Documents/prf.jcu/bash$ pwd  # print work dir
/home/pvl/Documents/prf.jcu/bash
```

Files and Directories

Directories are rooted (start) in the / (slash)

```
$ pwd  # print actual directory
/home/pvl/Documents/prf.jcu/bash/c1
$ ls -a  # list of all files in the current directory
.  ..  myfile
$ cd ..  # go to the parent directory
$ pwd
/home/pvl/Documents/prf.jcu/bash
$ cd c1  # go to c1 directory
$ pwd  # print actual directory
/home/pvl/Documents/prf.jcu/bash/c1
$ ls -a /  # list of files in the file system root
...
```

If path is not specified, files/directories in the current folder are assumed!

Task

Look what is in the file system root folders (`ls -a /`).

Directories and files - manipulation and access

```
$ mkdir mydir  # creates directory
$ rmdir mydir  # deletes empty directory
$ rm -r mydir  # deletes directory recursively (everything inside)
$ cp file newfile  # copy file in the same dir
$ cp file /home/pvl/somewhere  # copy file into absolute path
$ mv file ./folder/newfile  # rename/move file or directory to new path
$ rm file  # remove file
```

Access rights - filetype, 3x(**r**ead, **w**rite, e**x**ecute) for user,group,others

```
$ ls -la
total 12
drwxr-xr-x 2 pvl pvl 4096 Jan 29 09:32 .
drwxr-xr-x 3 pvl pvl 4096 Jan 29 09:32 ..
-rw-r--r-- 1 pvl pvl   22 Jan 22 21:28 myfile
```

Task

Look at `/dev` for more filetypes.

Directories and files

```
$ file myfile # file type info
myfile: ASCII text, with very long lines
$ chmod a+rx myfile # modifying access rights, all can rx (read execute)
$ chmod go-rwx myfile # group cannot rwx
$ wc -l myfile # print number of lines
104 myfile
$ ln -s myfile linkfile # create pointer to file, link
$ ls -la # print all files with details
drwxr-xr-x 2 pvl pvl 4096 Jan 29 10:45 .
drwxr-xr-x 3 pvl pvl 4096 Jan 29 10:45 ..
lrwxrwxrwx 1 pvl pvl    6 Jan 29 10:45 linkfile -> myfile
-rw-r--r-- 1 pvl pvl 3828 Jan 29 10:01 myfile
$ ls -ltr # list according date, reversally
...
```

Symbolic links are very useful, you can link also folders and do not need extra copies.

When shell starts up

Default shell of the user is mostly defined in the file `/etc/passwd`

```
$ grep pvl /etc/passwd # print lines with word pvl in file passwd
pvl:x:1000:1000:Pavel Fibich,,,:/home/pvl:/bin/bash
```

When shell starts, it runs startup files (distribution specific) to initialize itself.

- `/etc/profile` – system specific settings
- `/etc/bash.bashrc` – shell specific settings
- `~/.profile` – user specific settings
- `~/.bashrc` and `.bash_*` files) – user shell specific settings

If you want to make some user specific changes, do it in `~/.bash_*` or `~/.bashrc` files!

Bash settings * - tunning of shell

Many bash settings is done by `set` or `shopt` commands (running the commands print the list of settings or options). Some useful settings

- do not allow overwriting files

```
$ set -o noclobber # not allow
$ set +o noclobber # allow
```

- `emacs` (default) or `vi` options define differences of command prompt behaviour

```
$ set +o vi
```

- `xtrace` `verbose` set debugging or verbose mode

```
$ set -o xtrace
$ set -o verbose # print command before execution
```

- `dotglob` filenames starting with . or .. do not match using wildcards

```
$ set -o dotglob
```

Bash shortcuts - working faster

- UP and DOWN keys are used for the listing in the past commands
- two TAB keys find matching file name (e.g. write `whoa` and then press TAB twice)
- CTRL + R searches in the history of command line (e.g. `~/.bash_history`), press it and write `w`, then you can press it again for later appearance, or press enter and UP/DOWN
- !CMD runs last matching command CMD from history (!! runs last command), e.g.

```
$ !w
whoami
pvl
$ history  # print history of commands
...
```

- selected text in the shell, can be pasted by middle button of the mouse

Exercise

Exercise

- decide where to work: using linux machine or virtual one
- run shell and try presented commands
- prepare environment where you will work under linux (eg. get used with linux environment - set wifi connection, create directory for the course, make shortcut for running of terminal)
- look briefly at man pages of already mentioned commands
- go through the actual and one future lesson

envir and commands

environment and commands

Users and system info

```
$ id  # print user and group ids (see /etc/group)
uid=1000(pvl) gid=1000(pvl) groups=1000(pvl),24(cdrom)
   ,25(floppy),29(audio),30(dip),44(video),46(plugdev)
   ,105(scanner),110(bluetooth),111(netdev),123(scard)
$ w  # show who is logged on and what they are doing
 09:46:13 up 3 days, 15:21, 9 users, load average: 0.23,
    0.23, 0.23
USER TTY   FROM LOGIN@ IDLE   JCPU   PCPU WHAT
pvl  pts/0 :0.0 Tue19  12:35m 0.05s  0.05s bash
pvl  pts/1 :0.0 Tue20  12:47m 0.34s  0.02s vim disk26
...
$ uname -a  # show machine characteristics
Linux bartsia 3.8-trunk-amd64 SMP Debian 3.8.3-1~
   experimental.1 x86\_64 GNU/Linux
```

Task

Look at `/proc/` files to get more information (e.g. `cat /proc/cpuinfo`)

Printing and editing

```
$ head myfile  # print 10 first lines from file
...
$ tail -n 5 myfile  # print 5 last lines from file
...
$ less myfile  # for reading the file
$ nano myfile  # simple text editor (or use pico)
$ vim myfile  # editor preferred by linux guru ¨‿
```

Shell often have many editors:

nano ▸ www.nano-editor.org - CTRL+x to exit, CTRL+o to write actual state

Emacs ▸ www.gnu.org/emacs/

vi, vim ▸ www.vim.org/

- two states : press **i** (insert) or ESC (command)
- **:wq** to save and exit
- **:!q** to not save and exit

Task

Get familiar with some text editor!

Text manipulation and printing

Files manipulation and info

```
$ sort myfile  # sort file
...
$ uniq myfile  # unique lines
...
$ diff myfile newfile  # differences of files
...
```

Text printing

```
$ echo "hello"  # print text
$ printf "%d\n" 5  # C like print as digit (integer)
5
$ printf "%f\n" 5  # print as float
5.000000
$ printf "There are %d dogs and %d cats.\n" 3 2
There are 3 dogs and 2 cats.
$
```

Exercise

Exercise

- run following command, it will download file `ideff.csv`

```
$ wget http://botanika.prf.jcu.cz/fibich/ideff.csv
```

  or if you do not have wget installed, try `curl`

```
$ curl http://botanika.prf.jcu.cz/fibich/ideff.csv >
    ideff.csv
```

- try to list (print) the file, print only the first 3 lines
- count number of lines of the file
- try to edit the file
- create new directory and copy the file in
- delete the directory
- look briefly at man pages of used commands
- try and read all Tasks

Sequence of commands - background

We can write more commands in one line separated by `;`

```
$ pwd; wc -l myfile; cat myfile
```

We can easily run command in background (append &), or during running of command press CTRL+z to suspend and than write `bg` to move to background or `fg` to move it to foreground

```
$ ls -la &  # run ls in background
[1] 18472
...
[1]+  Done                    ls --color=auto
$ sleep 20 &  # run sleep for 20 s in background
[1] 18511
$ sleep 30 &
[2] 18512
$ jobs  # print jobs in background
[1]-  Running                 sleep 20 \&
[2]+  Running                 sleep 30 \&
$ kill 18512  # force end the job with given PID
```

What to do with the output? - pipe

One of the first options, when we want to combine commands, is the pipe (`|`). It takes output of one command as input for the next command (*redirection between commands*)

```
$ grep cpu /proc/cpuinfo | wc -l
16
$ ls -la | head -n 2  # print first two lines from ls command
total 12
drwxr-xr-x 2 pvl pvl 4096 Jan 29 10:28 .
# who print who is logged, tee send output to std. out and to file
$ who | tee log.txt | grep "2014-02-03"
pvl      pts/4        2014-02-03 20:33 (:0.0)
$ wc -l log.txt  # number of lines in the log.txt
4 log.txt
```

Task

Run commands separately and check their inputs and outputs.

What to do with the output? - redirection

Output redirection to (>) /from (<) file

```
$ ls -la > myfile  # redirect std. output into file
$ ls -la >> myfile  # append std. output at the end of file
$ cat < myfile  # redirects file to the command
```

Commands have also error output

```
$ ls -la /nonsense > myout  # non-existing directory, nothing redirected
$ ls -la /nonsense 2> myout  # good redirecton of error output
$ ls -l * 1> stdout 2> stderr  # separating outputs
```

To combine std. and error output use &>.

Commands also set variable $? (return value of the last command)

```
$ ls -la / &>/dev/null; echo $?
0
$ ls -la /nonsense &>/dev/null; echo $?
2
$
```

Files exercise

Run following commands to generate input for exercise

```
$ mkdir fexer; cd fexer
$ touch {a..e}file; touch {1..26}new
```

Exercise

- count files in the folder
- print reversaly sorted list of files (one per line)
- from the list, print names of the last 4 files and then the first 3 files
- count all files in the folder and files having ”1” in name (e.g. by `grep`)
- delete folder `fexer`
- where it is possible try to use both, redirection and pipe
- go through the actual and future lesson