Be in shell

r-cran, perl,python,exercise

Pavel Fibich

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

04-2015

Přírodovědecká
fakulta
Faculty
of Science

R

R

R introduction

R is 'GNU S', a freely available language and environment for statistical computing and graphics which provides a wide variety of statistical and graphical techniques: linear and nonlinear modelling, statistical tests, time series analysis, classification, clustering, etc. See `http://cran.r-project.org/`.

To run and quite R

```
$ R
R version 3.0.3 (2014-03-06) -- "Warm Puppy"
Copyright (C) 2014 The R Foundation for Statistical
   Computing
Platform: x86_64-pc-linux-gnu (64-bit)
...
> plot(1:10)  # plot number from 1 to 10
> q()  # quit R
Save workspace image? [y/n/c]: n
$
```

R introduction

Quite intuitive environment, in examples

```
> a=4+6  # assign the value to variable
> a  # print variable
[1] 10
> 4:10  # create sequence
[1]  4  5  6  7  8  9 10
> a+4:10  # sum value with sequence
[1] 14 15 16 17 18 19 20
> ls()  # list of current variables
[1] "a"
> rm(a)  # removing the variable
> getwd()  # get current directory, setwd() set it
[1] "/home/pvl/Documents/prf.jcu/bash"
> sqrt(5)  # square root of 5
[1] 2.236068
> seq(3,12,1.2)  # sequence defined by step
[1]  3.0  4.2  5.4  6.6  7.8  9.0 10.2 11.4
> ? seq  # get help for the function
```

R why

Why to know R?

- currently the most widespread statistical software
- great graphical features
- thousands of packages (addons with specialized functions) mostly for everything
- many tools for data manipulation (`grep`, `merge`, `split`, `aggregate`, `apply`, `uniq`, `sort`, `gsub`...)
- power of programming language (variables, conditions, loops, function, classes, regex, easy math, ...)
- binaries for Mac, Windows and Linux
- it is free

Often connected with some integrated development environment (IDE), e.g. R studio (`https://www.rstudio.com/`)

R start

Instead of interactive mode, R can be run with script (see `man R`)

```
$ cat script.r # script with R code
sqrt(2:20)
$ R -q --vanilla < script.r > out.r # run script
$ cat out.r # see the output
> sqrt(2:20)
 [1] 1.414214 1.732051 2.000000 2.236068 2.449490
     2.645751 2.828427 3.000000
...
```

or we can specify interpreter in the script

```
$ cat run.R # R script with interpreter
#!/usr/bin/Rscript
sqrt(2:20)
$ ./run.R # can be run directly in the terminal
 [1] 1.414214 1.732051 2.000000 2.236068 2.449490
     2.645751 2.828427 3.000000
...
```

R start

R often load and store current environment in the file `.RData` (there are stored current variables, function, settings, ...), but it depends which options where specified during the run of R.

- `--no-environ` do not load `.RData`
- `--no-save` do not save `.RData`
- `--vanilla` is shortcut for `--no-save, --no-restore, --no-site-file, --no-init-file` and `--no-environ`

In the interactive mode after quit (`q()`), R is asking if to save environment in to `.RData`. Image can be saved during the sesstion too (`save.image()`).

R packages

Packages are often installed in the personal library in the home directory `/home/$USERNAME/R`.

```
> search()  # what is currently loaded
> library()  # see all installed packages
```

Sometimes it is usefull to tell R where to install and where to get packages (e.g. for package called `fork`)

```
> install.packages("fork", lib="/home/pvl/Documents/prf.
   jcu/bash/r")
> library("fork", lib="/home/pvl/Documents/prf.jcu/bash/r
   ")
```

To install already downloaded package you can use

```
$ wget http://cran.r-project.org/src/contrib/fork_1.2.4.
   tar.gz
$ R CMD INSTALL fork_1.2.4.tar.gz
```

Many of packages are in repositories of OSs.

R data

To get data in, or store them in files, there are `read.*` and `write.*` functions (press TAB after dot to get the list of available function like in terminal)

```
> read.
read.csv read.delim2 read.table read.csv2 read.delim ...
> write.
write.csv write.csv2 write.ftable write.table ...
```

To read csv from the homework 2, you can use

```
> sla = read.csv("sla.csv")  # read csv file
> class(sla)  # check the data type
[1] "data.frame"
> summary(sla)  # see the summary
      species          plot          LEAFAREA_mm2
 FestRubr : 11   Min.   : 1.00   Min.   :   52.37
 CirsPalu :  9   1st Qu.: 6.00   1st Qu.:  793.94
 AgroCani :  8   Median :12.00   Median : 1508.18
 GaliUlig :  8   Mean   :12.53   Mean   : 2252.80
...
```

R data out

You can easily store your results of data manipulations

```
> sla2 = sla  # copy of variable
# to work with column you often use $ after variable name, then col. name
> sla2$LEAFAREA_mm2 = log(sla2$LEAFAREA_mm2 + 1 )
> summary(sla2)  # see the summary
      species           plot          LEAFAREA_mm2
 FestRubr : 11    Min.    : 1.00    Min.    :3.977
 CirsPalu :  9    1st Qu.: 6.00    1st Qu.:6.678
 AgroCani :  8    Median :12.00    Median :7.319
 GaliUlig :  8    Mean   :12.53    Mean   :7.284
...
```

and save it into file

```
# to write csv without rownames and quoting the text
> write.csv(sla2,"sla2.csv",row.names=FALSE,quote=FALSE)
# in write.table you can specify delimiter
> write.table(sla2,"sla2a.csv",row.names=FALSE, quote=
    FALSE,sep=":")
```

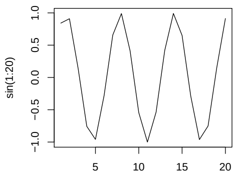
R graphics

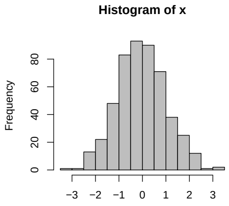
R terminal often open R graphic device when you plot, but you can easily redirect output to the file

```
> ?pdf  # or tiff, jpeg, png or bmp
> pdf("myout.pdf",width=4,height=4)  # graphical device in inches
> plot(sin(1:20),type='l')  # some plot command
> dev.off()  # store the output into file
```

To easily create histogram you can use

```
> ?hist
> x<-rnorm(500)  # 500 random numbers in Gaussian distribution, ?rnorm
> hist(x,col="gray")  # histogram of x in gray color
```





R habits

Working in R, you should

- check help pages (`?command`)
- check data, e.g. for variable `a`

```
> summary(a)
> a  # print content of variable
> dim(a)  # size of variable, e.g. for tables (data.frame, matrix)
> length(a)  # length of variable
> class(a)  # type of variable
```

- be careful with = (or <−) it can destroy your data in memory
- prepare a script with sequence of command (e.g. in RStudio where you can run script directly) for futher re-use
- be careful what is loaded (e.g. use `ls()` to check which variables were loaded from `.RData`)
- use comments (after #) in your code

R

Exercise

- plot histogram of LEAFAREA from the `sla.csv` (homework 2)
- create boxplot (`?boxplot`) of column name `RYO` from `ideff.csv` *per* column name `exp` (one figure with 3 boxes)

```
> boxplot(COLUMNforYaxis~COLUMNforXaxis)
```

- write script (or function) that will have 2 arguments (1. is filename), script will run R, that will do histogram of the column (2. argument), result will be saved into some graphical file (eg. add pdf suffix for the input filename)
- try to create the same plot as in GNUplot example based on `using.dat`

R

Exercise - solution

- write script (or function) that will have 2 arguments ...

```
$ cat rscr # R script version
#!/usr/bin/Rscript
args <- commandArgs(trailingOnly = TRUE)
mf=read.csv(args[1])
hist(mf[,args[2]])
$ ./rscr ideff.csv RYO
$ cat bscr # bash script version
echo "mf=read.csv(\"$1\")" > rfile
echo "pdf(\"our.pdf\")" >> rfile
echo "hist(mf\$$2)" >> rfile
echo "dev.off()" >> rfile
R --vanilla < rfile
$ ./bscr ideff.csv RYO
```

perl

Practical Extraction and Report Language alias `perl` is complete language with many adjectives introduced by Larry Wall in 1987. Perl uses syntax and concepts of `awk`, `sed`, `C`, `bash`, ... It stands on thousands of third-party modules stored in the repository Comprehesive Perl Achive Network (CPAN).

```
$ cat hello.pl  # perl files often end with .pl
#!/usr/bin/perl
print "Hello World.\n";
$ ./hello.pl
Hello World.
$ perl hello.pl  # other way
$ perl -d hello.pl  # to use perldebug
```

For basic help and man pages use

```
$ perl --help
$ man perl
```

perl characteristic

- was meant to be a sort of shell-script on steroids (condense syntax)
- large library support
- many OS and it is free
- powerful text processing facilities without the arbitrary data-length limits of many contemporary Unix commandline tools
- actual version 5.18.2 (January 7 2014)

Programming language features

- various variable types
- OO
- ability to package code in reusable modules
- automatic memory management

perl modules

The easy way to install perl module (library) is through CPAN (first run often pre-set environment)

```
$ perl -MCPAN -e shell
cpan> install HTML::Template
cpan> quit
```

It often pre-set ~/.cpan/ and install libraries in the ~/perl5/.
Manual way of install package is done by getting code of module from http://search.cpan.org/, e.g.

```
$ wget http://search.cpan.org/CPAN/authors/id/W/WO/WONKO/
    HTML-Template-2.95.tar.gz
$ tar -xvf *.tar.gz
$ cd HTML-Template-2.95
$ perl Makefile.PL
$ make; make test
$ make install
```

You can also specify prefix where to install module (PREFIX=/folder after Makefile.PL).

perl modules, example

To check if module is properly installed, you can by

```
$ export PERL5LIB=~/perl5/lib/perl5 # export PATH to perl modules
$ perl -e "use HTML::Template" # run script that load module
# if the output is no error, that in is ok
```

Many of packages are also in standard operating systems repositories (e.g. `apt-cache search perl MODULENAME`).

Few examples

```
# print first 3 columns
$ perl -pale '$_="@F[0..2]"' using.dat
# change GE for GL in the file and store the original
$ perl -pi'.orig' -e 's/GE/GL/' ideff.csv
$ cat b64.pl # script to decode argument from Base64 code
#!/usr/bin/perl
use MIME::Base64;
print decode_base64($ARGV[$1])
```

python

Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms. See `https://www.python.org`.

```
$ cat hello.py    # python sctipt often end with .py
#!/usr/bin/python
print 'Hello, world!'
$ ./hello.py # run python script
Hello, world!
$ python hello.py # other way of run
$ python -v hello.py # to debug what is loading during the execution
```

For basic help and man pages use

```
$ python --help
$ man python
```

python characteristic

- elegant syntax
- large library support
- easy extendable by modules, even in C or C++
- many OS and it is free
- ideal for prototype development and other ad-hoc programming tasks
- actual version 3.4.0 (March 17 2014)

Programming language features

- variety of basic variable types
- OO, generator and list comprehensions
- code in modules or packages
- supports exceptions
- automatic memory management

python packages

To install python package you can use easy ways by `pip` or `easy_install`

```
$ sudo python get-pip.py # you must download get-pip
# previous command is run only once
$ pip search PACKAGENAME # to search package
$ sudo pip install bioinfo --process-dependency-links
# install bioinfo package with dependencies
```

To chek if it was successful

```
$ python -c "import bioinfo;" # run script that load the package
```

The more manual way on already downloaded package

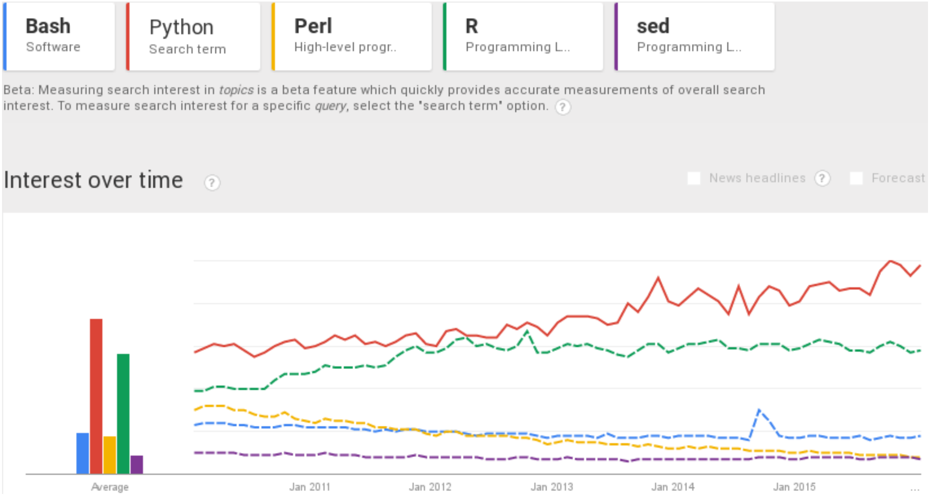
```
# extract the package, get in directory
$ TOIN=/software/EXPECTED_FOLDER # install folder
$ python setup.py install --install-scripts=$TOIN/bin/ --
    install-purelib=$TOIN/lib --install-lib=$TOIN/lib
```

Later, sometimes it is necessary to set `PYTHONPATH` and `PATH` variables to `$TOIN/lib` and `$TOIN/bin`. Many modules are also in OS repositories.

python examples

```
$ cat name.py # get input from user
#!/usr/bin/python
name = raw_input('What is your name?\n')
print 'Hi, %s.' % name
$ ./name
What is your name?
pvl
Hi, pvl.
$ cat sum.py # sum of integer arguments
#!/usr/bin/python
import sys
try:
    total = sum(int(arg) for arg in sys.argv[1:])
    print 'sum =', total
except ValueError:
    print 'Please supply integer arguments'
$ ./sum.py 3 4 5
sum = 12
```

tools - google trends



Beta: Measuring search interest in *topics* is a beta feature which quickly provides accurate measurements of overall search interest. To measure search interest for a specific *query*, select the "search term" option.

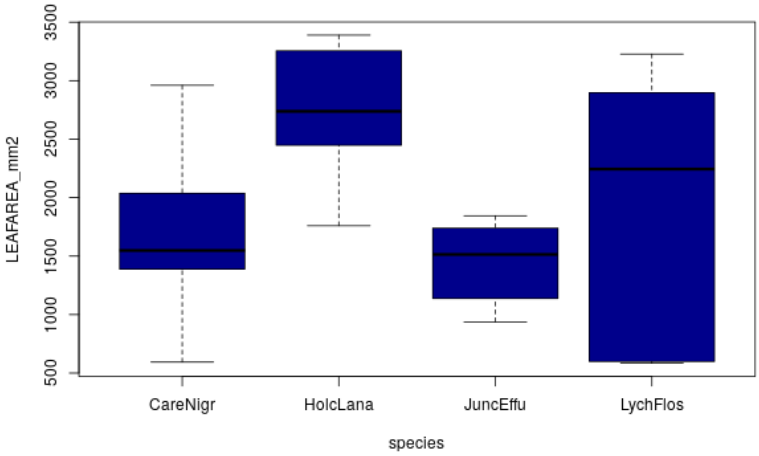
Homework 3

Download
`http://botanika.prf.jcu.cz/fibich/bash/sla.csv`.

- Write a script that have two arguments: filename (eg. sla.csv) and non-negative number (eg. X).
- Script will create one picture (whatever format pdf, jpg, tiff, png, ...), but having the same name as the first argument (e.g. it will create sla.pdf), where one can observe difference of LEAFAREA (the third column in the file) of different species (name from the first column of the file).
- For the plotting, we use ONLY species that have exactly X (2. argument of the script) occurences in the file.
- Send script (name according your surname: `plotdif.SURNAME`) by email until 24:00 8.5.
- For the example see following slide (it is not necessary to use boxplot).

Homework 3

```
$ plotdif.sh pic.csv 6 # create figure pic.pdf for species with exactly 6
      occurences
```



big exercise 2

big exercise 2

Big Exercise

Exercise

Write one line command that will

- check if the number of files in the current directory is bigger than 5
- set variable to the names of all files in the current directory
- print current year, month and day, in format, eg. `2014 Jan Mon`
- create directories from 1 to 99, each with the file `no` containing directory number plus 130
- create variable `evenNAME` containing names of directories with even number
- create variable `oddNO` containing number of directories with odd number
- delete directories containing 0 in the name
- for file `test` print number of lines having some upper case character

Big Exercise

Exercise

Write a function that will

- make its only one argument executable file, add argument checking
- print the last and the first line of the file given as the argument
- count lines that do not contain character specified as 1. argument in the file specified as 2. argument
- inform you if the seconds of the current time are more or less than 30
- run `date` command only if there are more than 3 files in the current folder
- substitute string (first argument) for string (second argument) in the file (third argument)
- remove all white spaces from the argument (eg. output of `date`)

Big Exercise

Exercise

Write a script or function that will

- check if varible `MYVAR` is in range 10-30 and inform about it
- create files `myfileX`, where X will be changed for all small case letter in the alphabet
- for aguments (always one arg.) like: `xxfileRRTT34`, `yzrrrru33`, remove and print strings without 3-6. characters
- create file every 3 seconds, name will be according to the seconds of the current time
- print `sla.csv` file with log transformed last column
- print range of numbers between lines (1. and 2. argument) from the file (3. arg)

Big Exercise

Exercise – regex

- write function that for aguments (there will be always only one argument) like: `abcd34g`, `a1`, `uhrfdsa355jj` print lengths of characters before digits (digits have various length) and after digits; so 2 numbers appear as the output, e.g. `4 1`, `1 0`, `7 2`
- write function or script that for arguments (there will be always only one argument) like `1232fa33a`, `2errerew9bb`, `11a9ere` will print difference of the numbers in the argument, e.g. `1199`, `-7`, `2`

```
$ mydiff 1232fa33a
1199
$
```

- write function or script that for arguments (there will be always only one argument) like `1232fa33a`, `2errerew9bb`, `11a9ere` will print only characters without digits at the beginning and also print theirs length, e.g. `fa33a 5`, `errerew9bb 10`, `a9ere 5`