Using R as a scripting language

Hywel Dunn-Davies
Tollervey lab

Wellcome Trust
Centre for Cell Biology

Background

- I work as a Bioinformatician in the Tollervey lab in Edinburgh University
- Prior to joining the lab I worked mainly in R / Bioconductor, within RStudio
- Now my main job is to automate the analysis of Next Generation Sequencing data using a combination of new scripts and existing programs
  - Most existing programs are written in scripting languages such as perl, python, and awk, and are run from the Linux command line

Motivation for writing command line scripts in R

- To make R functionality available to other members of the lab without them having to learn a new platform
- To link R functions with scripts written in languages such as perl and python in analysis pipelines that can be run automatically
- Using R to create scripts that can be run directly from the Linux command line makes these things possible

Overview

1. How to run an R script from the command line
2. How to pass command line arguments to an R script
3. Parsing command line arguments with optparse
4. Working with UNIX pipes

1. How to run an R script from the command line with Rscript

- Rscript, introduced in R version 2.5.0, provides a front end to R allowing it to be used to create command line scripts
- To run a simple R script from the Linux / Mac OS X command line using Rscript, either:
  - Type ***Rscript <script_name>***
- or:
  - Include ***#!/usr/bin/env Rscript*** as the first line of the script
  - Change the permissions using ***chmod u+x <script_name>***
  - Run the script by typing its name

Simple example:
Running a hello world script

- Running the program by calling Rscript explicitly:

```
sce-bio-c03583:Rscript_demo hywelrdd$ cat > ./hello_world
cat("Hello world!\n")
sce-bio-c03583:Rscript_demo hywelrdd$ Rscript ./hello_world
Hello world!
sce-bio-c03583:Rscript_demo hywelrdd$
```

- Running the program by calling it directly:

```
sce-bio-c03583:Rscript_demo hywelrdd$ cat > ./hello_world
#!/usr/bin/env Rscript
cat("Hello world!\n")
sce-bio-c03583:Rscript_demo hywelrdd$ chmod u+x ./hello_world
sce-bio-c03583:Rscript_demo hywelrdd$ ./hello_world
Hello world!
sce-bio-c03583:Rscript_demo hywelrdd$
```

More complex example: Adding a column to a dataframe

- This script adds a summary column to a tab separated file and saves it to a new file
  - Input:

```
sce-bio-c03583:Rscript_demo hywelrdd$ head -3 petals.tsv
Petal.Length    Petal.Width
1.4     0.2
1.4     0.2
```

  - Transformation function (add_summary_column_function.R):

```
1 add_summary_column <- function(df, add_average=FALSE) {
2     if(add_average==TRUE) {
3         df$average = (df$Petal.Length + df$Petal.Width) / 2
4     } else {
5         df$sum = df$Petal.Length + df$Petal.Width
6     }
7     return(df)
8 }
```

More complex example:
Adding a column to a dataframe

```
1  #!/usr/bin/env Rscript
2  source("add_summary_column_function.R")
3
4  input.filepath <- "./petals.tsv"
5  output.filepath <- "./petals_with_sum.tsv"
6
7  df <- read.delim(input.filepath, header=TRUE)
8  df_2 <- add_summary_column(df, add_average=FALSE)
9  write.table(df_2, file=output.filepath, sep="\t", col.names=TRUE, row.names=FALSE, quo
   te=FALSE)
10
~
~
~
~
~
~
~
~
~
~
~
~
~
~
NORMAL  add_sum_column_non_interactive +        unix | utf-8 | conf  100%  LN  10:1
```

More complex example: Adding a column to a dataframe

- Running the script from the command line as before loads in the input file, adds a column and saves the result to the output file

```
sce-bio-c03583:Rscript_demo hywelrdd$ ./add_sum_column_non_interactive
sce-bio-c03583:Rscript_demo hywelrdd$ head -3 petals.tsv
Petal.Length    Petal.Width
1.4     0.2
1.4     0.2
sce-bio-c03583:Rscript_demo hywelrdd$ head -3 petals_with_sum.tsv
Petal.Length    Petal.Width     sum
1.4     0.2     1.6
1.4     0.2     1.6
```

- Both input filepath and output filepath are hardcoded

2. How to pass command line arguments to an R script

- The simple way to access the command line arguments passed to an R script is to use the commandArgs variable:
  - ***commandArgs(trailingOnly = TRUE)***
  - Setting trailingOnly to TRUE ensures you only get the arguments that are explicitly passed to your script by the user, leaving out other arguments automatically added by Rscript (such as the name of the script file and the path to the R executable)

Example: Adding commandArgs to our script

```
 1 #!/usr/bin/env Rscript
 2 source("add_summary_column_function.R")
 3
 4 args <- commandArgs(trailingOnly=TRUE)
 5
 6 input.filepath=args[1]
 7 output.filepath=args[2]
 8
 9 if(length(args)==3 & args[3]=="-a") {
10     add.average=TRUE
11 } else {
12     add.average=FALSE
13 }
14
15 df <- read.delim(input.filepath, header=TRUE)
16 df_2 <- add_summary_column(df, add_average=add.average)
17 write.table(df_2, file=output.filepath, sep="\t", col.names=TRUE, row.names=FALSE, quo
   te=FALSE)
18
~
~
~
NORMAL  add_summary_column_commandArgs +        unix | utf-8 | conf  100%  LN  18:1
```

Example: Adding commandArgs to our script

```
sce-bio-c03583:Rscript_demo hywelrdd$ ./add_summary_column_commandArgs ./petals.tsv ./peta
ls_with_sum_commandArgs.tsv
sce-bio-c03583:Rscript_demo hywelrdd$ head -3 ./petals_with_sum_commandArgs.tsv
Petal.Length    Petal.Width     sum
1.4     0.2     1.6
1.4     0.2     1.6
sce-bio-c03583:Rscript_demo hywelrdd$ ./add_summary_column_commandArgs ./petals.tsv ./peta
ls_with_average_commandArgs.tsv -a
sce-bio-c03583:Rscript_demo hywelrdd$ head -3 ./petals_with_average_commandArgs.tsv
Petal.Length    Petal.Width     average
1.4     0.2     0.8
1.4     0.2     0.8
sce-bio-c03583:Rscript_demo hywelrdd$
```

- Using this script we use command line arguments to allow the user to:
  - read from and write to arbitrary files
  - specify whether to add the sum or the average

Drawbacks of commandArgs

- No automatic support for non-positional arguments
- No automatic parsing of options
- No automatic help. To see which arguments need to be passed to the script you need to look at the code
- No validation of input values

3. Parsing command line arguments with optparse

- optparse is an R package, modelled on python's optparse package, which has a number of useful functions:
  - It automatically parses command line options, and can handle options specified in an arbitrary order
  - It performs input validation
  - It automatically generates a help menu, allowing the user to find out how to use the script without looking at the source code

Example: Adding optparse to our script

```
#!/usr/bin/env Rscript
source("add_summary_column_function.R")

suppressWarnings(suppressPackageStartupMessages(library("optparse")))

option_list <- list(
  make_option(c("-i", "--input_filepath"), help="The path to the input table."),
  make_option(c("-o", "--output_filepath"), help="The path to the output table."),
  make_option(c("-a", "--add_average_column"), action="store_true", default=FALSE, help="Flag to specify that a column showing the average of Petal.Length and Petal.Width, should be added. Default is false, indicating that the added column should represent the sum of Petal.Length and Petal.Width.")
)
opt <- parse_args(OptionParser(option_list=option_list, description="Adds a column to the table given as input aggregating the Petal.Length and Petal.Width columns."))

df <- read.delim(opt$input_filepath, header=TRUE)
df_2 <- add_summary_column(df, add_average=opt$add_average_column)
write.table(df_2, file=opt$output_filepath, sep="\t", col.names=TRUE, row.names=FALSE, quote=FALSE)
~
~
NORMAL  add_summary_column_optparse          unix | utf-8 | conf   6%  LN  1:1
```

Example: Adding optparse to our script

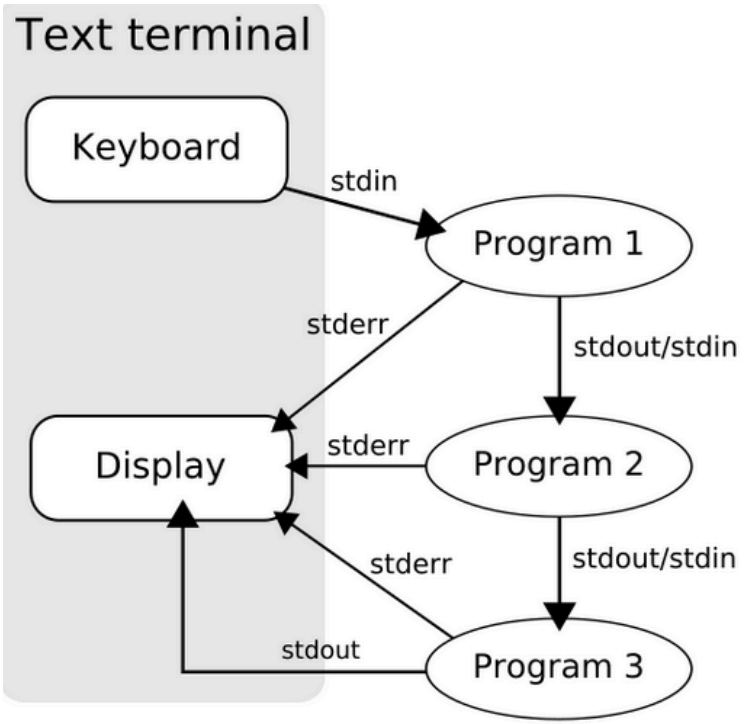
```
sce-bio-c03583:Rscript_demo hywelrdd$ ./add_summary_column_optparse -h
Usage: ./add_summary_column_optparse [options]
Adds a column to the table given as input aggregating the Petal.Length and Petal.Width col
umns.

Options:
        -i INPUT_FILEPATH, --input_filepath=INPUT_FILEPATH
                The path to the input table.

        -o OUTPUT_FILEPATH, --output_filepath=OUTPUT_FILEPATH
                The path to the output table.

        -a, --add_average_column
                Flag to specify that a column showing the average of Petal.Length and Peta
l.Width, should be added. Default is false, indicating that the added column should repres
ent the sum of Petal.Length and Petal.Width.

        -h, --help
                Show this help message and exit


sce-bio-c03583:Rscript_demo hywelrdd$
```

Example: Adding optparse to our script

```
sce-bio-c03583:Rscript_demo hywelrdd$ ./add_summary_column_optparse -i ./petals.csv -o ./p
etals_with_average.tsv -a average
Error in getopt(spec = spec, opt = args) :
  "average" is not a valid option, or does not support an argument
Calls: parse_args -> getopt
Execution halted
sce-bio-c03583:Rscript_demo hywelrdd$ ./add_summary_column_optparse -i ./petals.csv -o
Error in getopt(spec = spec, opt = args) : flag "o" requires an argument
Calls: parse_args -> getopt
Execution halted
sce-bio-c03583:Rscript_demo hywelrdd$ ./add_summary_column_optparse -o ./petals_with_avera
ge.tsv -a -i ./petals.tsv
sce-bio-c03583:Rscript_demo hywelrdd$ head -3 petals_with_average.tsv
Petal.Length    Petal.Width     average
1.4     0.2     0.8
1.4     0.2     0.8
sce-bio-c03583:Rscript_demo hywelrdd$
```

4. Adding support for UNIX pipes

- UNIX pipes are used to link together many UNIX and Linux command line utilities via text streams
- The two main text streams are standard input (stdin), specifying the input to the program, and standard output (stdout), specifying its output
- Syntax: program_1 | program_2 | program_3



"Pipeline" by TylzaeL - en:File:Pipeline.svg. Licensed under Public Domain via Commons - https://commons.wikimedia.org/wiki/File:Pipeline.svg#/media/File:Pipeline.svg

4. Adding support for UNIX pipes

- In R:
  - file("stdin") represents standard input
  - A range of commands exist for writing to standard output:
    - cat
    - write
    - print
    - …
- For pipes to work correctly, care must be taken to ensure that text printed to standard output is formatted correctly

Example: Adding UNIX pipe support

```
1  #!/usr/bin/env Rscript
2  source("add_summary_column_function.R")
3
4  suppressWarnings(suppressPackageStartupMessages(library("optparse")))
5
6  option_list <- list(
7    make_option(c("-i", "--input_filepath"), help="The path to the input table. Leaving
   out this option tells the script to read from standard input."),
8    make_option(c("-o", "--output_filepath"), help="The path to the output table. Leavin
   g out this option tells the script to write to standard output."),
9    make_option(c("-a", "--add_average_column"), action="store_true", default=FALSE, hel
   p="Flag to specify that a column showing the average of Petal.Length and Petal.Width,
   should be added. Default is false, indicating that the added column should represent t
   he sum of Petal.Length and Petal.Width.")
10 )
11 opt <- parse_args(OptionParser(option_list=option_list, description="Adds a column to
   the table given as input aggregating the Petal.Length and Petal.Width columns."))
12
13
14
15
16
NORMAL  add_summary_column_pipes                unix | utf-8 | conf    2%  LN  1:1
```

Example: Adding UNIX pipe support

```
15
16
17 if (is.null(opt$input_filepath)) {
18     input.filepath=file("stdin")
19 } else {
20     input.filepath=opt$input_filepath
21 }
22
23 df <- read.delim(input.filepath, header=TRUE)
24 df_2 <- add_summary_column(df, add_average=opt$add_average_column)
25
26 if (is.null(opt$output_filepath)) {
27     cat(c(paste0(colnames(df_2),collapse="\t")), "\n")
28     for (row in rownames(df_2)) {
29         cat(c(paste0(df_2[row,],collapse="\t")), "\n")
30     }
31 } else {
32     write.table(df_2, file=opt$output_filepath, sep="\t", col.names=TRUE, row.names=FA
   LSE, quote=FALSE)
33 }
34
35
NORMAL  add_summary_column_pipes          unix | utf-8 | conf  100%  LN  35:1
```

Example: Adding UNIX pipe support

```
sce-bio-c03583:Rscript_demo hywelrdd$ head -5 petals.tsv
Petal.Length    Petal.Width
1.4     0.2
1.4     0.2
1.3     0.2
1.5     0.2
sce-bio-c03583:Rscript_demo hywelrdd$ head -5 petals.tsv | ./add_summary_column_pipes -a
Petal.Length    Petal.Width     average
1.4     0.2     0.8
1.4     0.2     0.8
1.3     0.2     0.75
1.5     0.2     0.85
sce-bio-c03583:Rscript_demo hywelrdd$ head -5 petals.tsv | ./add_summary_column_pipes -a |
 ./add_summary_column_pipes
Petal.Length    Petal.Width     average sum
1.4     0.2     0.8     1.6
1.4     0.2     0.8     1.6
1.3     0.2     0.75    1.5
1.5     0.2     0.85    1.7
sce-bio-c03583:Rscript_demo hywelrdd$
```

Summary

- I've shown how to create command line scripts in R using Rscript and optparse
- This isn't the only way to do it. Other tools exist for both R scripting and option parsing, e.g.
  - Littler (http://dirk.eddelbuettel.com/code/littler.html) is a precursor to Rscript with similar functionality. R can also be run from the command line in batch mode (R CMD BATCH)
  - docopt (https://github.com/docopt/docopt.R) and argparse (https://github.com/trevorld/argparse) provide alternative approaches to option parsing