Adding R functions to Galaxy

Dan Tenenbaum

October 2, 2012

Contents

1 Introduction

Galaxy (`http://galaxy.psu.edu/`) is an open, web-based platform for data intensive biomedical research. It provides an easy-to-use web interface and can expose bioinformatics workflows written in any programming language.

Normally, in order to expose new functionality (a tool in Galaxy parlance) in a Galaxy instance, you have to manually create an XML file with information about the function, and modify an additional XML file.

The *RGalaxy* package automates this process, pulling most of the necessary information from the function itself and its manual page (you provide the remaining information as arguments to the `galaxy` function).

2 Best Practices For Writing 'Galaxifiable' Functions

A function that you wish to expose in Galaxy has certain limitations.

- The function's arguments must include the names of its input and output files (function return values are ignored).
- Any error conditions should be handled with `stop` with a useful/informative error message. The Galaxy user will see these messages if an error occurs.

- Functions which take datasets as input should accept as arguments the filenames pointing to those datasets. The Galaxy user interface will allow the user to choose the dataset graphically.
- Return values of functions are ignored. Function output should be written to one or more files, and the names of these files should be passed into the function as arguments.
- Functions should be documented with a manual page. `galaxy` will use this manual page to fill in relevant sections of the Galaxy XML file. The following sections of the man page are required:
  - **alias**
  - **title**
  - **description**
  - **arguments** - each argument must be documented
  - **details**

The following example illustrates these best practices (this function is in the *RGalaxy* package under the name `functionToGalaxify`:

```
function (inputfile1 = GalaxyInputFile(), inputfile2 = GalaxyInputFile(),
    plotTitle = character(), plotSubTitle = "My subtitle", outputfile1 = GalaxyOutput("mydata",
        "csv"), outputfile2 = GalaxyOutput("myplot", "pdf"))
{
    data1 <- tryCatch({
        as.matrix(read.delim(inputfile1, row.names = 1))
    }, error = function(err) {
        stop("failed to read first data file: ", conditionMessage(err))
    })
    data2 <- tryCatch({
        as.matrix(read.delim(inputfile2, row.names = 1))
    }, error = function(err) {
        stop("failed to read second data file: ", conditionMessage(err))
    })
    data3 <- data1 + data2
    write.csv(data3, file = outputfile1)
    pdf(outputfile2)
    if (missing(plotTitle))
        plotTitle <- ""
    plot(data3, main = plotTitle, sub = plotSubTitle)
    dev.off()
}
<environment: namespace:RGalaxy>
```

This is a trivial function that reads in two matrices from tab-delimited text files, adds them, writes the result to a text file and plots it to a PDF file.

3 Exposing a package in Galaxy

Because of the constraints above, most existing R functions will not immediately work in Galaxy; you'll need to write a simple wrapper function to expose the functionality. If your wrapper function is part of a package (and you want the functions in that package to be available to your wrapper), set the `package` parameter in your call to `galaxy` to be the name of the package. If your wrapper is exported (in your NAMESPACE), set the `exported` parameter to `TRUE`, otherwise set it to `FALSE`.

4 Function Dependencies

You are responsible for making sure that the machine where Galaxy is running (which may be different from the machine where you run *RGalaxy*) has all the appropriate dependcies installed. This includes R packages as well as any external programs. While *RGalaxy* itself is not required on this machine, its dependency `optparse` is required so that the wrapper R script generated by *RGalaxy* can properly parse its command-line arguments.

5 Installing and Starting Galaxy

Refer to the page `http://wiki.g2.bx.psu.edu/Admin/Get%20Galaxy`.

Follow the steps for installing and starting Galaxy. In a nutshell:

```
hg clone https://bitbucket.org/galaxy/galaxy-dist/
cd galaxy-dist
./run.sh --reload
```

Heed the tip about starting Galaxy with the "–reload" option so that it can be easily stopped (with control-C) and restarted.

6 Adding a function to Galaxy

Now that you have written a function that follows the best practices described above, you can make it available to Galaxy as follows:

```
> galaxy(functionToGalaxify,
+     manpage="functionToGalaxify",
+     version="0.99.0",
+     name="Add",
+     package="RGalaxy",
+     galaxyConfig=GalaxyConfig(getwd(), "mytool", "Test Section",
+         "testSectionId")
+     )
```

7 About This Example

Most of the information Galaxy needs is in the function and its manual page; here we point to those two items, and tell *RGalaxy* where Galaxy is installed. We also include version information and describe how the resulting tool in Galaxy can be found.

8 Did it work?

Let's check whether it worked. If Galaxy is running, interrupt it with control-C. (with the default local installation, Galaxy will not automatically detect when a new tool has been added; it must be restarted). Restart Galaxy like so:

```
./run.sh --reload
```

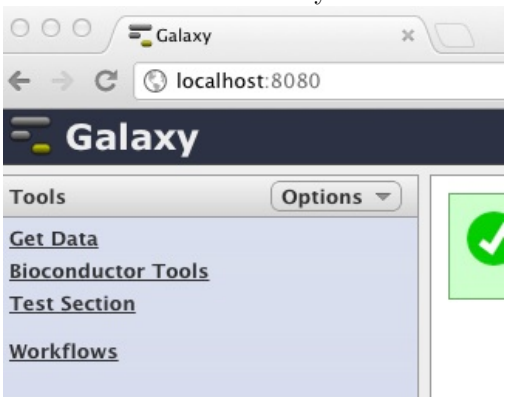
You can then go to Galaxy in your web browser at the URL `http://localhost:8080`.

First you need to upload some sample data sets to Galaxy.

In R, you can obtain the path to the data sets as follows:

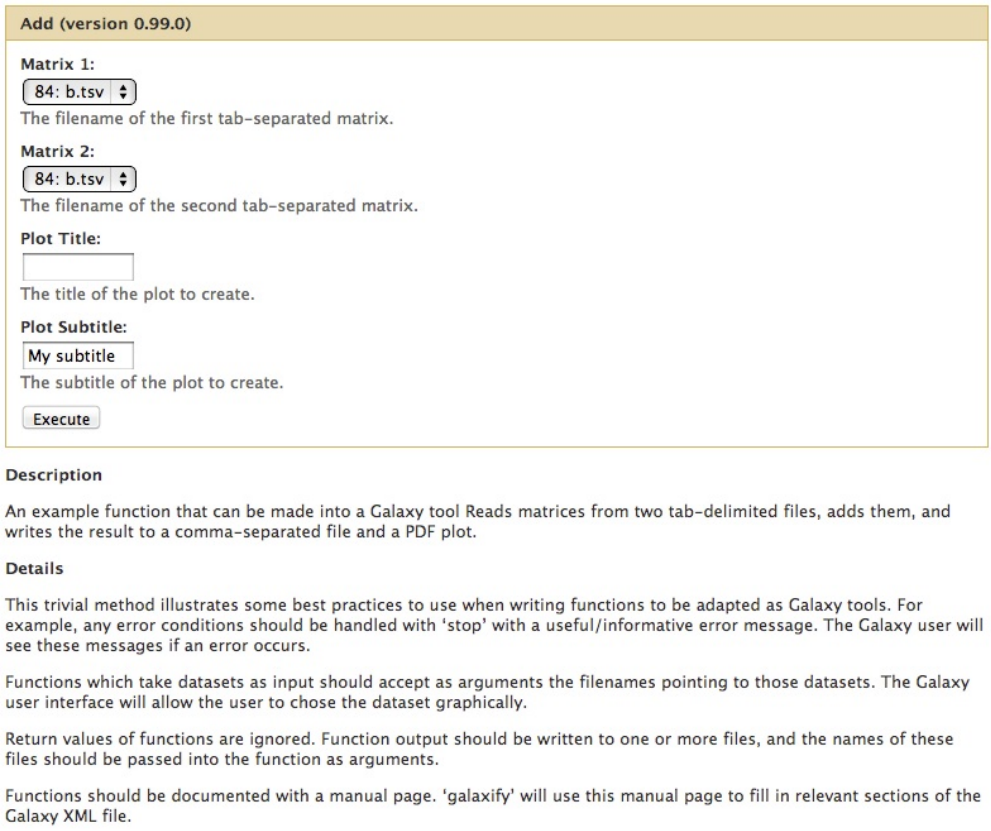
```
> system.file("extdata", "a.tsv", package="RGalaxy")
> system.file("extdata", "b.tsv", package="RGalaxy")
```

You can then upload these data sets to Galaxy by clicking Get Data, then Upload File. You only need to do this once for each Galaxy installation.



The section we added above, called "Test Section", is visible (note that many tools have been removed from this Galaxy installation for clarity).

If we click on "Test Section", then we see the tool that has been created:



Most of this information came from our function and its man page.

If your function or its man page changes, just run `galaxy` again. You will need to restart Galaxy to see the changes.

9 Sharing Your Galaxy Tool

To share your Galaxy tool with a wider audience, you need to move it to a production instance of Galaxy. The files you'll need to move are GALAXY_HOME/tool_conf.xml (or maybe just the section that was created by `galaxy`), and the directory with your tool files (in our example, that was GALAXY_HOME/tools/mytool).

10 A More Useful Example

Suppose you have some Affymetrix probe IDs and you want to look up the PFAM and SYMBOL names for them. It's quite easy to write a function to expose this in Galaxy:

```
> probeLookup <- function(probe_ids=character(),
+     outputfile=GalaxyOutput("probeLookup", "csv"))
+ {
+     ## probe_ids can be something like "1002_f_at 1003_s_at"
+     library(hgu95av2.db)
+     ids <- strsplit(probe_ids, " ")[[1]]
+     results <- select(hgu95av2.db, keys=ids, cols=c("SYMBOL","PFAM"),
+         keytype="PROBEID")
+     write.csv(results, file=outputfile)
+ }
```

Then, you can document the function with a man page:

```
> system.file("extdata", "probeLookup.Rd", package="RGalaxy")
```

...you can expose the function in Galaxy as follows:

```
> galaxy(probeLookup,
+     version="0.0.1",
+     manpage=system.file("extdata", "probeLookup.Rd", package="RGalaxy"),
+     galaxyConfig=GalaxyConfig(galaxyHome, "probeTool", "Test Section",
+         "testSectionId")
+     )
```