1. Control Files

   `ID: org_gcr_2017-05-12_mara:3F5D7D24-91F0-4CBD-A514-C9D0B9821BDA`

   (a) Configuration

   `ID: org_gcr_2017-05-12_mara:091465F6-8E3F-4736-9BF0-A498785F5E1D`

   The goal here is to get R set up thoughtfully and quickly. The profile and environ set it up correctly. Always remember this and run R using those configuration files.

   - Steps
     - Review and Run `.Rsetup.sh` prepare the OS for R and R for your configuration
     - Review `.Renviron` to verify it is what you expect
     - Review `.Rprofile` to recall how you set it up
     - Review and Source `Rinstall` to install required packages
       * Verify the lib path is in the home directory `.libPaths()`
       * When R starts, it won't find any of the required packages. It will complain a lot. That is OK. Just keep running the install script until it is happy: `source("~/.Rinstall.r")`. Then restart R and everything should be fine.
       * Run the installation script 3-5 times, each time
         · Check the version that is running
         · Go back and read the transcript and make sure that it looks right
         · Build warnings
         · Configuration errors
         · Mask warnings
         · Package notifications
         · For example

```
data.table 1.9.6  For help type ?data.table or https://github.com/Rdatatable/
The fastest way to learn (by data.table authors): https://www.datacamp.com/co
```

         · ...

```
Attaching package: ‘magrittr’

The following objects are masked from ‘package:testthat’:

    equals, is_less_than, not
```

         · ...

```
--------------------------------------------------------------------------------
data.table + dplyr code now lives in dtplyr.
Please library(dtplyr)!
--------------------------------------------------------------------------------
```

- Eventually that script will fail, for example
- there is no package called ‘devtools’
- Called `install.packages("devtools")` manually
- Install order error
- After running it a couple of times, it is finished

.Rsetup

```
«rsetup-defs»
```

.Rprofile

```
## -*- mode: R; -*-
«rprofile-def»
.First <- function() {
    gcr <- new.env()
    «rfirst-defs»
    base::attach(gcr, name="gcr", warn.conflicts=FALSE)
}
fortune()
```

.Renviron

```
# -*- mode: sh; -*-
«renviron-def»
```

.Rinstall

```
«rinstall-def»
```

(b) .Rsetup

```
header-args: :noweb-ref rsetup-defs
ID: org_gcr_2017-05-12_mara:898031D5-9930-420A-9DDD-2F5FA0AA6D63
```

Prepare the operating system to host R.

Recreate the packages directory.

```
rm -rf ~/.Rpackages
mkdir ~/.Rpackages
```

Symbolically link all of the control files.

```
rm ~/.Rprofile
ln -sfn ~/src/help/.Rprofile ~/.Rprofile
rm ~/.Renviron
ln -sfn ~/src/help/.Renviron ~/.Renviron
rm ~/.Rinstall.r
```

```
ln -sfn ~/src/help/.Rinstall.r ~/.Rinstall.r
```

(c) .Rprofile [1]

```
header-args: :noweb-ref rprofile-def
ID: org_gcr_2017-05-12_mara:F665989E-724D-4983-A8B1-29F566291722
```

- When you install packages, R needs to know which repository it should use. If you don't tell it, then it will ask you every time. It is just doing its job. Make it easier for yourself and specify a repo for once and for all.
  - Via [2]
  - Built-in docs explain that `local` should be used here
  - This could also be a one-liner: `options("repos" = c(CRAN = "http://cran.r-project.org/")`

```
local({
    r = getOption("repos")
    r["CRAN"] = "https://cran.r-project.org/"
    options(repos = r)
})
```

- By default, hitting enter in the `browser` will submit a `c` for "continue execution at the next statement"
  - It is too easy to hit enter when you didn't mean it
  - It just *feels* imprecise
  - Never let this happen, disable that feature

```
options(browserNLdisabled = TRUE)
```

Show timestamps to 3 sub-seconds:

```
options("digits.secs"=3)
```

Do not allow automatic coercion of strings into factors, as you can specify this by argument to make it *real* obvious. Looks like the best way is to leave it alone globally and always do it by hand though. hwickam commented that it is bad, bad idea to make this option global not because of your code, but because of everyone else's that you are using which relies on the option being set to `TRUE`. Learning more about this, before this was an option, it was the default behavior (being true) because statisticians rarely dealth with character arrays. As the popularity of R skyrocketed, suddently people wanted to work with them a lot. It isn't custom code that expects it to be true, rather, it is library code that expects it to true. Definitely something that you don't want to mess with.

```
options(stringsAsFactors=TRUE)
```

This might be *too much*, but always show a call stack when **any** warnings or errors occur

```
options(showWarnCalls=TRUE)
options(showErrorCalls=TRUE)
```

[1] `https://stackoverflow.com/questions/1189759/expert-r-users-whats-in-your-rprofile`
[2] `http://www.r-bloggers.com/installing-r-packages/`

Be highly conservative about errors and warnings: handle the former immediately and cause the latter to be errors. However, only do this after your workspace has initialized correctly. Do so too soon and most things won't work because this approach is only to handle *my* issues. Imagine of the whole work just handled their own issues! Anway, the safe values are set here, leave them alone. Making them more aggressive will break your startup. When you are ready to set things to be more aggressive, turn it on yourself

```
options(error=NULL)
options(warn=0)
```

Don't print more than 500 lines. If you can grok more than 500 lines then please teach me. Be at ease, there is a helper to remove that restriction, just in case.

```
options(max.print=500)
```

Partial matching is a neat and flexible feature for objects. In theory, it is quite powerful and convenient. In practice it seems like a really bad idea to me. It is a *personal preference*. It only makes sense from that perspective. This could bork 3rd party code.

```
options(warnPartialMatchDollar = TRUE)
```

Locale:

- Make sure that the language is set correctly. I couldn't find anything specific about setting it this way other than various posts. In practice you would really put all of this in your system environment configuration, but I'm wanting to be a little more particular here because it affects operations on data structures, in particular sorting.
- Error messages are mostly useful when they are displayed in English, so make sure that the locale is always English [3]. "Note that the LANGUAGE environment variable has precedence over `LC_MESSAGES` in selecting the language for message translation on most R platforms." [4]
- Note:
  - My previous approach was to define a top level binding for the locale string and pass that reference to bind each of the following settings. That was fine until I wanted to be able to easily clear out all of the top-level bindngs to "reset" it with a `rm(ls())` kind of thing. For that reason, I just use the manifest strings here.

```
Sys.setenv(LANG = "en_US.UTF-8")
Sys.setlocale("LC_COLLATE", "en_US.UTF-8")
Sys.setlocale("LC_MESSAGES", "en_US.UTF-8")
```

Set the same random seed.

```
set.seed(970396220)
```

i. Packages

   `ID: org_gcr_2017-05-12_mara:E5E7A597-B53A-476B-83A4-0B4114125E0C`

[3] `http://cran.r-project.org/doc/manuals/r-patched/R-admin.html#Localization-of-messages`
[4] `http://stat.ethz.ch/R-manual/R-devel/library/base/html/locales.html`

<2014-11-14 Fri> Below is a try to auto-install packages if they are not available, and, it seems to have failed. Perhaps there is a better way, and I do need to find it. Until then I will install as-needed. As such, I suppose that I've found it.

A. assertthat

`ID: org_gcr_2017-05-12_mara:404A7BC2-F0E1-4E49-8CCA-D693D3430741`

Design-by-contract [5] is a great, great thing. Make it much easier with valuable functions and useful messages!

Cheatsheet:

- Always use `assert_that`
- Use
  - built ins to check tests
  - Additionally via `assertthat`:
    - `is.flag`
    - `is.string`
    - `has_name`
    - `has_attr`
    - `is.count`
    - `are_equal`
    - `not_empty`
    - `noNA`
    - `is.dir`
    - `is.writeable` and `is.readable`
    - `has_extension`

```
library(assertthat)
```

```
if(! require(assertthat)){
    install.packages("assertthat")
}
```

B. testthat

`ID: org_gcr_2017-05-12_mara:8B68967C-5F28-47AC-BD20-758661328199`

Design-by-contract and unit-tests [6] go hand-in-hand.

Expectations:

- `equals()` :: uses `all.equal()` to check for equality with numerical tolerance

[5] `http://cran.r-project.org/web/packages/assertthat/index.html`
[6] `http://cran.r-project.org/web/packages/testthat/index.html`

  - Shorthand: `expect_equal(x, y)`

`is_identical_to()` uses `identical()` to check for exact equality

  - Shorthand: `expect_identical(x, y)`

`is_equivalent_to()` is a more relaxed version of `equals()` that ignores attributes

  - Shorthand: `expect_equivalent(x, y)`

`is_a()` checks that an object `inherit()`'s from a specified class

  - Shorthand: `expect_is(x, y)`

`matches()` matches a character vector against a regular expression.

  - The optional all argument controls where all elements or just one element need to match.
  - Shorthand: `expect_matches(x, y)`

`prints_text()` matches the printed output from an expression against a regular expression

  - Shorthand: `expect_output(x, y)`

`shows_message()` checks that an expression shows a message

  - Shorthand: `expect_message(x, y)`

`gives_warning()` expects that you get a warning

  - Shorthand: `expect_warning(x, y)`

`throws_error()` verifies that the expression throws an error.

  - You can also supply a regular expression which is applied to the text of the error
  - Shorthand: `expect_error(x, y)`

`is_true()` is a useful catchall if none of the other expectations do what you want - it checks that an expression is true

  - `is_false()` is the complement of `is_true()`
  - Shorthand: `expect_true(x)`
  - Shorthand: `expect_false(x)`

- Notes
  - "Each test is run in its own environment so it is self-contained."
    - Plain old code so you can modify the global environment FYI

```
library(testthat)
```

```
if(! require(testthat)) {
    install.packages("testthat")
}
```

C. stringr

`ID: org_gcr_2017-05-12_mara:A25AE0EF-0169-4D2A-9907-D59FA7F0E9C4`

Make it really easy to work with strings [7]. That is indeed a good goal, and the reason that I installed this initially was because `testthat` mentions that it is used.

```
library(stringr)
```

```
if(! require(stringr)) {
    install.packages("stringr")
}
```

D. sqldf

`ID: org_gcr_2017-05-12_mara:D6EBC636-F22D-452F-B777-15F7C4C025A4`

How you extract data from a dataframe is flexible and everyone can and may do it differently. One option available is to use `SQL` [8], so make it available.

Comments taken from [9]

- "This [using SQL] is a skill that every analyst should possess"
- "Being able to write SQL will save you time and provide you with a way of getting repeatable results so that you don't have to focus on doing the calculations all the time and worrying about errors in Excel"
- "[instead] You can focus on the task of actually analyzing your data"

Notes from the user manual [10]

- Interesting package info
  - "Title Perform SQL Selects on R Data Frames"
  - "Author G. Grothendieck <ggrothendieck@gmail.com>"
  - "Description Description: Manipulate R data frames using SQL."
  - "Depends R (>= 2.14.0), gsubfn (>= 0.6), proto, RSQLite (>= 0.8-0),RSQLite.extfuns"
- Google group mentioned [11], joined it
- Official site mentioned and it has good docs
- Seems to uses SQLLite

[7] `http://cran.r-project.org/web/packages/stringr/index.html`
[8] `http://cran.r-project.org/web/packages/sqldf/index.html`
[9] `http://randyzwitch.com/sqldf-package-r/`
[10] `http://cran.r-project.org/web/packages/sqldf/index.html`
[11] `https://groups.google.com/forum/#!forum/sqldf`

- `read.csv.sql`
  - "Read a file into R filtering it with an sql statement. Only the filtered portion is processed by R so that files larger than R can otherwise handle can be accommodated."
  - Parms
    - Handles `http` and `ftp` `URLs`
    - `filter`
      - "If specified, this should be a shell/batch command that the input file is piped through. For read.csv2.sql it is by default the following on non-Windows systems: tr , .. This translates all commas in the file to dots."
      - Why is that specific example mentioned?
    - `field.types`
      - State the SQLite types for the column names
      - Rarely needed
    - `dbname`
      - "As in `sqldf` except that the default is `tempfile()`. Specifying `NULL` will put the database in memory which may improve speed but will limit the size of the database by the available memory."
  - Details
    - "Reads the indicated file into an sql database creating the database if it does not already exist. Then it applies the sql statement returning the result as a data frame. If the database did not exist prior to this statement it is removed."
    - "Note that it uses facilities of SQLite to read the file which are intended for speed and therefore not as flexible as in R. For example, it does not recognize quoted fields as special but will regard the quotes as part of the field. See the sqldf help for more information."
    - "`read.csv2.sql` is like `read.csv.sql` except the default sep is ";" and the default filter translates all commas in the file to decimal points (i.e. to dots)."
  - Value
    - "If the sql statement is a select statement then a data frame is returned."
- `sqldf`

  Description SQL select on data frames

Arguments

- =stringsAsFactors does what you think
- `row.names` could be useful
- `envir` could make it safer
- `method` determines how to type the data from the database into a dataframe
  - Looks like a **powerhouse** feature
  - Could greatly simplify data brokering
- `file.format`
  - `eol` handling mentioned across platforms
  - Ran into this with the built-in reader
- `dbname`
  - SQLite creates an in-memory database!

Details

- The typical action of sqldf is to

create a database in memory

read in the data frames and files used in the select statement. This is done by scanning the select statement to see which words in the select statement are of class "data.frame" or "file" in the parent frame, or the specified environment if envir is used, and for each object found by reading it into the database if it is a data frame. Note that this heuristic usually reads in the wanted data frames and files but on occasion may harmlessly reads in extra ones too.

run the select statement getting the result as a data frame

assign the classes of the returned data frame's columns if method = "auto". This is done by checking all the column names in the read-in data frames and if any are the same as a column output from the data base then that column is coerced to the class of the column whose name matched. If the class of the column is "factor" or "ordered" or if the column is not matched then the column is returned as is. If method = "auto.factor" then processing is similar except that "factor" and "ordered" classes and their levels will be assigned as well. The "auto.factor" heuristic is less reliable than the "auto" heuristic. If method = "raw" then the classes are returned as is from the database.

cleanup If the database was created by sqldf then it is deleted; otherwise, all tables that were created are dropped in order to leave the database in the same state that it was before. The database connection is terminated.

Warning Although sqldf is usually used with on-the-fly databases which it automatically sets up and destroys if you wish to use it with existing databases be sure to back up your database prior to using it since incorrect operation could destroy the entire database.

Value

- The result of the specified select statement is output as a data frame.
- If a vector of sql statements is given as x then the result of the last one is returned.
- If the x and connection arguments are missing then it returns a new connection and also places this connection in the option sqldf.connection.
    - Great to know that the connection is cached!

Notes

- Big FYI: Commas in columns will be parsed as column separators!
    - Recommends using `read.table` if this matter

Examples

- They all demonstrate how to do it in R and then again with SQL
- Super helpful
- You seem to be able to do everything that you would expect possible

Thoughts

- Need to grok both R and SQL to use this safely
- Using temp tables is kind of huge
- Via [12]
    - Use `_` instead lf `.` in column names from a R call
        - Where is this in the documentation?

Notes from the official site [13]

- Opening
    - How it works

[12] `https://stackoverflow.com/questions/19019883/how-to-handle-column-names-not-supported-by-sqldf-in-r`

[13] `https://code.google.com/p/sqldf/`

        * The user simply specifies an SQL statement
        * in R using data frame names in place of table names
        * and a database with appropriate table layouts/schema is automatically created,
        * the data frames are automatically loaded into the database,
        * the specified SQL statement is performed,
        * the result is read back into R
        * and the database is deleted all automatically behind the scenes making the database's existence transparent to the user who only specifies the SQL statement.
    - Supports
        * SQLite
        * H2
        * PostgreSQL
        * MySQL
    - The FAQ mostly talks about SQLite
- Overview
    - with sqldf the user is freed from having to do the following, all of which are automatically done:
        * database setup
        * writing the create table statement which defines each table
        * importing and exporting to and from the database
        * coercing of the returned columns to the appropriate class in common cases
    - It an be used for
        * learning R if you know SQL
        * Doing it faster than R
        * Load portions of a really large file
- Troubleshooting
    - Set the driver expicitly
    - "error messages regarding a data frame that has a dot in its name. The dot is an SQL operator. Either quote the name appropriately or change the name of the data frame to one without a dot."
- FAQ

- Column class conversion touched upon
- Dots in names
    - Dots are SQL operators so can't use them
        - See `?SQL92Keywords`
    - For columns
        - Either use underscore
        - Or simply remove them
    - For tables
        - Double quote the name
- H2 supports date types, which seems quite helpful
- Name a column ending with two underscores and a type and the library will convert the type to R correctly
    - Mentioned in the docs
- SQL is case **insensitive**
    - Don't rely on casing to differentiate column names
- We may examine the in-memory database table structure
- Be quite careful about CSV data that contains commas again as this lib won't handle it
- Good examples of cleaning data gettig int into a R friendly format
- Be sure to specify numeric values as integers or doubles so you get expected results from division

• Examples
- Example 1. Ordering and Limiting
- Example 2. Averaging and Grouping
- Example 3. Nested Select
- Example 4. Join
- Example 5. Insert Variables
    - Hugely convenient
- Example 6. File Input
- Example 7. Nested Select
- Example 8. Specifying File Format
- Example 9. Working with Databases
- Example 10. Persistent Connections

- Example 11. Between and Alternatives
- Example 12. Combine two files in permanent database
- Example 13. read.csv.sql and read.csv2.sql
  * Uses SQLite's import facility to create an in-memory database
  * Then it reads the results of the query into R
  * The import does not involve R so it can handle larger files than R can assuming that the query results in a size that does fit
- Example 14. Use of spatialite library functions
- Example 15. Use of RSQLite.extfuns library functions
- Example 16. Moving Average

SQLite, SQL As Understood By SQLite:

- Core Functions
- Aggregate Functions
- Date And Time Functions
- These previous are all provided by RSQLite.extfuns

`proto` wouldn't load, so first configure `sqldf` via this solution.

```
options(gsubfn.engine="R")
```

```
library(sqldf)
```

```
if(! require(sqldf)) {
    install.packages("sqldf")
}
```

E. MASS

`ID: org_gcr_2017-05-12_mara:A577D755-41C6-4E7E-9637-6118FCD5944B`

"Functions and datasets to support Venables and Ripley, 'Modern Applied Statistics with S' (4th edition, 2002)." Also, `sqldf` recommended it be installed, so it is the right time. [14]

```
library(MASS)
```

```
if(! require(MASS)) {
    install.packages("MASS")
}
```

F. jsonlite

`ID: org_gcr_2017-05-12_mara:B2715DBA-4A80-4DB8-B3BD-C660302D3FB9`

[14] `http://cran.r-project.org/web/packages/MASS/index.html`

Make it easy to work with JSON [37138455: http://cran.r-project.org/web/packages/jsonlite/index.html]. Reading the vignette's, it does a lot more, for example `rbind.pages`.

```
library(jsonlite)
```

```
if(! require(jsonlite)) {
    install.packages("jsonlite")
}
```

G. dplyr

ID: org_gcr_2017-05-12_mara:F2C4419D-59D8-423F-97A8-2C4ED2246186

dplyr: a grammar of data manipulation in R

The fact that I am loading both `plyr` and `dplyr` is something that I am questioning. I do so because I learned them in that order, so left it that way. However, this just results in **more** binding shadowing, and I am not sure of the implications, and they are usually never good.

> A fast, consistent tool for working with data frame like objects, both in memory and out of memory.

Readme. Manual. Introduction to dplyr.

```
library(dplyr)
```

H. data.table

ID: org_gcr_2017-05-12_mara:1604B2A7-7110-45B5-BB29-C27BDE9F37C4

`data.table` [15] is quite nice.

```
library(data.table)
```

```
if(! require(data.table)) {
    install.packages("data.table")
}
```

I. dtplyr

ID: org_gcr_2017-05-12_mara:169E4CE8-D5BE-48F7-A43F-E89CEE4C6211

```
library(dtplyr)
```

```
if(! require(dplyr)) {
    install.packages("dplyr")
}
```

```
if(! require(dtplyr)) {
    install.packages("dtplyr")
}
```

J. XML

ID: org_gcr_2017-05-12_mara:4F801FC6-2D93-495E-9B7A-5FE6C5A1003A

[15] http://cran.r-project.org/web/packages/data.table/index.html

Make truly enterprise [16].

```
library(XML)
```

```
if(! require(XML)) {
    install.packages("XML")
}
```

K. devtools

`ID: org_gcr_2017-05-12_mara:6BC90505-4FF2-4DB0-8020-F5F8057BA984`

`devtools`: Tools to make developing code easier

> Collection of package development tools

That is a bit too terse. Intro to the README follows

> The aim of devtools is to make your life as a package developer easier by providing R functions that simplify many common tasks. R packages are actually really simple, and with the right tools it should be easier to use the package structure than not. Package development in R can feel intimidating, but devtools does every thing it can to make it as welcoming as possible. devtools comes with a small guarantee: if because of a bug in devtools a member of R-core gets angry with you, I will send you a handwritten apology note. Just forward me the email and your address, and I'll get a card in the mail.

Excellent.

Readme. Manual. Github.

At the very least, just *know of* this package, as you will be installing it if you want to us `tidyr`.

```
library(devtools)
```

```
if(! require(devtools)) {
    install.packages("devtools")
    devtools::install_github("hadley/devtools")
}
```

L. magrittr

`ID: org_gcr_2017-05-12_mara:E4F80968-F698-4B63-B3F8-745625D74C39`

This is a add from the *most understated package definition* of the year department. `magrittr` [17] is, much like every Scheme library ever, deceptively simple in its power and ease of use that it provides.

```
library(magrittr)
```

[16] `http://cran.r-project.org/web/packages/XML/index.html`
[17] `http://cran.r-project.org/web/packages/magrittr/index.html`

```
if(! require(magrittr)) {
    devtools::install_github("smbache/magrittr")
}
```

M. reshape2

`ID: org_gcr_2017-05-12_mara:68274778-C58F-43D2-92D3-EBD90025A8D0`

`reshape2`: Flexibly reshape data: a reboot of the `reshape` package

> Reshape lets you flexibly restructure and aggregate data using just two functions: melt and cast.

CRAN. Manual. Github.

This seems to be a defacto standard.

```
library(reshape2)
```

```
if(! require(reshape2)) {
    install.packages("reshape2")
}
```

N. tidyr

`ID: org_gcr_2017-05-12_mara:5CE86E38-28A9-48F0-92F8-0874C7B89DEE`

`tidyr`: Easily tidy data with spread and gather functions for

> tidyr is an evolution of reshape2. It's design specifically for data tidying (not general reshaping or aggregating) and works well with dplyr data pipelines.

Readme. Manual. Github.

Not on CRAN yet so install via

```
library(tidyr)
```

```
if(! require(tidyr)) {
    devtools::install_github("hadley/tidyr")
}
```

O. lubridate

`ID: org_gcr_2017-05-12_mara:7F0E1F05-B547-4E6A-AFBF-E9E9D3FF1059`

lubridate: Make dealing with dates a little easier in

> Lubridate makes it easier to work with dates and times by providing functions to identify and parse date-time data, extract and modify components of a date-time (years, months, days, hours, minutes, and seconds), perform accurate math on date-times, handle time zones and Daylight Savings Time. Lubridate has a consistent, memorable syntax, that makes working with dates fun instead of frustrating.

Manual. Vignette.

```
library(lubridate)
```

```
if(! require(lubridate)) {
    install.packages("lubridate")
}
```

Perhaps in some *time* there will be a unified approach to time-management among all programming languages.

P. testit

`ID: org_gcr_2017-05-12_mara:4D15CAB4-E41F-4329-ABD3-6352BB53211C`

testit: A simple package for testing R packages

GitHub. CRAN. Manual.

Gives you `assert` and `test_pkg`. Save characters.

```
library(testit)
```

```
if(! require(testit)) {
    install.packages("testit")
}
```

Q. markdown

`ID: org_gcr_2017-05-12_mara:A060E8EA-7F3C-4A33-8001-EE936782065A`

- CRAN
  - reference
  - vignettes: markdown-examples
  - vignettes: markdown-output
- GitHub

```
library(markdown)
```

```
if(! require(markdown)) {
    install.packages("markdown")
}
```

> This package is referred to as R Markdown v1 when combined with knitr. The primary output format is HTML. Now we have introduced R Markdown v2, which is based on Pandoc and knitr, and supports much more types of output formats.

R. knitr

`ID: org_gcr_2017-05-12_mara:90B8880A-A32A-4761-A05A-164144AE93AF`

knitr: A general-purpose package for dynamic report generation in R

Read the home page. It has great resources.

Watched the video. Very nice to see; comfortable and familiar. Need to set up RStudio for it. Clearly a critical tool. Cites Knuth.

Features are amazingly understated. If you've worked with all of these tools, you will appreciate the importance of the author's effort!

`Objects`, `Options`, `Hooks`, and `Patterns` ... what is this, Emacs?

There are demo links. There is a project for examples. This showcase has links to websites, book reviews, solutions, R packages, courses, workshops and presentations, books, papers and reports, wrappers, and blog posts on `knitr`.

Here is the GitHub project. Read the motivations and see the hours and days and weeks that you have had spared! Uses `testit`, so read up on that and added it.

Read the Frequently Asked Questions. Joined the mailing list. `ess` supports it. Sure that I can configure the custom prompt. Great `README`.

CRAN as expected. Much better summary eg HTML, Makrdown, reStructuredText, and AsciiDoc are mentioned. Curious about the cacheing, and how I would do it in `org`. Custom code to run before and after a hunk are another thoughtful touch one would expect coming from `org`. Also support Python and shell. The $\LaTeX$ and LyX support is also pretty neat. Same README. Reference.

Somehow missed the reference card initially.

How to build package vignettes with knitr.

```
library(knitr)
```

```
if(! require(knitr)) {
    install.packages("knitr")
}
```

S. fortunes

`ID: org_gcr_2017-05-12_mara:420C813F-A6C0-41C9-8F4A-042DD8C4D059`

R Fortunes.

CRAN.

```
library(fortunes)
```

```
if(! require(fortunes)) {
    install.packages("fortunes")
}
```

T. ggplot2

`ID: org_gcr_2017-05-12_mara:2CB46933-90E6-4366-BA74-71D2B1A4DE8D`

- CRAN
  - reference

- Github
  - wiki
    - Lots of great resources
      - Whyu use it, how to support it, improvide i
      - Publications using it, around the web
      - FAQ, roadmap
    - Case studies
    - Tips and tricks
    - Enhancements
- Mail list
- Homepage
  - Documentation

```
library(ggplot2)
```

```
if(! require(ggplot2)) {
    install.packages("ggplot2")
}
```

U. tikzDevice

`ID: org_gcr_2017-05-12_mara:9CDF266B-9DE6-42A4-846E-212B11070D6F`

- CRAN
  - reference
  - vignettes: tikzDevice
- GitHub

```
library(tikzDevice)
```

```
if(! require(tikzDevice)) {
    install.packages("tikzDevice")
}
```

V. ascii

`ID: org_gcr_2017-05-12_mara:1645490F-7DEB-4227-8EAA-FBB1EFC329A0`

- CRAN
  - reference
- GitHub

```
library(ascii)
```

```
if(! require(ascii)) {
    install.packages("ascii")
}
```

Always display `org` representations; I'm assuming that it will be *the* dominant vehicle for analysis.

```
options(asciiType="org")
```

W. xtable

`ID: org_gcr_2017-05-12_mara:6438AA32-0335-4527-91BC-A90404AB36DD`

- CRAN
    - reference
    - vignettes: margintable
    - vignettes: xtableGallery
- R-Forge

```
library(xtable)
```

```
if(! require(xtable)) {
    install.packages("xtable")
}
```

X. Hmisc

`ID: org_gcr_2017-05-12_mara:F42F3746-7E10-478D-ACAE-125BE4A15D5C`

- CRAN
    - reference
- GitHub

```
library(Hmisc)
```

```
if(! require(Hmisc)) {
    install.packages("Hmisc")
}
```

Y. log4r

`ID: org_gcr_2017-05-12_mara:C6F05623-CF1C-46D0-A911-2ABB014267AF`

- CRAN
    - reference
- GitHub

```
library(log4r)
```

```
if(! require(log4r)) {
    install.packages("log4r")
}
```

Z. boot

`ID: org_gcr_2017-05-12_mara:1EC5636D-D340-4F00-BBEF-579000725366`

- CRAN
  - reference

```
library(boot)
```

```
if(! require(boot)) {
    install.packages("boot")
}
```

. kernlab

`ID: org_gcr_2017-05-12_mara:1860ABE6-06F1-449C-A5E4-EBCDEE4B8EB8`

- CRAN
  - reference
  - vignettes: kernlab
- GitHub

```
library(kernlab)
```

```
if(! require(kernlab)) {
    install.packages("kernlab")
}
```

. R Utils

`ID: org_gcr_2017-05-12_mara:5AAB5490-A3D4-42E1-9630-15E9CE8A6D63`

Programatically extract BZ2 files. Helpful for making decompression a separarate task from loading.

```
library(R.utils)
```

```
if(! require(R.utils)) {
    install.packages("R.utils")
}
```

. Not explicitly loaded, but interesting packages

`ID: org_gcr_2017-05-12_mara:62914096-7336-4480-B8E4-950E2A553AA8`

- ProjectTemplate
- evaluate.utils
- yaml
- whisker
- formatR

- General caching [18]
- stringi
  - Via
  - Seems focused on Unicode details
  - Why this instead of `stringr`?
- futile.options: Futile options management
  - Referenced by the `settings` article
- settings: Software Option Settings Manager for R
  - Via
- r-travis
  - Via
- testCoverage
  - Via
- xkcd
  - Via
- circ.graph.R
  - Via
- pandasql
  - Via
- qdap: Bridging the Gap Between Qualitative Data and Quantitative Analysis
  - Glad to find this
  - Via
- Rsenal
  - Via
  - Wished I had this the ohther day
- pander
  - Via
- DeployR
  - Via
- openssl: Bindings to OpenSSL

[18] `https://stackoverflow.com/questions/7262485/options-for-caching-memoization-hashing-in-r`

  - Via
- simmer
  - Again I would have had to have written this myself
  - Via
- checkpoint: Install Packages from Snapshots on the Checkpoint Server for Reproducibility
  - Via
- miniCRAN: Tools to create an internally consistent, mini version of CRAN with selected packages only
  - Via
- roxyPackage
  - Via
- roxygen2: In-source documentation for R
  - `roxypackage` mentioend it
- archivist: Tools for storing, restoring and searching for R objects
  - Via
- pkgKitten
  - Via
- rCharts
  - Via
- broom: Convert statistical analysis objects from R into tidy format
  - Via
- igraph: Network analysis and visualization
  - Why did they link to MRAN?
  - Looks like a great tool for learning about graphcs
  - Via

(d) .First

```
header-args: :noweb-ref rfirst-defs
ID: org_gcr_2017-05-12_mara:DB8DE38B-83C2-4CF9-B428-4629E4E06547
```

Reading:

Startup mandatory reading, the definitive source

Kickstarting R I just like this tutorial `attach` is a powerfuly convenient function. Sure, it can make you and your program go bonkers, but you know, it is worth it for the convenience. Joking aside, it has its place, so it should not go away completely. However, it ought not be used much, and if you do need to use it, the it should be really really obvious.

- Eg: [19]

```
gcr$attach.unsafe <- attach
gcr$attach <- function(...) {
    warning("NEVER USE ATTACH! Use ‘unsafe.attach‘ if you must.")
    attach.unsafe(...)
}
```

- `library` reports issues immediately, and by design, `require` does not… remind the useR that they *may* want the former not the latter
  - Via [20]
  - Just like everything else here, this is a *personal preference* thing!

```
gcr$require <- function(...) {
    warning("Are you sure you wanted ‘require‘ instead of ‘library‘?")
    base::require(...)
}
```

Sometimes you only want to list everything **but** functions [21]:

```
gcr$lsnofun <- function(name = parent.frame()) {
    obj <- ls(name = name)
    obj[!sapply(obj, function(x) is.function(get(x)))]
}
```

Make it really simple to specify how to handle errors in a given session:

```
gcr$recoveronerror <- function() {
    options(error=recover)
}

gcr$recoveronerroroff <- function() {
    options(error=NULL)
}
```

Make it really simple to specify how to handle warnings in a given session:

```
gcr$erroronwarn <- function() {
    options(warn=2)
}

gcr$erroronwarnoff <- function() {
    options(warn=0)
}
```

[19] `http://www.r-bloggers.com/to-attach-or-not-attach-that-is-the-question/`
[20] `http://www.r-bloggers.com/library-vs-require-in-r/`
[21] `https://stackoverflow.com/questions/13094324/hiding-function-names-from-ls-results-to-find-a-variable-name-more-qu`

`sqldf` should always use `SQLite`.

```
options(sqldf.driver = "SQLite")
```

Save your fingers from having to type `head` the next `n` thousand times [22] because I can't. `ess-rdired` and friends use the dataframe print function, so I didn't make dataframes print using it.

```
gcr$printdf <- function(df) {
    if (nrow(df) > 10) {
        print(head(df, 5))
        cat("---\n")
        print(tail(df, 5))
    } else {
        print(df)
    }
}
```

Sometimes you want to see all of the data in a dataframe, and sometimes you don't. Make it really easy to change whenever you feel like it.

```
gcr$printlen <- function(len=500) {
    options("max.print" = len)
}
```

When you've got `n`-thousand rows of data, make it easier to get a sample from it, just make it specific and keep it simple.

```
gcr$hundred <- function(df, idx=0) {
    df[idx:(idx+100),]
}
```

(e) .Renviron

```
header-args: :noweb-ref renviron-def
ID: org_gcr_2017-05-12_mara:9C053DA5-163C-4C31-9590-15F1BC5EF2DD
```

Install all packages to my home directory [23]

- Call `.libPaths()` to verify
- The directory must exist otherwise will ignore it
  - Solution:
    - Manual for now
    - Unsure of best way to generalize it

```
R_LIBS=~/.Rpackages
```

For the time being, GUI work will only be performed on OSX so utilize OSX's renderer [91578029: `http://emacs.1067599.n5.nabble.com/unable-to-start-device-X11-td330804.html`].

[22] `https://stackoverflow.com/questions/13024167/how-to-make-head-be-applied-automatically-to-output`
[23] `http://www.r-bloggers.com/installing-r-packages/`

That worked fine until I actually starting using that graphics device!

Then even though I was on OSX I **had** to switch to `X11`.

<2014-11-05 Wed> That was probably a mistake. The folks on-list said that `quartz` should be super. Perhaps the error was between the keyboard and the chair, so I am switching back.

<2014-11-08 Sat> When I us `ggplot` and quarts on this system, it blows up.

<2014-11-25 Tue> Switched to the official CRAN R build, which works fine on OSX.

```
R_INTERACTIVE_DEVICE=quartz
```

Explicitly state the timezone. This could be done either here or in the `.profile`. I'm not totally sure where to put it. Because I am trying to do **everything** with, I will put it here. Perhaps this should get set via `Sys.setenv` instead? I'll leave it for now and fix it later if necessary. I did test this out with a call to `Sys.time()` and it worked correctly.

```
TZ=America/Chicago
```