Introduction to package development

Hadley Wickham

Assistant Professor / Dobelman Family Junior Chair

Department of Statistics / Rice University

June 2012

1. What is a package?
2. Where do packages live?
3. Development cycle

This will be frustrating!

What is a package?

1. A name (`stringr`)

Recommendations

- All lowercase
- Be memorable. Be googleable!
- Ideas: remove vowels, add r (`stringr`), find related word and modify (`plyr`, `lubridate`), abbreviations (`rhipe`)
- Change package name if you make large API breaking changes

Your turn

Brainstorm a better name than rv!

2. A root directory
   `(stringr/)`

3. A directory of R code
   `(stringr/R/)`

```
├── R
|   ├── file1.r
|   ├── file2.r
```

Your turn

Break up the function `rv/one-file` into files and put in an `R` directory.

4. Add a description file

```
├── DESCRIPTION
├── R
|    ├── file1.r
|    ├── file2.r
```

```
Package: stringr
Type: Package
Title: Make it easier to work with strings.
Version: 0.5
Author: Hadley Wickham <h.wickham@gmail.com>
Maintainer: Hadley Wickham <h.wickham@gmail.com>
Description: stringr is a set of simple wrappers that make R's string
    functions more consistent, simpler and easier to use.  It does this
    by ensuring that: function and argument names (and positions) are
    consistent, all functions deal with NA's and zero length character
    appropriately, and the output data structures from each function
    matches the input data structures of other functions.
Imports: plyr
Depends: R (>= 2.11.0)
Suggests: testthat (>= 0.3)
License: GPL-2
```

https://github.com/hadley/devtools/wiki/Package-basics

```
Package: stringr
Title: Make it easier to work with strings.
Version: 0.5
Author: Hadley Wickham <h.wickham@gmail.com>
Maintainer: Hadley Wickham <h.wickham@gmail.com>
Description: stringr is a set of simple wrappers that make R's string
    functions more consistent, simpler and easier to use.  It does this
    by ensuring that: function and argument names (and positions) are
    consistent, all functions deal with NA's and zero length character
    appropriately, and the output data structures from each function
    matches the input data structures of other functions.
License: GPL-2
```

https://github.com/hadley/devtools/wiki/Package-basics

Your turn

Following that template, add a `DESCRIPTION` file to your rv package.

5. Documentation `(stringr/man)`

(Best if automatically generated from code comments)

```
├── DESCRIPTION
├── man
├── R
```

```
#' The length of a string (in characters).
#'
#' @param string input character vector
#' @return numeric vector giving number of characters in
#'   each element of the character vector.  Missing strings have
#'   missing length.
#' @keywords character
#' @seealso \code{\link{nchar}} which this function wraps
#' @export
#' @examples
#' str_length(letters)
#' str_length(c("i", "like", "programming", NA))
str_length <- function(string) {
  string <- check_string(string)

  nc <- nchar(string, allowNA = TRUE)
  is.na(nc) <- is.na(string)
  nc
}
```

https://github.com/hadley/devtools/wiki/docs-function

Your turn

Inspect the package sources for `stringr`, `lubridate` and `coin`. What files and directories didn’t I mention?

Where do packages live?

Libraries

A library is a collection of installed packages. You can have multiple libraries on your computer.

`.libPaths()` lists currently available libraries. Packages are installed into the first library.

Usually have at least two libraries: base packages and packages that you installed. Default is R-version specific: set `R_LIBS` to preserve packages across upgrades.

Your turn

Find your library directories. How are they structured? What files are in each package directory?

Download a binary package for your platform and unzip it - how does it compare to the packages in your library?

Types of package

- Development: all files on disk
- Source: only recognised directories, `build()`
- Binary: R code cached, C code compiled, `build(binary = T)`
- Bundled: `tar.gz` (source), `.zip` (windows binary), `.tgz` (mac binary)

Setting R_LIBS

- Mac/Linux: Create file `.Renviron` in your home directory and add `R_LIBS=~/R`
- Windows: see next page
- `?Startup` has all the gory details – there are many many options.
- After upgrading R, run
  `update.packages(checkBuilt = T, ask = F)`

Windows

1. Right-click on “My Computer”
2. Click “Properties”
3. Select “Advanced” tab
4. Click “Environment Variables”
5. Under “System Variables” scroll click add
6. Use `R_LIBS` and `c:/R`

Your turn

What libraries are you currently using? Why? Set up `R_LIBS` as described previously if you'd like to keep your packages when you upgrade R.

Make sure to create the directory too.

Dev mode

When simultaneously developing and using your own packages, it makes sense to have an extra library for development versions

Separates your buggy/experimental package code from your stable/production code.

```r
# Switch to alternative library for in-development
# packages - makes it easier to keep your existing
# code working

dev_mode()

# Switch back to normal
dev_mode()
```

```
# How to get R code installed into your libraries

# Download and install released version from CRAN:
install.packages()

# Download and install dev version from github:
install_github()

# Install personal version from local directory
install()
```

```r
# Gets the latest released version
install.packages("roxygen2")

# Gets the latest development version
install_github("roxygen", "klutometis")

# Installs my local personal development version
install("roxygen/")
```

```r
# How to load code into R:

# Uses currently installed package
library("ggplot2")

# Uses current source code
load_all("ggplot2")

# Installs package and then reloads
install("ggplot2", reload = T)

# Install package and restart
install("ggplot2")
# restart R
library("ggplot2")
```

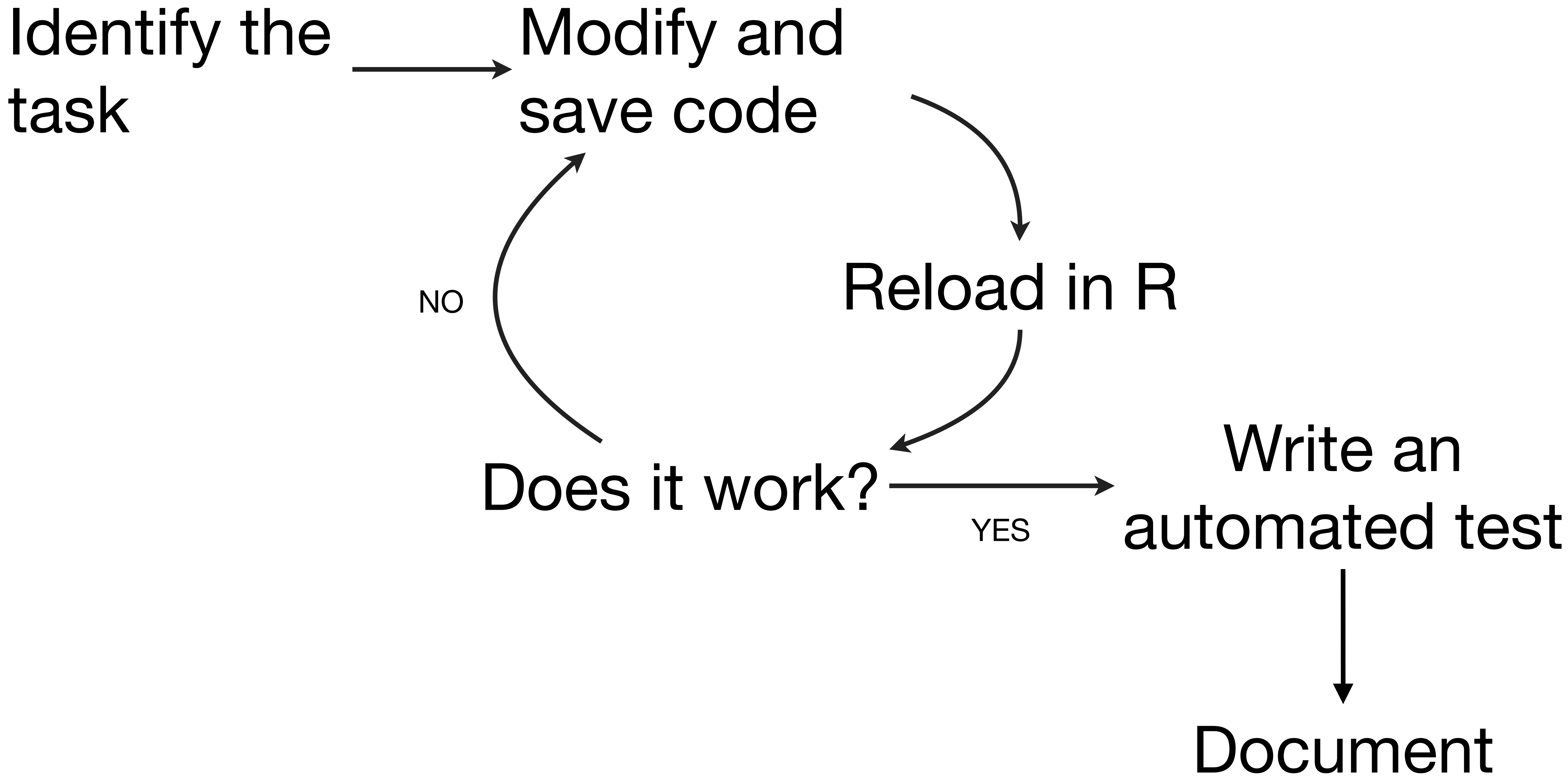
Your turn

Load the code in your `rv` package with `load_all("package-path")`. Modify a function and re-run `load_all()`.

Install the package and then load with `library(rv)`.

Development cycle

Exploratory programming



```r
library(devtools)

# * Reload code and data
load_all("rv")

# * Update documentation (roxygen2)
document("rv")

# * Run automated tests (testthat)
test("rv")

# All these functions take a path as their
# first argument
# If you've opened the project:
library(devtools)

# * Reload code and data
load_all()

# * Update documentation (roxygen2)
document()

# * Run automated tests (testthat)
test()
```