Package basics

Hadley Wickham

Assistant Professor / Dobelman Family Junior Chair

Department of Statistics / Rice University

October 2011

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

What is a package?

1. A name (`stringr`)
2. A root directory (`stringr/`)
3. A directory of R code (`stringr/R/`)

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

Recommendations

- All lowercase
- Be memorable
- Be googleable!
- Change package name if you make large API breaking changes

4. Add a description file

```
├── DESCRIPTION
├── 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

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

Download the source code for `stringr` and `plyr` from github, and `coin` from CRAN.

Unzip and explore. What files and directories didn't I mention?

Where do packages live?

Libraries

A library is a collection of 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.

Setting R_LIBS

- Windows: right-click shortcut, choose properties, and under path add `R_LIBS=c:/R/`
- Mac/Linux: Create file .Renviron in your home directory and add `R_LIBS=~/R`
- After upgrading R, run `update.packages(checkBuilt = T, ask = F)`

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.

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.

```
# 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 onto your computer:

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

# Download and install from github:
install_github()

# Install from local directory
install()
```

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

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

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

```
# 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)
```

Your first package

Your turn

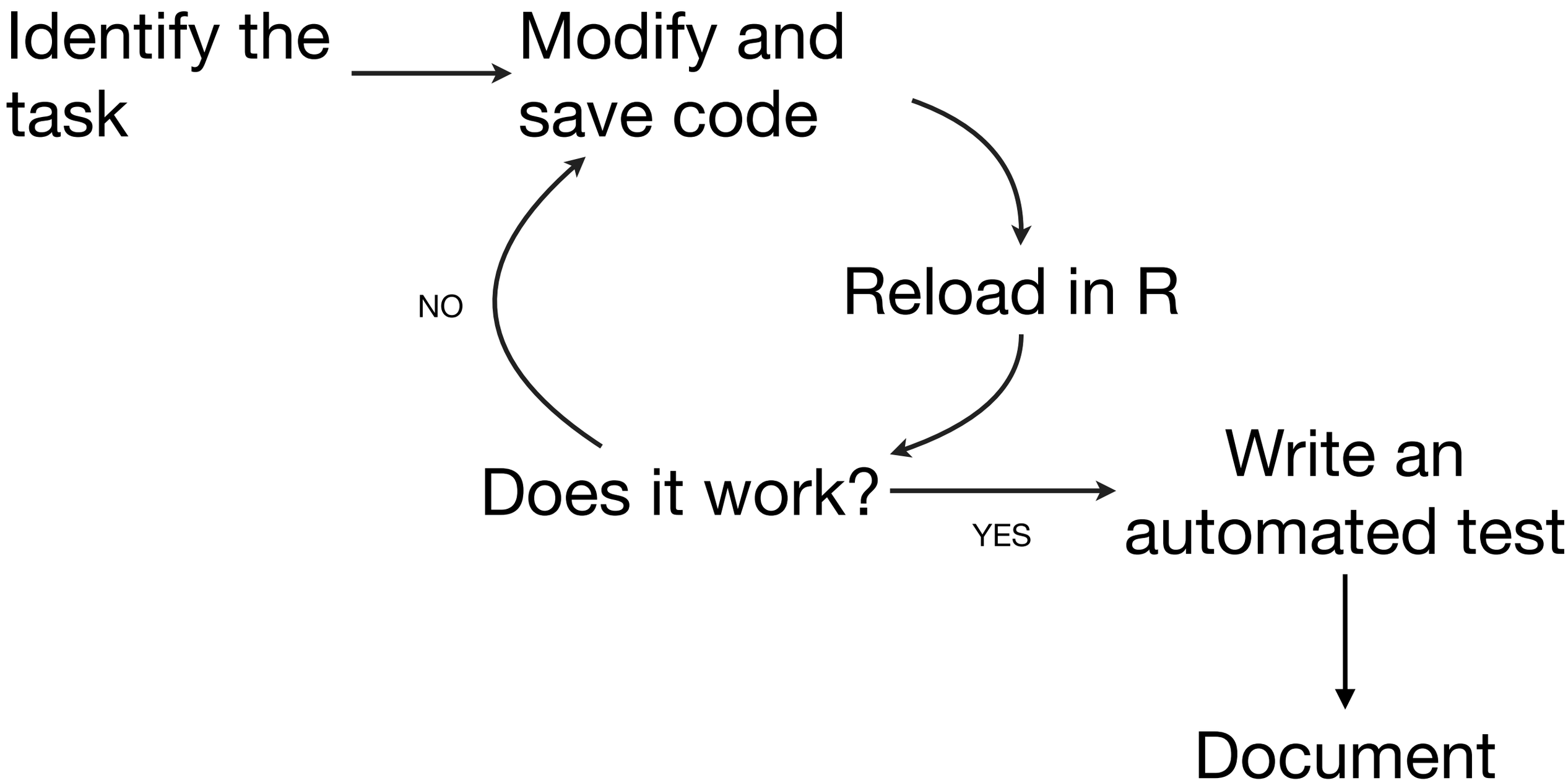
In the `hof-1` directory, you'll find a few functions that I'm considering turning into a package.

Start the process by putting them in the appropriate directory structure and creating a `DESCRIPTION` file.

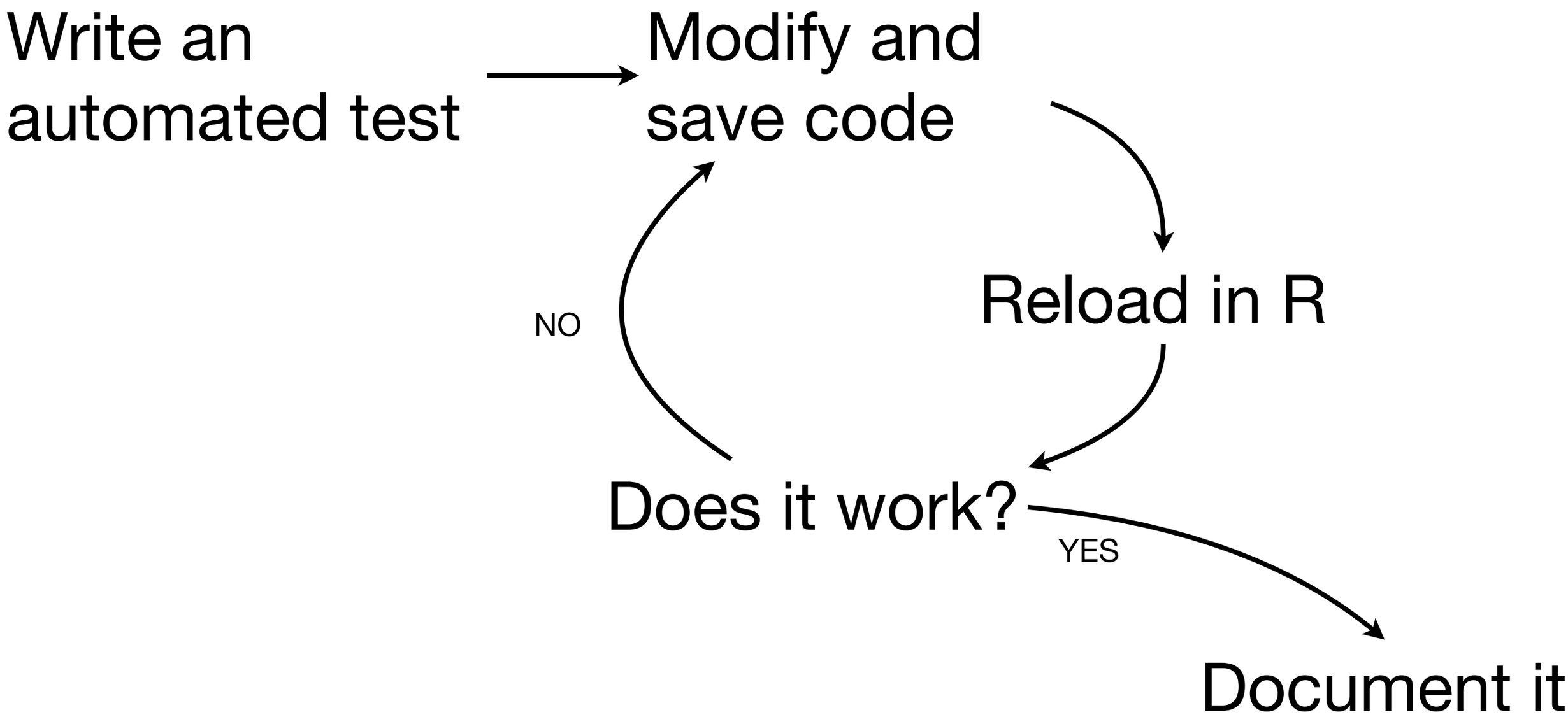
Load the code with `load_all("hof-1")`.

Development cycle

Exploratory programming



Confirmatory programming



aka test driven development (**TDD**)

```
library(devtools)

# * Reload code and data
load_all("hof-ok")

# * Run automated tests
test("hof-ok")

# * Update documentation
document("hof-ok")

# My text editor automatically saves all open
# files when I leave it, so I don't even need to
# explicitly save
```

Your turn

Load, test and document the `plyr` and `stringr` packages you just downloaded.

CRAN

At some point you will want to release your package to the public on CRAN.

To do so you also need to pass a stringent set of checks: `R CMD check`.

Devtools makes this a bit easier with the `check()`, `check_doc()`, `run_examples()`, and `release()` functions.

https://github.com/hadley/devtools/wiki/Release


This will be
frustrating!

Up next

- Package basics: `devtools`
- Documentation: `roxygen2`
- Testing: `testthat`
- Releasing your package: `devtools`