Release

Hadley Wickham

Assistant Professor / Dobelman Family Junior Chair

Department of Statistics / Rice University

June 2011

1. `R CMD check`
2. The release process
3. Post-release marketing
4. Source code control
5. Learning more

R CMD

check

1. R CMD build mypackage

2. R CMD check mypackage_0.1.tar.gz

3. Fix problems

4. Repeat ad nauseum

```
# Experimental!

check("stringr")
```

```
* using log directory ‘/Users/hadley/Documents/devtools/profr.Rcheck’
* using R version 2.13.0 (2011-04-13)
* using platform: x86_64-apple-darwin9.8.0 (64-bit)
* using session charset: UTF-8
* checking for file ‘profr/DESCRIPTION’ ... OK
* checking extension type ... Package
* this is package ‘profr’ version ‘0.3’
* checking package name space information ... OK
* checking package dependencies ... OK
* checking if this is a source package ... OK
* checking for executable files ... WARNING
Found the following executable file(s):
  .git/objects/00/d85478024279d5eb80be52b67c52263dd8b5e7
  .git/objects/01/779b25f0763f5cf41dab9ea18960a398bdc6b3
Source packages should not contain undeclared executable files.
See section 'Package structure' in manual 'Writing R Extensions'.
* checking whether package ‘profr’ can be installed ... OK
* checking installed package size ... OK
* checking package directory ... OK
* checking for portable file names ... OK
* checking for sufficient/correct file permissions ... OK
* checking DESCRIPTION meta-information ... OK
* checking top-level files ... OK
* checking index information ... OK
* checking package subdirectories ... OK
* checking R files for non-ASCII characters ... OK
* checking R files for syntax errors ... OK
* checking whether the package can be loaded ... OK
```

```
* checking whether the package can be loaded with stated dependencies ... OK
* checking whether the package can be unloaded cleanly ... OK
* checking whether the name space can be loaded with stated dependencies ... OK
* checking whether the name space can be unloaded cleanly ... OK
* checking for unstated dependencies in R code ... OK
* checking S3 generic/method consistency ... OK
* checking replacement functions ... OK
* checking foreign function calls ... OK
* checking R code for possible problems ... NOTE
.simplify: no visible binding for global variable ‘f’
.simplify: no visible binding for global variable ‘leaf’
ggplot.profr: no visible binding for global variable ‘level’
ggplot.profr: no visible binding for global variable ‘f’
* checking Rd files ... OK
* checking Rd metadata ... OK
* checking Rd cross-references ... OK
* checking for missing documentation entries ... OK
* checking for code/documentation mismatches ... OK
* checking Rd \usage sections ... WARNING
Undocumented arguments in documentation object 'explore'
  df

Functions with \usage entries need to have the appropriate \alias
entries, and all their arguments documented.
The \usage entries must correspond to syntactically valid R code.
See the chapter 'Writing R documentation files' in manual 'Writing R
Extensions'.
```

```
* checking Rd contents ... OK
* checking for unstated dependencies in examples ... OK
* checking contents of 'data' directory ... OK
* checking data for non-ASCII characters ... OK
* checking data for ASCII and uncompressed saves ... OK
* checking examples ... ERROR
Running examples in ‘profr-Ex.R’ failed
The error most likely occurred in:

> ### Name: parse_rprof
> ### Title: Parse Rprof output.
> ### Aliases: parse_rprof
> ### Keywords: debugging
>
> ### ** Examples
> nesting_ex <- system.file("samples", "nesting.rprof", package="profr")
> nesting <- parse_rprof(nesting_ex)
>
> reshape_ex <- system.file("samples", "reshape.rprof", package="profr")
> diamonds <- parse_rprof(reshape_ex)
> p <- profr(parse_rprof(reshape_ex))
```

Tips

- **ERROR**: must fix
- **WARNING**: probably should fix
- **NOTE**: might need to fix. Won't prevent CRAN submission
- Best practice is to build before checking, but it's faster to skip until the last round

Some tips

- Roxygen prevents many problems
- The more often you run it, the easier it gets.
- Use `devtools::check_rdoc()` and `run_examples()`

```
# R
document("stringr")
check_rdoc("stringr")
run_examples("stringr")
run_examples("stringr", "str_match.Rd")
check("stringr")

# command line
R CMD install ggplot && R -e "library(stringr);
devtools::run_examples('stringr')"
```

Your turn

Run `R CMD check` on `hof-problems` and fix the problems that you find.

Remember to re-run `document()` if you change the documentation.

There are sixteen problems for you to find and fix. Don't look at the answers until you've made an attempt at finding them all.

```
# ask.r
1. Need \dontrun around lapply(elements, log)
2. Missing param name for x

# funcall.r
3. Missing @export
4. Missing } in \code{f}
5. Trailing } in examples

# guard.r
6. No documentation for failwith
7. Missing \code{} around TRUE and FALSE
8. Missing @examples
```

pluck.r
9. Unnecessary @param
10. Missing ” in examples

tee.r
11. Missing @S3method for print.remember
12. Missing full stop in title.
13. stop()s in examples
14. @returns instead of @return
15. Missing link for \tee
16. Used # instead of #'

Release process

To CRAN

- update `NEWS`, checking that dates are correct. Use `devtools::show_news` to check that it's in the correct format
- `R CMD build package`
- Upload to cran
  `ftp -u ftp://cran.R-project.org/incoming/ package_name.tar.gz`
- Email cran@r-project.org

Next

- You'll get an email back from Kurt saying either that your package is now on CRAN, or there were problem you need to fix.
- A day or two later you'll get an email from the windows builder
- If all is well, you can now publicise your package

Post-release

Publicity

- No one will use your package if they don't know it exists.
- You can send a release announcement to r-packages@stat.math.ethz.ch. This should include your README and NEWS for this version
- Also announce on your blog or twitter, if appropriate.

Other resources

- **Website**: github pages a good place to start. Include links to papers, presentations etc *(I'm bad at this)*
- **Mailing list**: create a mailing list on google groups
- **Google alerts**: Set up a google alert so you can find out when people are using your package

Source code control

git + github

Makes collaboration much much easier.

Allow you to track changes, and rollback mistakes.

Steep learning curve, but worth it.

Resources

http://gitref.org/

http://www.syntevo.com/smartgit/index.html

http://help.github.com/

Learning more

The command line

Mastery of the command line requires some upfront investment but will save you a lot of time in the long run.

You'll notice I run R from the command line.

```
# .profile
alias R='R --no-save --no-restore-data --quiet'
export R_LIBS=~/R

# .Rprofile
.First <- function() {
  library(grDevices)
  options(
    repos = c(CRAN = "http://cran.r-project.org/"),
    device = "quartz")
}

# .inputrc
"\e[A": history-search-backward
"\e[B": history-search-forward
```

Where next

http://cran.r-project.org/doc/manuals/R-exts.html#Creating-R-packages

Subscribe to r-devel. Skim emails with interesting topics.

Download and inspect package source code. What makes packages easy to understand? What makes them hard to understand?