Introduction to R Package Development

Dirk Eddelbuettel
dirk.eddelbuettel@R-Project.org
edd@debian.org
@eddelbuettel

Big Data and Open Science with R
Warren Center for Network and Data Sciences
University of Pennsylvania, Philadelphia, PA
21 November 2014

Outline

A Good Forecast from About 10 Years Ago

```
> fortunes::fortune(92)

##
## If you don't go with R now, you will someday.
##    -- David Kane (on whether to use R or S-PLUS)
##       R-SIG-Finance (November 2004)
```

R: Very Briefly Summarized

- *A language and an environment* (cf R FAQ)
- *Has forever altered the way people analyze, visualize and manipulate data* (cf 1999 ACM citation)
- *A vibrant community and ecosystem*: CRAN + BioConductor provide > 6k packages that "just work"
- *The lingua franca of (applied) statistical research*
- Reliable cross-platform + cross-operating system
- Yet occassional challenges of getting R and code to collaborators, students, ...

Packages Rule: Part One

Key points from the previous slide:

- **community**: CRAN / packages part of R's success
- **cross-platform** / **cross-OS**: packages are portable
- **gettting R [...] code to colloborators**: distribution

Packages Rule: Part Two

More key points:

- **reproducibility**: aided greatly by identifiable package versions
- **version control**: learn about git (or svn)
- **quality control**: package creation / update is QA

Packages Rule: Part Three

Borrowing from Jeff Leek

Moreover:

- **have impact**: write software others use
- **software is the new publication**: name five recent papers, or name five recent packages...
- **save yourself time**: your own use of your own code is eased

Outline

The Key R Interface to Packages

`R CMD build someDirectory` to create a package

`R CMD check somePackage_1.2-3.tar.gz` to check a package

`R CMD INSTALL somePackage_1.2-3.tar.gz` to install a (source) package

Do It By Hand – Once

Create a directory and file `pkgA/DESCRIPTION`:

```
Package: pkgA
Type: Package
Title: A First Test
Version: 0.0.1
Date: 2014-11-15
Author: Dirk Eddelbuettel
Maintainer: Dirk Eddelbuettel <edd@debian.org>
Description: A minimal package
License: GPL (>= 2)
```

Run `R CMD build pkgA` and `R CMD check pkgA_0.0.1.tar.gz`.

Do It By Hand – Once

```
edd@max: ~/src/latex/conferences/penn-2014/code
edd@max:~/src/latex/conferences/penn-2014/code$ R CMD build pkgA && R CMD check
pkgA_0.0.1.tar.gz
* checking for file ‘pkgA/DESCRIPTION’ ... OK
* preparing ‘pkgA’:
* checking DESCRIPTION meta-information ... OK
* checking for LF line-endings in source and make files
* checking for empty or unneeded directories
* creating default NAMESPACE file
* building ‘pkgA_0.0.1.tar.gz’

* using log directory ‘/home/edd/src/latex/conferences/penn-2014/code/pkgA.Rchec
k’
* using R version 3.1.2 (2014-10-31)
* using platform: x86_64-pc-linux-gnu (64-bit)
* using session charset: UTF-8
* checking for file ‘pkgA/DESCRIPTION’ ... OK
* checking extension type ... Package
* this is package ‘pkgA’ version ‘0.0.1’
* checking package namespace information ... OK
* checking package dependencies ... OK
* checking if this is a source package ... OK
* checking if there is a namespace ... OK
* checking for executable files ... OK
* checking for hidden files and directories ... OK
* checking for portable file names ... OK
* checking for sufficient/correct file permissions ... OK
* checking whether package ‘pkgA’ can be installed ... OK
* checking installed package size ... OK
* checking package directory ... OK
* checking DESCRIPTION meta-information ... OK
* checking top-level files ... OK
* checking for left-over files ... OK
* checking index information ... OK
* checking package subdirectories ... 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 namespace can be loaded with stated dependencies ... OK
* checking whether the namespace can be unloaded cleanly ... OK
* checking loading without being on the library search path ... OK
* checking examples ... NONE
* checking PDF version of manual ... OK
* DONE

edd@max:~/src/latex/conferences/penn-2014/code$
```

Note the OKs – and absence of NOTE, WARNING or ERROR.

Do It By Hand – Once

The package build told us it added a file `NAMESPACE`:

```
# Default NAMESPACE created by R
# Remove the previous line if you edit this file

# Export all names
exportPattern(".")
```

These lines are now mandatory and control want you "export" and "import".

About DESCRIPTION

There is more than we have time to discuss now:

Author: Give credit where credit is due

Maintainer: Generally you, with a valid email address

Version: *Semantic Versioning* in the form a.b.c is popular, and sensible.

License: Matters, and worth giving it some thought.

Depends: Important when you have dependency

Imports: Dependency as `import()` or `importFrom()` – no time for this today

LinkingTo: No time to dive into this today

OS_type: Occassional restriction

SystemRequirements: For special needs

With R Code

So the initial attempt worked and created a valid – but useless – package.

Now add a directory `R/` and a file with a function or two.

Suggestion: Compute tail quantiles of a vector.

With R Code

Something like this in a file `R/myqs.R`:

```
## simple function to return quantiles of vector
myqs <- function(x,
                 at=c(0.01, 0.05, 0.10, 0.50,
                      0.90, 0.95, 0.99)) {

    ## should do some sanity checks on x here

    res <- quantile(x, probs=at)

}
```

Do It By Hand – Once

```
edd@max: ~/src/latex/conferences/penn-2014/code
edd@max:~/src/latex/conferences/penn-2014/code$ R CMD check pkgB_0.0.1.tar.gz
* using log directory ‘/home/edd/src/latex/conferences/penn-2014/code/pkgB.Rchec
k’
* using R version 3.1.2 (2014-10-31)
* using platform: x86_64-pc-linux-gnu (64-bit)
* using session charset: UTF-8
* checking for file ‘pkgB/DESCRIPTION’ ... OK
* checking extension type ... Package
* this is package ‘pkgB’ version ‘0.0.1’
* checking package namespace information ... OK
* checking package dependencies ... OK
* checking if this is a source package ... OK
* checking if there is a namespace ... OK
* checking for executable files ... OK
* checking for hidden files and directories ... OK
* checking for portable file names ... OK
* checking for sufficient/correct file permissions ... OK
* checking whether package ‘pkgB’ can be installed ... OK
* checking installed package size ... OK
* checking package directory ... OK
* checking DESCRIPTION meta-information ... OK
* checking top-level files ... OK
* checking for left-over 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 namespace can be loaded with stated dependencies ... OK
* checking whether the namespace can be unloaded cleanly ... OK
* checking loading without being on the library search path ... OK
* checking 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 ... OK
* checking for missing documentation entries ... WARNING
Undocumented code objects:
  ‘myqs’
All user-level objects in a package should have documentation entries.
See the chapter ‘Writing R documentation files’ in the ‘Writing R Extensions’ ma
nual.
* checking examples ... NONE
* checking PDF version of manual ... OK
* DONE

WARNING: There was 1 warning.
See
  ‘/home/edd/src/latex/conferences/penn-2014/code/pkgB.Rcheck/00check.log’
for details.

edd@max:~/src/latex/conferences/penn-2014/code$
```

Note the WARNING.

With Rd Documentation

Now we need to add a help page in Rd format.

By convention a file in `man` with the same name as the corresponding R code, ie `man/myqs.Rd`.

Do It By Hand – Once

edd@max: ~/src/latex/conferences/penn-2014/code

```
edd@max:~/src/latex/conferences/penn-2014/code$ cat pkgC/man/myqs.Rd
\name{myqs}
\alias{myqs}
\title{Simple quantile calculator}
\description{
  A simple demo function which returns quantiles
}
\usage{
  myqs(x, at=c(0.01, 0.05, 0.10, 0.50, 0.90, 0.95, 0.99))
}
\arguments{
  \item{x}{A vector for which quantiles are to be calculated}
  \item{at}{A vector with p-values for the desired quantiles}
}
\value{
  A named vector with the desired quantiles.
}
\details{
  This is just an examples.
}
\references{
  None.
}
\seealso{
  \link[stats:quantile]{quantile}
}
\examples{
  set.seed(123)     # be reproducible
  x <- rnorm(1000)
  myqs(x)
}
\author{Dirk Eddelbuettel}
\keyword{package}
edd@max:~/src/latex/conferences/penn-2014/code$
```

Not really worth typing by hand.

Better tools are available as eg `roxygen2` – covered later.

Do It By Hand – Once

```
edd@max: ~/src/latex/conferences/penn-2014/code
edd@max:~/src/latex/conferences/penn-2014/code$ R CMD check pkgC_0.0.1.tar.gz
* using log directory '/home/edd/src/latex/conferences/penn-2014/code/pkgC.Rchec
k'
* using R version 3.1.2 (2014-10-31)
* using platform: x86_64-pc-linux-gnu (64-bit)
* using session charset: UTF-8
* checking for file 'pkgC/DESCRIPTION' ... OK
* checking extension type ... Package
* this is package 'pkgC' version '0.0.1'
* checking package namespace information ... OK
* checking package dependencies ... OK
* checking if this is a source package ... OK
* checking if there is a namespace ... OK
* checking for executable files ... OK
* checking for hidden files and directories ... OK
* checking for portable file names ... OK
* checking for sufficient/correct file permissions ... OK
* checking whether package 'pkgC' can be installed ... OK
* checking installed package size ... OK
* checking package directory ... OK
* checking DESCRIPTION meta-information ... OK
* checking top-level files ... OK
* checking for left-over 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 namespace can be loaded with stated dependencies ... OK
* checking whether the namespace can be unloaded cleanly ... OK
* checking loading without being on the library search path ... OK
* checking 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 ... OK
* 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 ... OK
* checking Rd contents ... OK
* checking for unstated dependencies in examples ... OK
* checking examples ... OK
* checking PDF version of manual ... OK
* DONE

edd@max:~/src/latex/conferences/penn-2014/code$
```

Once again full of OKs.

Outline

Helpers For Creating a Package

`package.skeleton()` main worker, has warts; also called by RStudio

`kitten()` corrects issues with `package.skeleton()`

`create()` an alternative from `devtools` (which I do not use much)

Using package.skeleton()

```
> setwd("code")
> package.skeleton("pkgD")

## Creating directories ...
## Creating DESCRIPTION ...
## Creating NAMESPACE ...
## Creating Read-and-delete-me ...
## Saving functions and data ...
## Making help files ...
## Done.
## Further steps are described in './pkgD/Read-and-delete-me'.
```

Looks good, but `R CMD check ...` fails and dies.

Using kitten()

```
> setwd("code"); library(pkgKitten); kitten("pkgE")

## Creating directories ...
## Creating DESCRIPTION ...
## Creating NAMESPACE ...
## Creating Read-and-delete-me ...
## Saving functions and data ...
## Making help files ...
## Done.
## Further steps are described in './pkgE/Read-and-delete-me'.
##
## Adding pkgKitten overides.
## Deleted 'Read-and-delete-me'.
## Done.
##
## Consider reading the documentation for all the packaging
details.
## A good start is the 'Writing R Extensions' manual.
##
## And run 'R CMD check'.  Run it frequently.  And think of
those kittens.
```

Checking kitten()

edd@max: ~/src/latex/conferences/penn-2014/code

```
edd@max:~/src/latex/conferences/penn-2014/code$ R CMD check pkgE_1.0.tar.gz
* using log directory '/home/edd/src/latex/conferences/penn-2014/code/pkgE.Rchec
k'
* using R version 3.1.2 (2014-10-31)
* using platform: x86_64-pc-linux-gnu (64-bit)
* using session charset: UTF-8
* checking for file 'pkgE/DESCRIPTION' ... OK
* checking extension type ... Package
* this is package 'pkgE' version '1.0'
* checking package namespace information ... OK
* checking package dependencies ... OK
* checking if this is a source package ... OK
* checking if there is a namespace ... OK
* checking for executable files ... OK
* checking for hidden files and directories ... OK
* checking for portable file names ... OK
* checking for sufficient/correct file permissions ... OK
* checking whether package 'pkgE' can be installed ... OK
* checking installed package size ... OK
* checking package directory ... OK
* checking DESCRIPTION meta-information ... OK
* checking top-level files ... OK
* checking for left-over 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 namespace can be loaded with stated dependencies ... OK
* checking whether the namespace can be unloaded cleanly ... OK
* checking loading without being on the library search path ... OK
* checking 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 ... OK
* 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 ... OK
* checking Rd contents ... OK
* checking for unstated dependencies in examples ... OK
* checking examples ... OK
* checking PDF version of manual ... OK
* DONE

edd@max:~/src/latex/conferences/penn-2014/code$
```

Once again full of OKs.

Using roxygen2

The header of our `myqs()` function could look like this:

```
##' A simple demo function which returns quantiles
##'
##' This is just an examples.
##' @title Simple quantile calculator
##' @param x A vector for which quantiles are to be calculated
##' @param at A vector with p-values for the desired quantiles
##'
##' @return A named vector with the desired quantiles.
##'
##' @seealso \link[stats:quantile]{quantile}
##' @references None
##' @author Dirk Eddelbuettel
##' @examples
##' set.seed(123)      # be reproducible
##' x <- rnorm(1000)
##' myqs(x)
myqs <- function(x, at=c(0.01, 0.05, 0.1, 0.5, 0.9, 0.95, 0.99)){
    ## should do some sanity checks on x here
    res <- quantile(x, probs=at)
}
```

Using roxygen2

The usage is simple: call `roxygenize()`.

```
> setwd("code/pkgCroxy")
> library(roxygen2)
> roxygenize(".", roclets="rd")
```

which writes `myqs.Rd` for us.

Outline

Shipping Data

We can include data sets in packages.

Each data set should have a manual page as well, and there is `roxygen2` support.

A (very) recent example is the sp500 dataset in the `l1tf` package by Hadley.

Documentation is in Chapter 1.1.6 of Writing R Extensions.

Shipping Data: Example

See `pkgEdata` – a copy of `pkgE` with a `data/` directory.

It contains one data set, one helper function and documentation for the data set.

```
#' @name somedat
#' @title somedat - fake data set as an example
#' @description This data set contains a columns of data, a time-trend variable
#' foo and a noise variable bar
#' @docType data
#' @usage data(somedat)
#' @source Made-up by internal function \code{.dataCreation()}
#' @author Dirk Eddelbuettel
#' @keywords datasets
NULL

.dataCreation <- function() {
    # a boring fictious data.frame
    set.seed(124)
    N <- 100
    somedat <- data.frame(date=as.Date("2001-01-01")+0:(100-1),
                          foo=100 + seq(1,N)*0.25 + rnorm(N),
                          bar=runif(100)*0.5 + 50)
    save(somedat, file="../data/somedat.RData")
    invisible(NULL)
}
```

Shipping Data: Usage

```
> data(somedat, package="pkgEdata")
> head(somedat)

##         date       foo      bar
## 1 2001-01-01  98.86493 50.15971
## 2 2001-01-02 100.53832 50.43209
## 3 2001-01-03  99.98697 50.06274
## 4 2001-01-04 101.21231 50.46950
## 5 2001-01-05 102.67554 50.43260
## 6 2001-01-06 102.24448 50.15180
```

Shipping "Other Things"

We can include other files in packages as well.

By convention, each file or directory below `inst/` is shipped "as is". This allows us to access installed files via `system.file`.

A common special case is including header files in `inst/include`, example applications in `inst/examples` or unit tests in `inst/unitTest` or `inst/tests`.

Other useful cases are helper scripts in other languages (Perl, Python, ...).

Shipping "Other Things": Example

We include a shell script `inst/scripts/silly.sh`. It is not important what it does – but that we can call it.

```
#!/bin/bash

## this is just an example in which we simply output
## data to stdout -- which is 'fixed'.
##
## a real script would potentially do some work, maybe
## work with command-line arguments etc pp -- but our
## focus here is on the R side of things
##
## also worth reiterating that this could be a Perl,
## Python, Ruby, Node/JS, ... "whatever" script. The
## only thing that matter is that we should be able to
## invoke it on each platform, which may be easiest for
## shell.  So this is shell.

cat <<EOF
date,foo,bar
2001-01-01,10,12
2001-02-01,9,13
2001-03-01,11,12
2001-04-01,12,14
2001-05-01,13,15
2001-06-01,14,17
EOF
```

Shipping "Other Things": Example

```
##' Example of calling a script via system
##'
##' Uses \code{system.file()} to portably obtain the path
##' of a shell script and uses \code{system()} to execute.
##' @title Example of using system on package-supplied script
##' @return Several lines of text
##' @author Dirk Eddelbuettel
otherViaSystem <- function() {
    path <- system.file("scripts", "silly.sh",
                        package="pkgEother")
    cmd <- paste("sh", path)
    res <- system(cmd, intern=TRUE)
}
```

Shipping Data: Usage

We can run this:

```
> library(pkgEother)
> head(otherViaSystem())


## [1] "date,foo,bar"      "2001-01-01,10,12" "2001-02-01,9,13"
## [4] "2001-03-01,11,12" "2001-04-01,12,14" "2001-05-01,13,15"
```

Shipping "Other Things": Example

```
##' Example of calling a script via pip
##'
##' Uses \code{system.file()} to portably obtain the path
##' of a shell script and uses \code{pipe()} to execute it,
##' using the command output as input to read.
##' @title Example of using pipe on package-supplied script
##' @return A data.frame read from the output
##' @author Dirk Eddelbuettel
otherViaPipe <- function() {
    path <- system.file("scripts", "silly.sh",
                        package="pkgEother")
    cmd <- paste("sh", path)
    res <- read.csv(pipe(cmd))
}
```

Shipping Data: Usage

We can run this as well:

```
> library(pkgEother)
> head(otherViaPipe())

##         date foo bar
## 1 2001-01-01  10  12
## 2 2001-02-01   9  13
## 3 2001-03-01  11  12
## 4 2001-04-01  12  14
## 5 2001-05-01  13  15
## 6 2001-06-01  14  17
```

Shipping Tests

Adding unit tests may be one of the best way to ensure quality.

In a nutshell, it means adding short functions which test *invariants*.

Given input, and a function under consideration, compared the generated output to the expected value(s).

This is supported by several packages, notable `RUnit` and `testthat` but we probably do not have time to dive into this.

Shipping Tests: Simplest approach

Place a file `foo.R` in the directory `tests/`, run `R CMD check` on the package and *copy the resulting* `foo.Rout` *as* `foo.Rout.save` *in* `tests`.

```
> library(pkgEsimpletests)
>
> set.seed(123)       # be reproducible
> x <- rnorm(1000)    # some data
> res <- myqs(x)      # run our function of interest
> print(res)          # print result
```

Shipping Tests: Simplest approach

Below is the corresponding `foo.Rout.save` – R will during check time compare its output to the freshly generated one.

```
R version 3.1.2 (2014-10-31) -- "Pumpkin Helmet"
Copyright (C) 2014 The R Foundation for Statistical Computing
Platform: i686-pc-linux-gnu (32-bit)
[...]

> library(pkgEsimpletests)
>
> set.seed(123)      # be reproducible
> x <- rnorm(1000)   # some data
> res <- myqs(x)     # run our function of interest
> print(res)         # print result
          1%           5%          10%          50%          90%          95%
-2.158176203 -1.622584310 -1.267328289  0.009209639  1.254751947  1.676133871
         99%
 2.397645689
>
> proc.time()
   user  system elapsed
  0.312   0.352   0.271
```

Shipping Tests: Using RUnit

`RUnit` is one of several packages supporting unit tests.
When used, we need to add `Suggests: RUnit` to the `NAMESPACE` file.
We place a (essentially fixed) script calling the `RUnit` testrunner engine in `tests/`

```
stopifnot(require(RUnit, quietly=TRUE))
stopifnot(require(pkgEunittests, quietly=TRUE))
set.seed(42)                            # Set a seed to make the test deterministic

## Define tests
testSuite <- defineTestSuite(name="pkgEunittests Unit Tests",
                             dirs=system.file("tests", package="pkgEunittests"),
                             testFuncRegexp = "^[Tt]est+")

tests <- runTestSuite(testSuite)        # Run tests
printTextProtocol(tests)                # Print results

                                        # Return success or failure to R CMD CHECK
if (getErrors(tests)$nFail > 0) stop("TEST FAILED!")
if (getErrors(tests)$nErr > 0) stop("TEST HAD ERRORS!")
if (getErrors(tests)$nTestFunc < 1) stop("NO TEST FUNCTIONS RUN!")
```

Shipping Tests: Using RUnit

We also place files matching `runit.*.R` in `inst/tests`.

```
#.setUup <- function() { }
# can run some code needed below here, eg a database connection,
#.tearDown <- function() { }
# similar function to clean up at end

test01leftTail <- function() {
    set.seed(123)       # be reproducible
    x <- rnorm(1000)    # some data
    res <- myqs(x)      # run our function of interest
    comp <- quantile(x, probs=c(0.01, 0.05, 0.10, 0.50, 0.90, 0.95, 0.99))

    checkEquals(res[1],   comp[1],   msg="checking 1%-tile")
    checkEquals(res[2:3], comp[2:3], msg="checking 5% and 10%-tile")
}

test02rightTail <- function() {
    set.seed(123)       # be reproducible
    x <- rnorm(1000)    # some data
    res <- myqs(x)      # run our function of interest
    comp <- quantile(x, probs=c(0.01, 0.05, 0.10, 0.50, 0.90, 0.95, 0.99))

    checkEquals(res[7],   comp[7],   msg="checking 99%-tile")
    checkEquals(res[5:6], comp[5:6], msg="checking 90% and 95%-tile")
}
```

Shipping Vignettes

Vignettes – documentation in pdf or html format – can be added the via the directory `vignettes/`.

They may also require an entry `VignetteBuilder:` in `DESCRIPTION`.

Formats are either generally markdown or latex – and either form can incorporate embedded R code (and even code in other languages).

Yihui will cover the markdown variant tomorrow morning.

Shipping Vignettes: Example

edd@don: ~/git/samplecode/penn-2014-11

```
edd@don:~/git/samplecode/penn-2014-11(master)$ cat pkgEvignette/vignettes/pkgEvi
gnette.Rnw
\documentclass[11pt]{article}
%\VignetteIndexEntry{pkgEvignette}
%\VignetteKeywords{package,example}
%\VignettePackage{pkgEvignette}

\usepackage[margin=1in,paper=letterpaper]{geometry}
\usepackage[T1]{fontenc}
\usepackage{pslatex}        % just like RJournal
\usepackage{palatino,mathpazo}

<<echo=FALSE,print=FALSE>>=
prettyVersion <- packageDescription("pkgEvignette")[["Version"]]
prettyDate <- format(Sys.Date(), "%B %e, %Y")
@

\author{Some Author}
\title{\pkg{pkgEvignette}: A small Example}
\date{\pkg{pkgEvignette} version \Sexpr{prettyVersion} as of \Sexpr{prettyDate}}

\begin{document}
\maketitle

\abstract{
  \noindent Not much to say at this point.
}

\section{Some Text}

Some code:

<<sample,fig=TRUE>>=
set.seed(123)
x <- cumsum(rnorm(100))
plot(x, type='l', main="A random walk")
@

Code in a line: \Sexpr{sqrt(3)}.

\end{document}
edd@don:~/git/samplecode/penn-2014-11(master)$
```

Cannot type set Sweave inside of Sweave :)

Using RcppArmadillo

The `RcppArmadillo` package (discussed more tomorrow) has a variant `RcppArmadillo.package.skeleton()`:

```
> setwd("code")
> library(RcppArmadillo)
> RcppArmadillo.package.skeleton("pkgF")
```

Using RcppArmadillo

```
edd@max: ~/src/latex/conferences/penn-2014/code

edd@max:~/src/latex/conferences/penn-2014/code$ R CMD check pkgF_1.0.tar.gz
* using log directory '/home/edd/src/latex/conferences/penn-2014/code/pkgF.Rchec
k'
* using R version 3.1.2 (2014-10-31)
* using platform: x86_64-pc-linux-gnu (64-bit)
* using session charset: UTF-8
* checking for file 'pkgF/DESCRIPTION' ... OK
* checking extension type ... Package
* this is package 'pkgF' version '1.0'
* checking package namespace information ... OK
* checking package dependencies ... OK
* checking if this is a source package ... OK
* checking if there is a namespace ... OK
* checking for executable files ... OK
* checking for hidden files and directories ... OK
* checking for portable file names ... OK
* checking for sufficient/correct file permissions ... OK
* checking whether package 'pkgF' can be installed ... [13s/15s] OK
* checking installed package size ... OK
* checking package directory ... OK
* checking DESCRIPTION meta-information ... OK
* checking top-level files ... OK
* checking for left-over 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 namespace can be loaded with stated dependencies ... OK
* checking whether the namespace can be unloaded cleanly ... OK
* checking loading without being on the library search path ... OK
* checking 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 ... OK
* 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 ... OK
* checking Rd contents ... OK
* checking for unstated dependencies in examples ... OK
* checking line endings in C/C++/Fortran sources/headers ... OK
* checking line endings in Makefiles ... OK
* checking compilation flags in Makevars ... OK
* checking for portable use of $(BLAS_LIBS) and $(LAPACK_LIBS) ... OK
* checking compiled code ... OK
* checking examples ... OK
* checking PDF version of manual ... OK
* DONE

edd@max:~/src/latex/conferences/penn-2014/code$
```

Also full of OKs.

More on this tomorrow.

Outline

Other resources on packaging

Writing R Extensions by the R Core team authorative but somewhat terse

R packages by H Wickham book in progress, thorough but too opinionated on `devtools` etc

Building R Packages by D Diez nice slide deck, though getting a little dated

Developing R packages by J Leek excellent, but somewhat BioConductor focussed

Creating R Packages by F Leisch classic, thorough, also covers OO in R, a little dated too

Write your own Package by Stat545/UBC very new, `devtools`-centric as well.

Other resources on packaging

And of course maybe *the* best resource:

CRAN with over 6000 packages

BioConductor is also a very good source, with a strong development culture and many fine tutorials

GitHub now contains mirrors of CRAN and more, and can be searched.

Topics not covered

- Version control systems [VCS] (eg `git`, `svn`, ...) – highly recommended
- Continued Intgegration [CI] (ie Travis, Jenkins, ...) working with VCS
- Reproducible research and attempts snapshot installations such as `packrat`
- Docker (and our Rocker project) containers for R deployment, testing, reproducibility, ...
- Plus much, much more.

So go on ...

While there is always more to learn, and more details to uncover – you should now have a basis to start from.

So package on!