

# Sharing the Love

## Building Your First Package

Chris Campbell



# Agenda

- What is a Package?
- How do I Build a Package?
- Unit Testing
- Sharing a Package

# What is a Package?

- R is like a giant combining robot

```
> search()  
[1] ".GlobalEnv" "package:stats"  
[3] "package:graphics" "package:grDevices"  
[5] "package:utils" "package:datasets"  
[7] "package:methods" "Autoloads"  
[9] "package:base"
```



# What is a Package?



- R is like a giant combining robot
- Add units of functionality

```
> install.packages("MSToolkit", lib = .libPaths()[1],  
+   repos = "http://cran.ma.imperial.ac.uk/")  
  
> require(MSToolkit)
```

# What is a Package?

- Add units of functionality
- Until ultimate functionality is achieved

```
> search()  
[1] ".GlobalEnv"           "package:MSToolkit"  
[3] "package:MASS"         "package:stats"  
[5] "package:graphics"    "package:grDevices"  
[7] "package:utils"       "package:datasets"  
[9] "package:methods"    "Autoloads"  
[11] "package:base"
```



# What is a Package? Convenience

- Multiple units of code can be accessed together
- Interdependent units of code can be shared together
- Interdependent units of code can be tested together

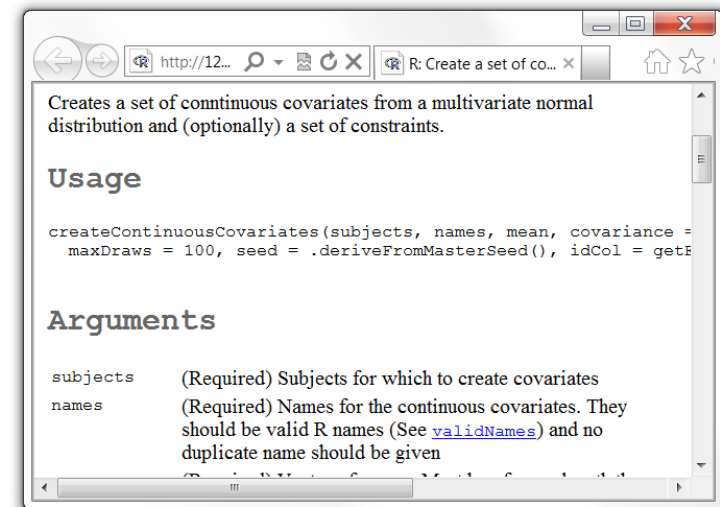




# What is a Package? Communication

- Scripts may not be clear to others
- We can define workflow or process with reusable content
- High level functions to combine analysis steps
- Help files
- Demonstration function

```
> help("createContinuousCovariates")
```



# What is a Package? Reproducible Results

- Scripts can be messy and hard to reuse
- Functions should use variables, rather than hard coded values
- Since fewer manual steps, the workflow is standardised





# What is a Package? Memory Efficient Functionality

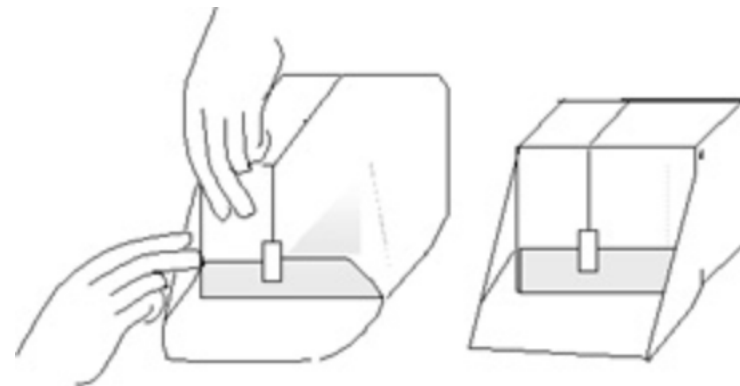
- Systems can become bloated with excess functionality
- Currently 4301 packages on CRAN
- Choose only essential tools
- Rest is not loaded



# How do I make a Package?



- Rtools (and R)
- Set PATH
- Create files with `package.skeleton`
- Compile package



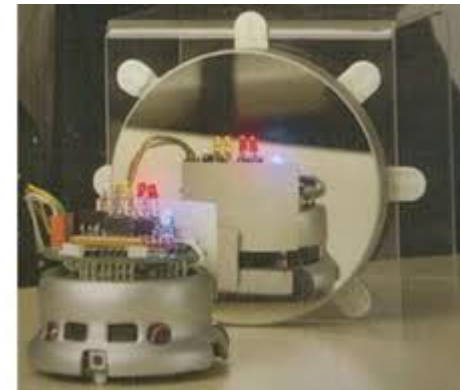
# How do I make a Package? Rtools



- Building R for Windows
- <http://cran.ma.imperial.ac.uk/bin/windows/Rtools/>
- Collated by Prof Brian Ripley
- Maintained by Duncan Murdoch

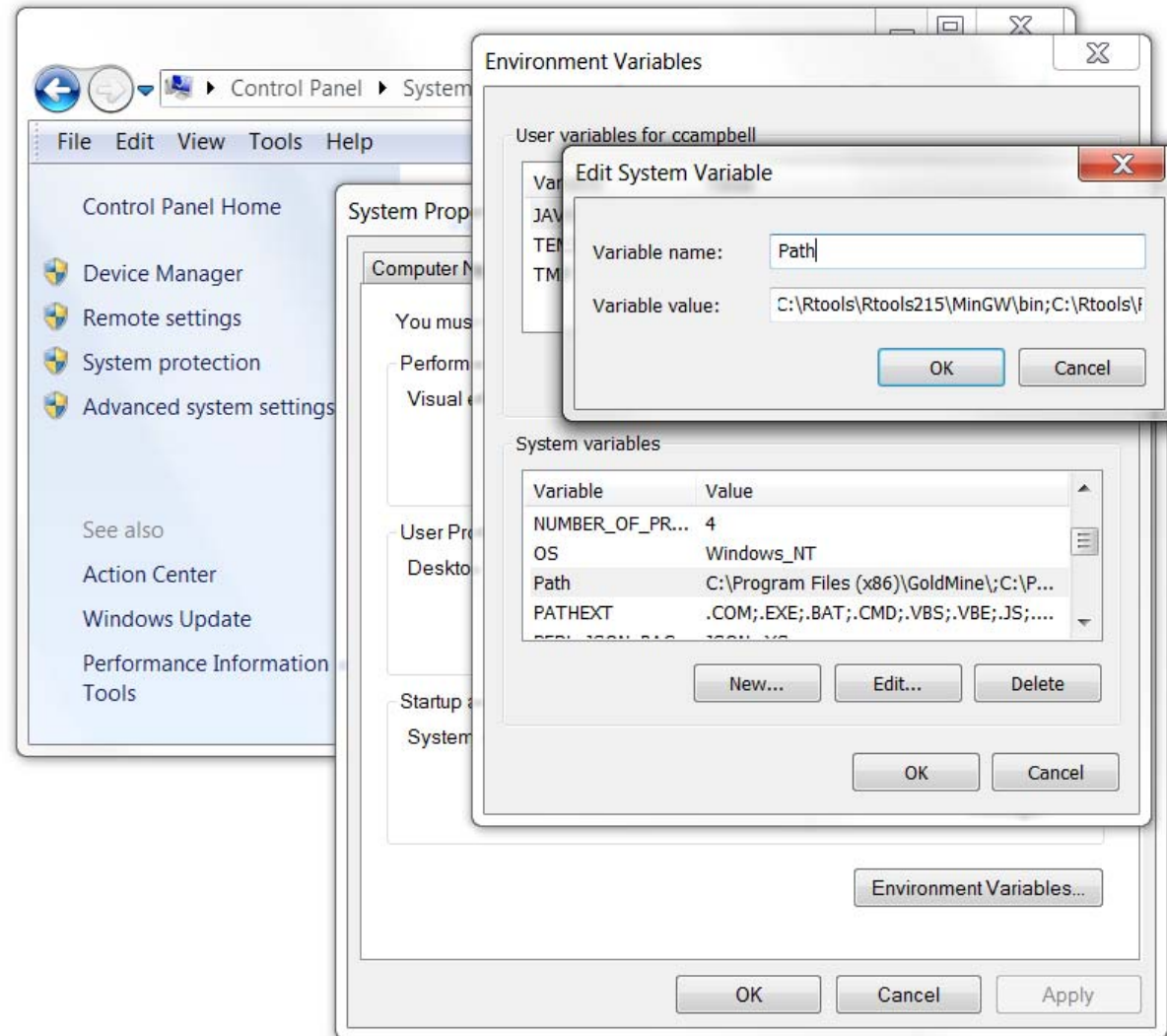
# How do I make a Package? Set PATH

- Computers are not self-aware (yet)
- They need to be told where to find programs
- The list of places looks is the PATH



# How do I make a Package? Set PATH

- Add bin folders of Rtools and R, separated by ; e.g.
- ;C:\Rtools\Rtools215\MinGW\bin;C:\Rtools\Rtools215\bin;C:\Program Files\R\R-2.15.1\bin\i386



# How do I make a Package? package.skeleton



- A package consists of files & folders following a strict structure
- DESCRIPTION describes the package!
- NAMESPACE defines visible functions in the package
- Rd files are help files in LaTeX markup; use tags with syntax `\tag{text}` to add text and formatting to help documents
- These files can be created for you by `package.skeleton`



# How do I make a Package? package.skeleton

```
> # a function to calculate elapsed time
> # the Format must be the same for Start and Time
> # see ?strptime for format codes

> getElapsedTime <- function(Start, Time,
+   Format = "%m/%d/%y %H:%M") {
+
+   startT <- as.POSIXct(Start, format = Format)
+   timeT <- as.POSIXct(Time, format = Format)
+   elapsedT <- timeT - startT
+
+   cat("Time difference in", attr(elapsedT, "units"), "\n")
+   return(c(elapsedT))
}

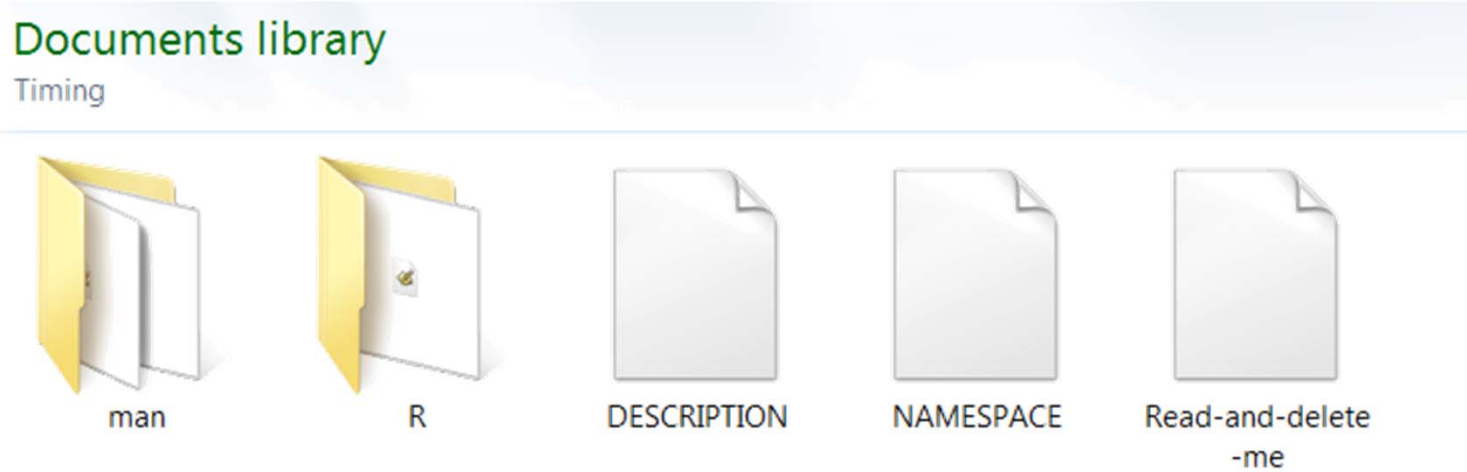
> ls(".GlobalEnv")
[1] "getElapsedTime"
```

# How do I make a Package? package.skeleton



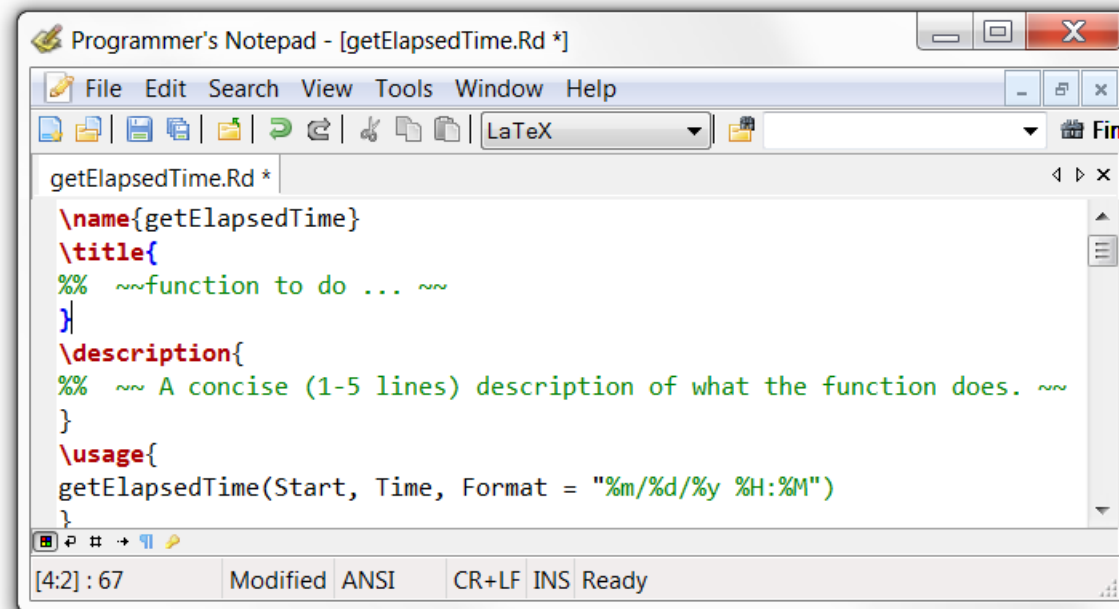
```
> package.skeleton("Timing")
Creating directories ...
Creating DESCRIPTION ...
Creating NAMESPACE ...
Creating Read-and-delete-me ...
Saving functions and data ...
Making help files ...
Done.
Further steps are described in './Timing/Read-and-delete-me'.
```

# How do I make a Package? package.skeleton



# How do I make a Package? package.skeleton

- Edit help files

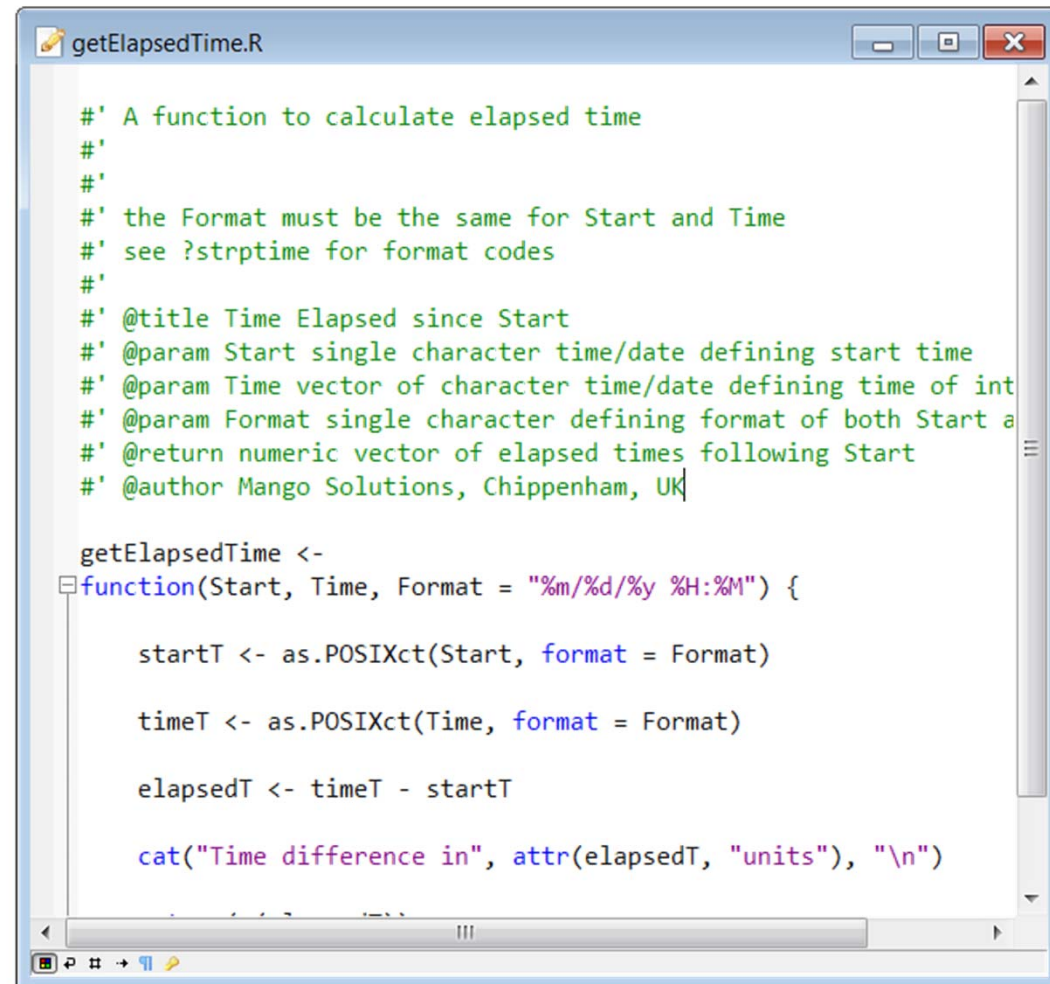


```
Programmer's Notepad - [getElapsedTime.Rd *]  
File Edit Search View Tools Window Help  
getElapsedTime.Rd *  
\name{getElapsedTime}  
\title{  
%% ~function to do ... ~  
}  
\description{  
%% ~ A concise (1-5 lines) description of what the function does. ~  
}  
\usage{  
getElapsedTime(Start, Time, Format = "%m/%d/%y %H:%M")  
}
```

[4:2] : 67 Modified ANSI CR+LF INS Ready

# How do I make a Package? package.skeleton

- Or use roxygen2 turn flags inline in code into Rd files



```
getElapsedTime.R

#' A function to calculate elapsed time
#'
#'
#' the Format must be the same for Start and Time
#' see ?strptime for format codes
#'
#' @title Time Elapsed since Start
#' @param Start single character time/date defining start time
#' @param Time vector of character time/date defining time of int
#' @param Format single character defining format of both Start a
#' @return numeric vector of elapsed times following Start
#' @author Mango Solutions, Chippenham, UK

getElapsedTime <-
function(Start, Time, Format = "%m/%d/%y %H:%M") {

  startT <- as.POSIXct(Start, format = Format)

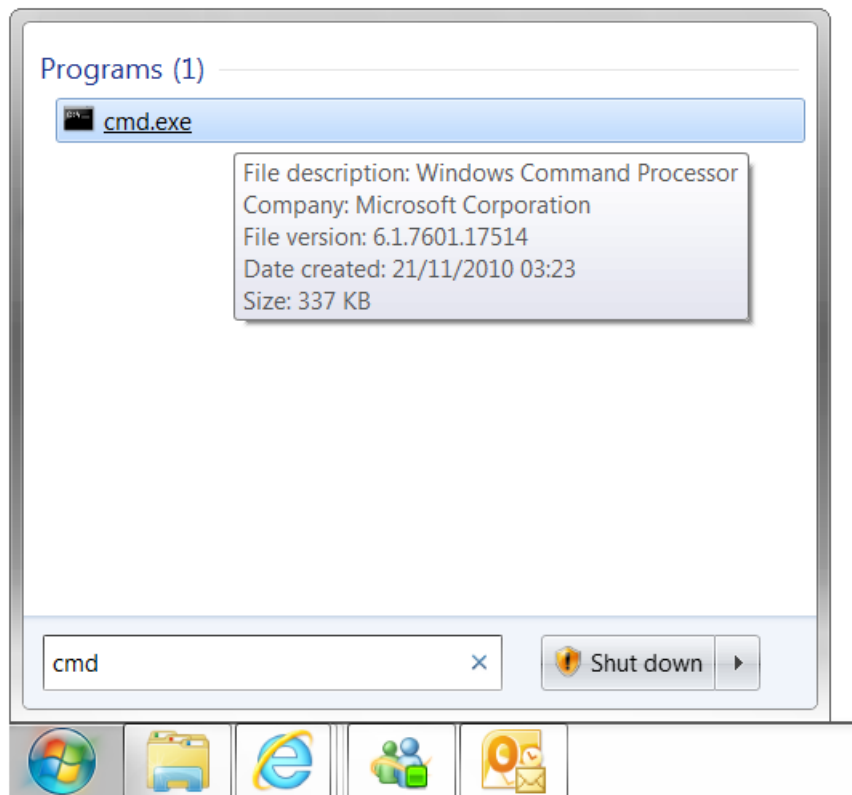
  timeT <- as.POSIXct(Time, format = Format)

  elapsedT <- timeT - startT

  cat("Time difference in", attr(elapsedT, "units"), "\n")
}
```

# How do I make a Package? Compile

- R on PATH is available from cmd





# How do I make a Package? Compile

- R on PATH is available from cmd



```
C:\Windows\system32\cmd.exe
C:\Users\ccampbell>R
R version 2.15.1 (2012-06-22) -- "Roasted Marshmallows"
Copyright (C) 2012 The R Foundation for Statistical Computing
ISBN 3-900051-07-0
Platform: i386-pc-mingw32/i386 (32-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

Natural language support but running in an English locale

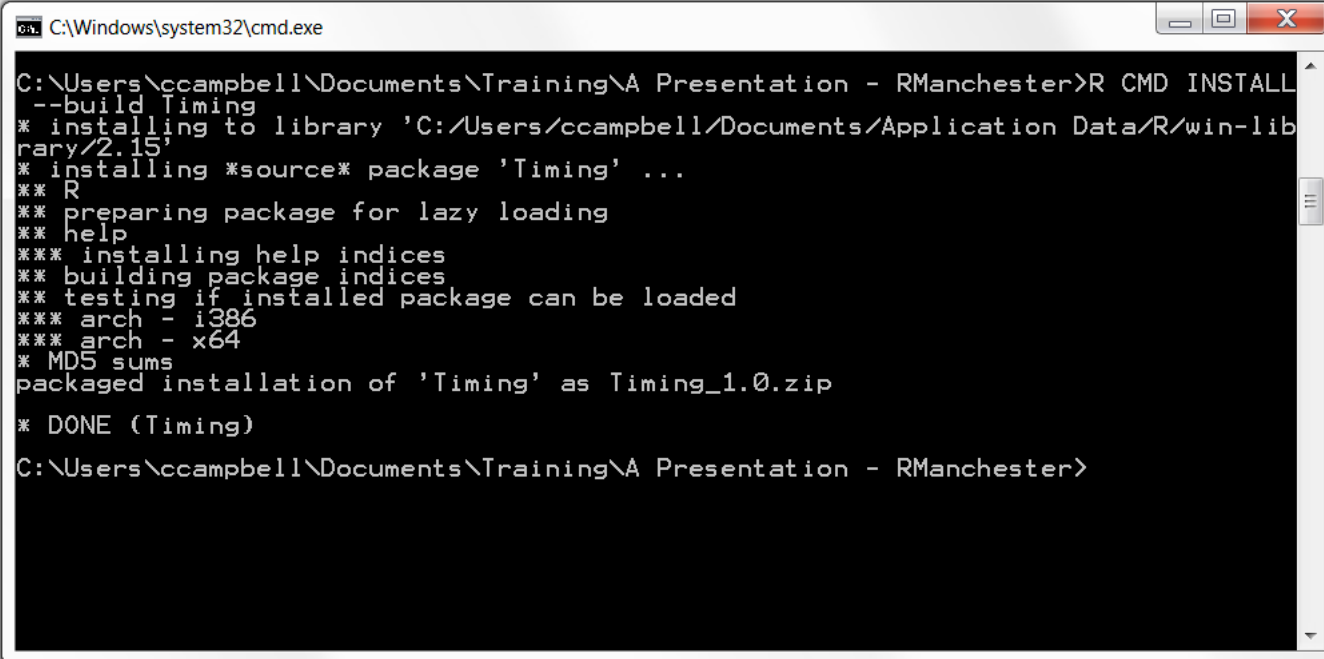
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> 1 + 1
[1] 2
> q()
Save workspace image? [y/n/c]: n
C:\Users\ccampbell>
```

# How do I make a Package? Compile

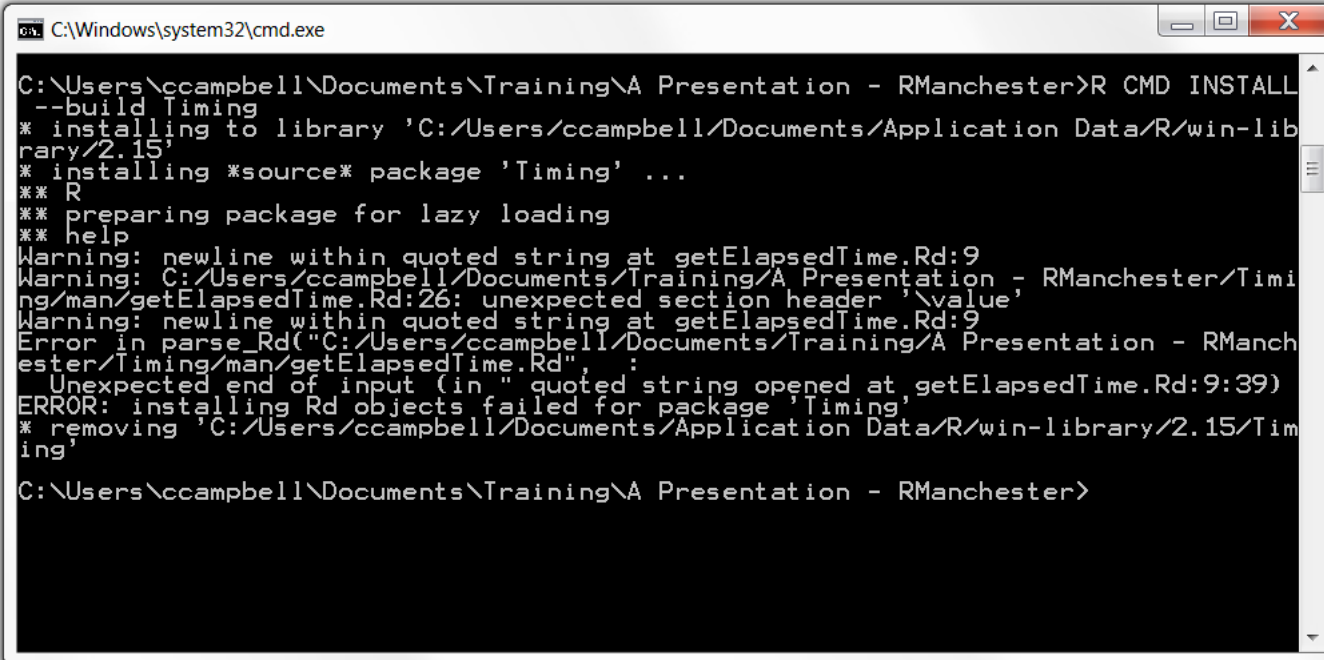
- R CMD INSTALL --build Timing



```
C:\Windows\system32\cmd.exe
C:\Users\ccampbell\Documents\Training\A Presentation - RManchester>R CMD INSTALL
--build Timing
* installing to library 'C:/Users/ccampbell/Documents/Application Data/R/win-lib
rary/2.15'
* installing *source* package 'Timing' ...
** R
** preparing package for lazy loading
** help
*** installing help indices
** building package indices
** testing if installed package can be loaded
*** arch - i386
*** arch - x64
* MD5 sums
packaged installation of 'Timing' as Timing_1.0.zip
* DONE (Timing)
C:\Users\ccampbell\Documents\Training\A Presentation - RManchester>
```

# How do I make a Package? Might take a few tries!

- Rd files can be tricky to get right manually!



```
C:\Windows\system32\cmd.exe
C:\Users\ccampbell\Documents\Training\A Presentation - RManchester>R CMD INSTALL
--build Timing
* installing to library 'C:/Users/ccampbell/Documents/Application Data/R/win-lib
rary/2.15'
* installing *source* package 'Timing' ...
** R
** preparing package for lazy loading
** help
Warning: newline within quoted string at getElapsedTime.Rd:9
Warning: C:/Users/ccampbell/Documents/Training/A Presentation - RManchester/Timi
ng/man/getElapsedTime.Rd:26: unexpected section header '\value'
Warning: newline within quoted string at getElapsedTime.Rd:9
Error in parse_Rd("C:/Users/ccampbell/Documents/Training/A Presentation - RManch
ester/Timing/man/getElapsedTime.Rd", :
Unexpected end of input (in " quoted string opened at getElapsedTime.Rd:9:39)
ERROR: installing Rd objects failed for package 'Timing'
* removing 'C:/Users/ccampbell/Documents/Application Data/R/win-library/2.15/Tim
ing'

C:\Users\ccampbell\Documents\Training\A Presentation - RManchester>
```

# How do I make a Package? Versions



- Our code is important; we must keep track of it!
- Manually - change filename with every save
- Automatically - use SVN, GIT

# Unit Testing

## Check it Works

- Unit Testing is an important part of coding
- Try packages testthat or RUnit
- Check code does what we expect

```
> getElapsedTime(Start = "05/21/12 15:20",  
+   Time = c("05/21/12 15:20", "05/21/12 15:21",  
+   "05/21/12 15:30", "05/21/12 16:20"))  
Time difference in secs  
[1]    0    60   600 3600
```

# Unit Testing Checks

- Test for each function
- Contains multiple checks

```
test.getElapsedTime <- function()  
{  
  test1 <- getElapsedTime(Start = "05/21/12 15:20",  
    Time = c("05/21/12 15:20", "05/21/12 15:21",  
      "05/21/12 15:30", "05/21/12 16:20"))  
  checkEquals(test1, c(0, 60, 600, 3600),  
    msg = " TEST 1: Default example")  
  ...  
}
```



# Unit Testing Automated Tests

- Run all tests at once

```
runTests <- function(testPath =  
  system.file(package = "Timing", "testing")  
{  
  testSuite <- defineTestSuite(name = "Unit Test Suite",  
    dirs = testPath,  
    testFileRegexp = "^runit.+\\. [rR]$",  
    testFuncRegexp = "^test.+" )  
}
```

# Unit Testing Robustness

- Unit Testing is an important part of coding
- Function may be passed data outside it's anticipated structure
- We don't want to create misleading results

```
> getElapsedTime(c("05/21/12 12:01", "05/21/12 18:03"),  
+ "05/21/12 15:20")  
Time difference in hours  
[1] 3.316667 -2.716667
```

# Unit Testing

## Errors are Good!

- We expect a function to behave in a certain way
- We can use `stop` and `warning` to flag problems

```
getElapsedTime <- function(Start, Time,  
  Format = "%m/%d/%y %H:%M") {  
  
  if (any(length(Start) != 1)) {  
    stop("Start should be length 1") }  
}
```

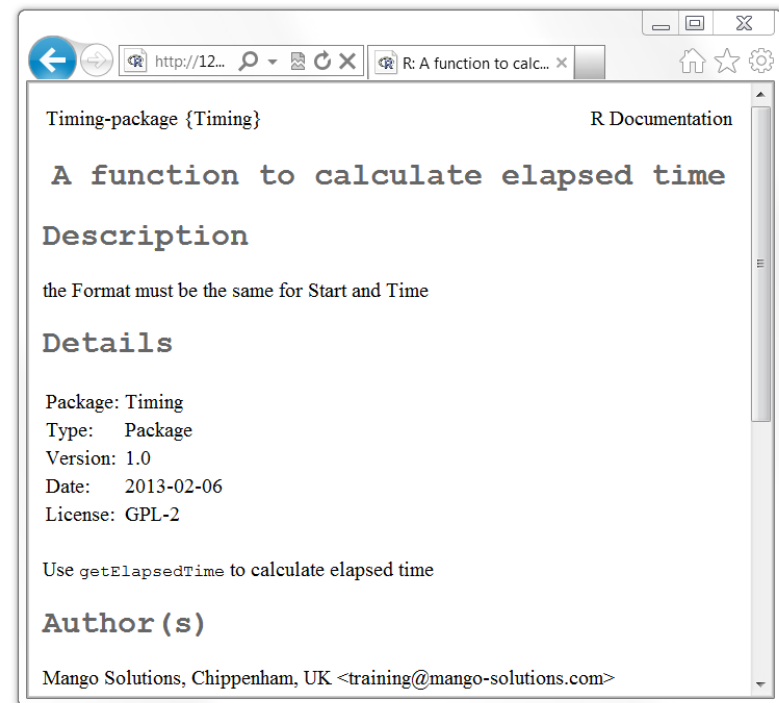
...

# Sharing a Package It Works!

- Now you can share your code

```
> require(Timing)
```

```
> help(package = "Timing")
```



# Sharing a Package Giving Back to the Community

- With colleagues & peers as a zip
- Globally on r-forge, CRAN



Timing\_1.0.zip

## Summary

- Packages allow us to collate code
- We are encouraged to make scripts work generically, so analysis more reproducible
- We can share ideas with others, allowing them to use our methods
- We are encouraged to split our scripts into units that can be efficiently tested