Advanced R Programming - Lecture 3

Leif Jonsson

Linköping University

leif.jonsson@ericsson.com
leif.r.jonsson@liu.se

August 29, 2016

Today

- Best practices for scientific computing
- R packages
- Git and GitHub
- Creating R packages
- Documentation with ROxygen
- Unit testing with testthat
- R-Studio debugger

Questions since last time?

Best practices for scientific computing

Based on the article referred to on course page...

1. Write code for people

1. Write code for people
   1.1 A program should not require its readers to hold more than a handful of facts in memory at once
   1.2 Make names consistent, distinctive, and meaningful
   1.3 Make code style and formatting consistent

Let the computer do the work

2. Let the computer do the work
   - 2.1 Make the computer repeat tasks
   - 2.2 Save recent commands in a file for re-use
   - 2.3 Use a build tool to automate workflows

Make incremental changes

3. Make incremental changes
   - 3.1 Work in small steps with frequent feedback and course correction
   - 3.2 Use a version control system
   - 3.3 Put everything that has been created manually in version control

Dont repeat yourself (or others)

4. Dont repeat yourself (or others)
   - 4.1 Every piece of data must have a single authoritative representation in the system
   - 4.2 Modularize code rather than copying and pasting
   - 4.3 Re-use code instead of rewriting it

Plan for mistakes

5. Plan for mistakes
   - 5.1 Add assertions to programs to check their operation
   - 5.2 Use an off-the-shelf unit testing library
   - 5.3 Turn bugs into test cases
   - 5.4 Use a symbolic debugger

Optimize software only after it works correctly

6. Optimize software only after it works correctly
   - 6.1 Use a profiler to identify bottlenecks
   - 6.2 Write code in the highest-level language possible

Document design and purpose, not mechanics

7. Document design and purpose, not mechanics
   - 7.1 Document interfaces and reasons, not implementations
   - 7.2 Refactor code in preference to explaining how it works
   - 7.3 Embed the documentation for a piece of software in that software

Collaborate

8. Collaborate
   - 8.1 Use pre-merge code reviews
   - 8.2 Use pair programming when bringing someone new up to speed and when tackling particularly tricky problems
   - 8.3 Use an issue tracking tool

R packages

An environment with functions and/or data

The way to share code and data

4 000 developers
>7000 packages

Package basics

Usage

`library()`

`::`

`:::`

Installation

`install.packages()`

`devtools::install_github()`

`devtools::install_local()`

Package namespace

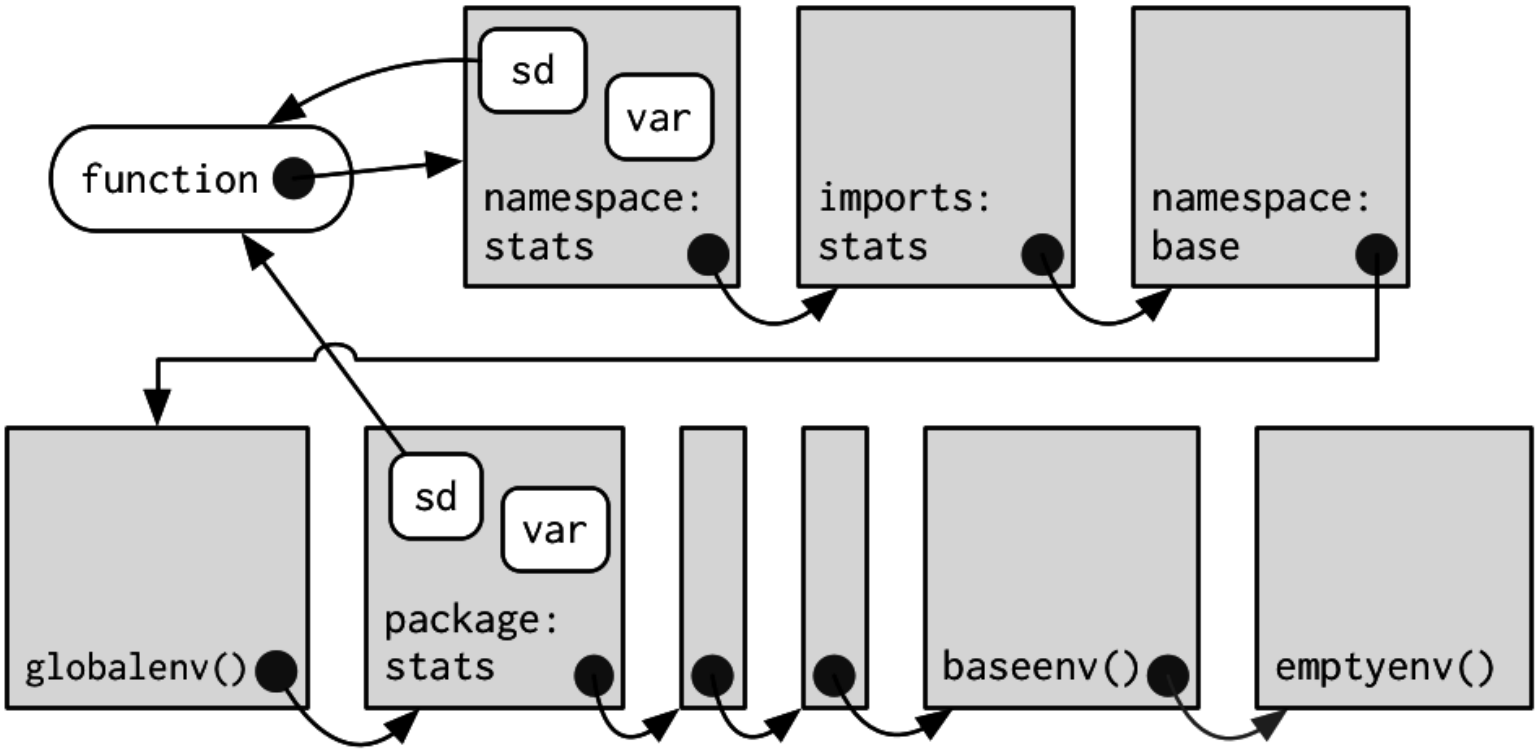

Figure: Package namespace

Which are good packages

Examine the package

1. Who?
2. When updated?
3. In development?

What is Version control?

Video!!
Version Control

Why version control?

1. Collaboration
2. Storing versions (properly)
3. Restoring versions
4. Understanding what happens
5. Backup

Why git?

1. Simple to use
2. Distributed
3. Fast
4. Common in practice
5. R packages uses github
6. Integrated with R-Studio

Why git?

1. Simple to use
2. Distributed
3. Fast
4. Common in practice
5. R packages uses github
6. Integrated with R-Studio

Created by Linus Torvalds! ;)

Basic git

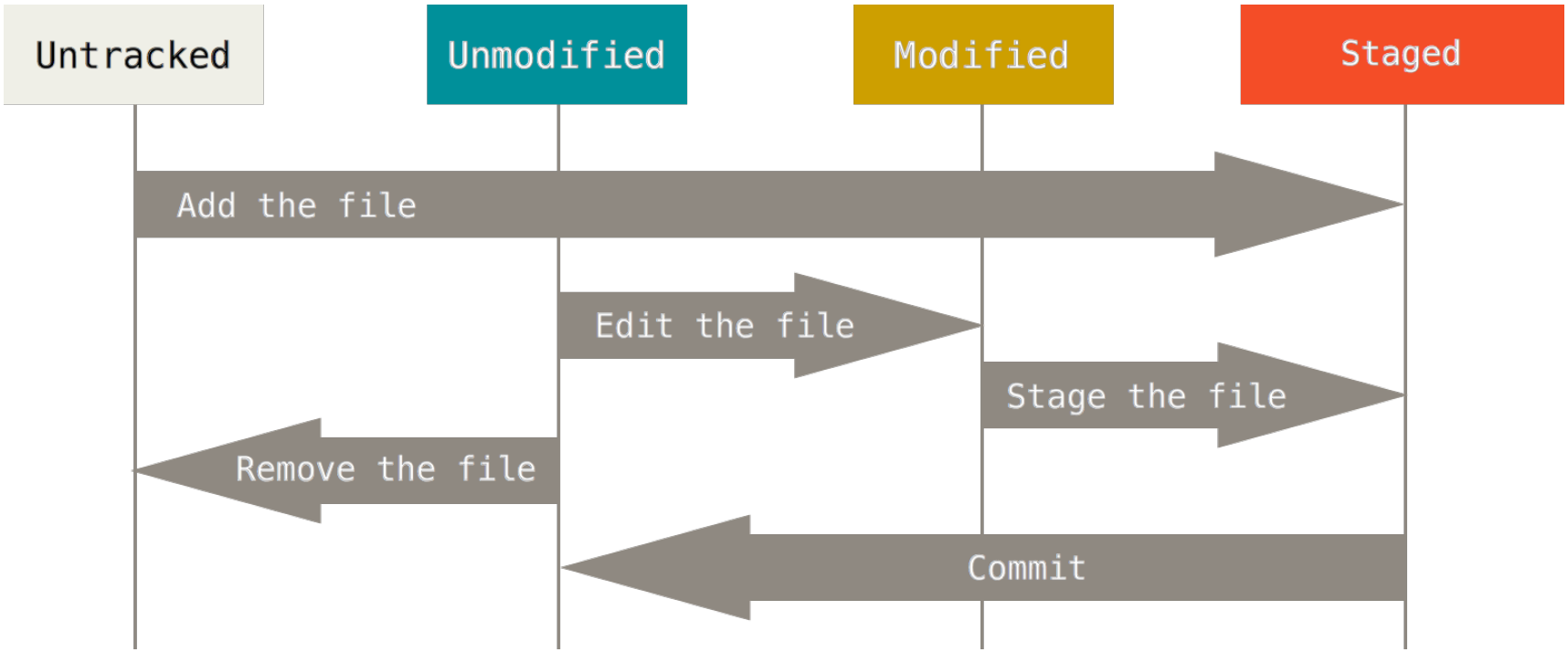

Figure: File life cycle

GitHub

1. Remote (push/pull)
2. Barebone homepage (using md)
3. Collaborations
4. Issue tracker / Wiki / discussions

Free for public repos
Private repos cost
Student accounts

Why part of the course?

Writing performant code (best practice)
The way to collaborate (R ecosystem)
Combine code, data and analysis
Easy to distribute and reuse (public api)

Learn how to reuse code from other packages

Package structure

Package structure

`DESCRIPTION`

Package structure

```
DESCRIPTION
NAMESPACE
```

Package structure

```
DESCRIPTION
NAMESPACE
R/
```

Package structure

```
DESCRIPTION
NAMESPACE
R/
man/
```

Package structure

```
DESCRIPTION
NAMESPACE
R/
man/
vignette/
```

Package structure

```
DESCRIPTION
NAMESPACE
R/
man/
vignette/
tests/
```

Package structure

```
DESCRIPTION
NAMESPACE
R/
man/
vignette/
tests/
data/
```

Package structure

```
DESCRIPTION
NAMESPACE
R/
man/
vignette/
tests/
data/
scr/
```

Package structure

```
DESCRIPTION
NAMESPACE
R/
man/
vignette/
tests/
data/
scr/
inst/
```

Why roxygen2?

1. Performant code (docs close to code)
2. Automatically generates all man files
3. Simple to use
4. Handles NAMESPACE
5. Similar to JavaDoc and DOxygen

roxygen2 syntax

[example]

sweidnumbr
Full support in R-Studio

Why unit testing?

Fewer bugs
Better code structure
Faster restarts
Robust code - correct a bug only once

A must in complicated projects!

Types of testing

1. White box testing
2. Black box testing
3. Probabilistic testing

testthat

Unit testing framework for R
Integrated with R-Studio

[example]

sweidnumbr testsuite

Introduction to Debugging in R

Another Video!!
Debugging in R

The End... for today.
Questions?
See you next time!