

# **R packages I: Finding, Installing, and building R Packages**

# What is an R Package?

- In a nutshell, an R package is a collection of functions and datasets, with manuals.
- Remember one needs to “source” a user defined function into the workspace, or “load” a pre-saved dataset. But
  - It’s troublesome to source/load many functions/datasets.
  - Cannot provide function helps in R.
  - Functions written in other language (C, Fortran) depend on OS, so need to be recompiled.
  - More advanced issues: accessibility of functions (Namespace).

# What is an R Package?

*“R Packages allow for easy, transparent and cross-platform extension of the R base system.”*

*“R package is a comfortable way to maintain collections of R functions and data sets. As an article distributes scientific ideas to others, a package distributes statistical methodology to others.”*

*[cran.r-project.org/doc/contrib/Leisch-CreatingPackages.pdf](http://cran.r-project.org/doc/contrib/Leisch-CreatingPackages.pdf)*

**Nowadays, it's typical (often expected) to have an R package with a new statistical method.**

# Terminologies for R Packages

Let's clarify some terms which sometimes get confused:

- **Package:** An extension of the R base system with code, data and documentation in standardized format.
- **Library:** A directory containing installed packages.
- **Repository:** A website providing packages for installation.
- **Package source:** The original version of a package with human-readable text and code.

- **Binary package:** A compiled version of a package with computer-readable text and code, may work only on a specific platform. So Each OS has its own version of the binary package.
- **Base packages:** Part of the R source tree, maintained by R Core team.
- **Recommended packages:** Part of every R installation, but not necessarily maintained by R Core.
- **Contributed packages:** All the rest. Contributed by developers and researchers. Many contributed packages on CRAN are written and maintained by R Core members.

# All R functions are distributed with Packages

- The functions don't "float around" within R. Every single function in R belongs to a package.
- Most functions we've seen are in **base** or **stats** packages (two major base packages).
- Currently loaded package can be seen by calling **sessionInfo**:

```
> sessionInfo()  
R version 3.2.1 (2015-06-18)  
Platform: x86_64-apple-darwin13.4.0 (64-bit)  
Running under: OS X 10.10.4 (Yosemite)
```

```
locale:[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-  
8/en_US.UTF-8
```

```
attached base packages:
```

```
[1] stats      graphics  grDevices  utils      datasets  methods    base
```

# All R functions are distributed with Packages

For a function, the package it belongs to can be seen in the help page:

```
> ?sum
```

sum

package:base

R Documentation

Sum of Vector Elements

Description:

‘sum’ returns the sum of all the values present in its arguments.

Usage:

```
sum(..., na.rm = FALSE)
```

# **Finding and installing R packages**



# Where to obtain R packages

There are several Repositories where one can obtain R packages, including:

- CRAN: The Comprehensive R Archive Network. This is the “official” R package distribution center.
- Bioconductor: for bioinformatics related R packages.
- R-Forge: similar to CRAN but better. It provides an “ecosystem” for R package developments, including version control system and forums/ mailing list.
- Researchers’ website.

# CRAN: The Comprehensive R Archive Network

## Contributed Packages

### Available Packages

Currently, the CRAN package repository features 4342 available packages.

[Table of available packages, sorted by date of publication](#)

[Table of available packages, sorted by name](#)

### Installation of Packages

Please type `help("INSTALL")` or `help("install.packages")` in R for information on how to install packages from this repository. The manual [R Installation and Administration \[PDF\]](#) (also contained in the R base sources) explains the process in detail.

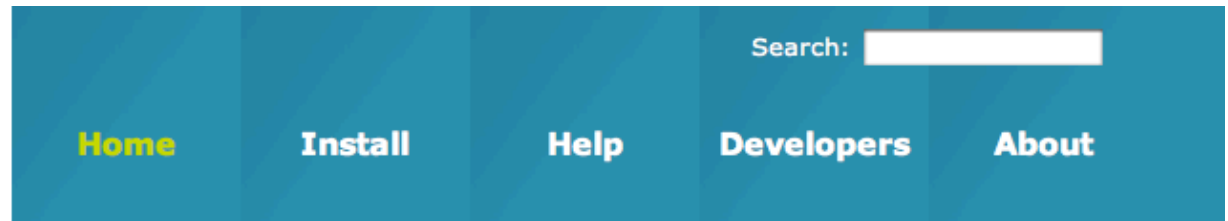
[CRAN Task Views](#) allow you to browse packages by topic and provide tools to automatically install all packages for special areas of interest. Currently, 30 views are available.

### Package Check Results

All packages are tested regularly on machines running [Debian GNU/Linux](#), [Fedora](#) and Solaris. Packages are also checked under MacOS X and Windows, but typically only on the day the package appears on CRAN.

The results are summarized in the [check summary](#) (some [timings](#) are also available). Additional details for Windows checking and building can be found in the [Windows check summary](#).

# Bioconductor



## About Bioconductor

Bioconductor provides tools for the analysis and comprehension of high-throughput genomic data. Bioconductor uses the R statistical programming language, and is open source and open development. It has two releases each year, [610 software packages](#), and an active user community. Bioconductor is also available as an [Amazon Machine Image \(AMI\)](#).



## Use Bioconductor for...

### ➤ [Microarrays](#)

Import Affymetrix, Illumina, Nimblegen, Agilent, and other platforms. Perform quality assessment, normalization, differential expression, clustering, classification, gene set enrichment, genetical genomics and other workflows for expression, exon, copy number, SNP, methylation and other assays. Access GEO, ArrayExpress, Biomart, UCSC, and other community resources.

### ➤ [Variants](#)

Read and write VCF files. Identify structural location of variants and compute amino acid coding changes for non-synonymous variants. Use SIFT and PolyPhen database packages to predict consequence of amino acid coding changes.

### ➤ [Annotation](#)

Use microarray probe, gene, pathway, gene ontology, homology and other annotations. Access GO, KEGG, NCBI, Biomart, UCSC, vendor, and other sources.

### ➤ [High Throughput Assays](#)

Import, transform, edit, analyze and visualize flow cytometric, mass spec, HTqPCR, cell-based, and other assays.

# R-Forge



Project  Search

[Log In](#) | [New Account](#)

[Home](#)

[My Page](#)

[Projects](#)

## What are R and R-Forge?

**R** is 'GNU S', a freely available language and environment for statistical computing and graphics which provides a wide variety of statistical and graphical techniques: linear and nonlinear modelling, statistical tests, time series analysis, classification, clustering, etc. Please consult the [R-project homepage](#) for further information.

**R-Forge** offers a central platform for the development of R packages, R-related software and further projects. It is based on [FusionForge](#) offering easy access to the best in SVN, daily built and checked packages, mailing lists, bug tracking, message boards/forums, site hosting, permanent file archival, full backups, and total web-based administration.

## A Platform for the Whole R Community

In order to get the most out of R-Forge you'll need to [register as a site user](#) and then [login](#). This will allow you to participate fully in all we have to offer, e.g., you may [register your project](#). Of course, you may also browse the site without registration, but will only have limited access to some features. For details see the documentation.

## Documentation

- [Short Introduction](#): Stefan Theußl and Achim Zeileis. Collaborative software development using R-Forge. *The R Journal*, 1(1):9-14, May 2009. URL <http://journal.R-project.org/> [[bib](#)] [[pdf](#)] [[local copy](#)](Official R-Forge citation)
- [User's Manual](#): [[pdf](#)] (*Detailed technical documentation*)

If you experience any problems or need help you can submit a [support request](#) to the R-Forge team or write an email to [R-Forge@R-Project.org](mailto:R-Forge@R-Project.org). Thanks... and enjoy the site.

## Tag Cloud

Bayesian **Bioinformatics** Multivariate  
Regression **R** biostatistics classification clustering data  
mining ecology finance mixed effect models mixed  
model model estimation multivariate optimization  
**spatial** **spatial data** spatial methods spatio-  
temporal time series visualization

## R-Forge Statistics

Hosted Projects: **1,477**  
Registered Users: **6,054**

## Most Active This Week

( 100.0% ) **R.\* packages**  
( 99.3% ) **vegan - Community Ecology Package**  
( 98.6% ) **NMF - Nonnegative Matrix Factorization**  
( 97.9% ) **iHELP**  
( 97.2% ) **Bayesian AuTomed Metabolite Analyser**

# A few remarks for R packages

- There's no guarantee for bug free (as for all free software).
- The major packages (base, stat, util, etc.) should work fine.
- There are a lot of badly written packages.
- Packages are updated often (especially Bioconductor packages), and backward compatibility is not guaranteed. This means sometimes your old codes stop working with the new package.
- To get help, use mailing list (see <http://www.r-project.org/mail.html>), or ask the developer directly.

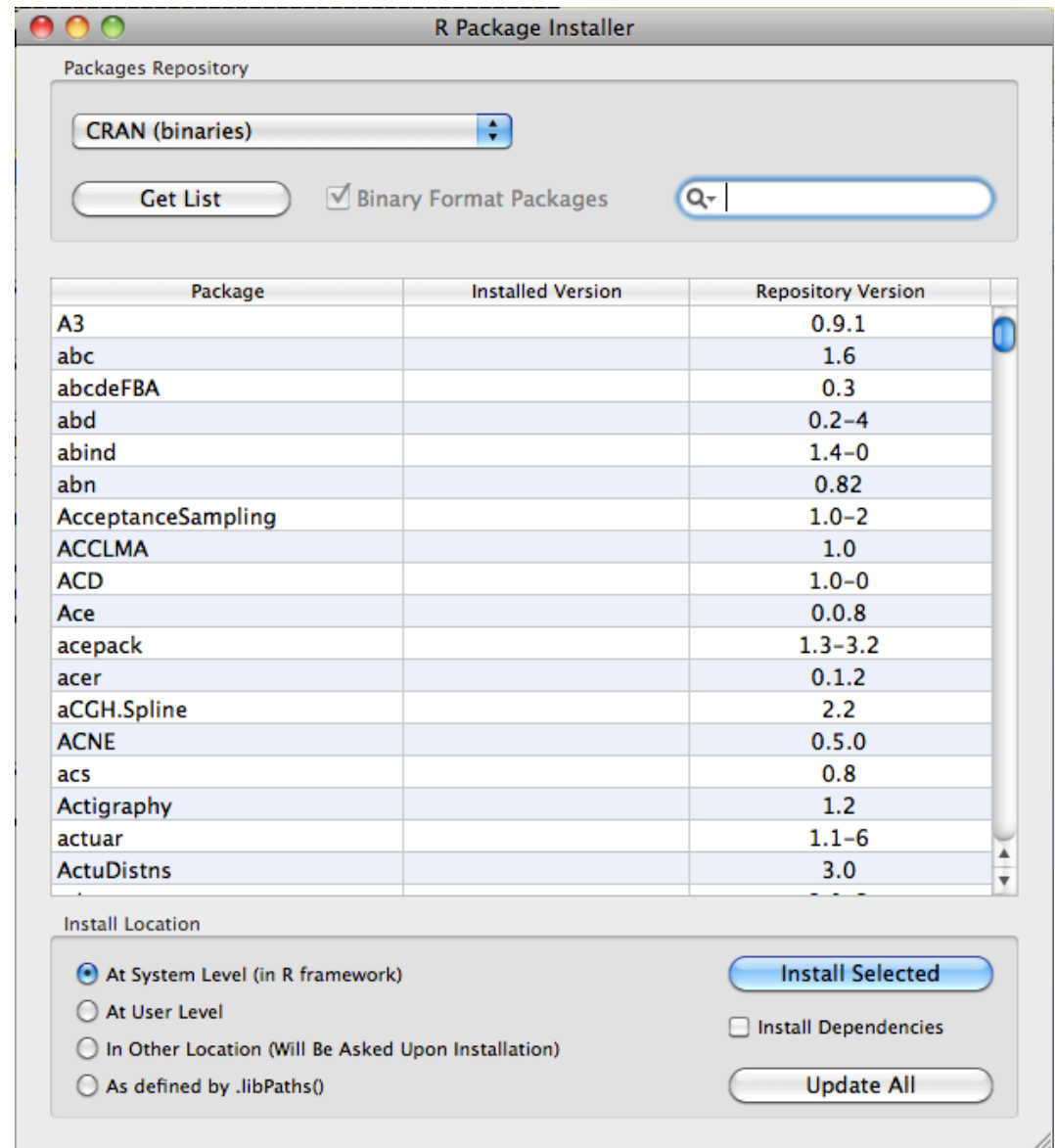
# R package installation

- Can be done in different ways:
  - Use GUI.
  - In R console.
  - Use command line.
- Using the GUI or R console is easier. It will determine the correct platform and R version, also install dependence packages.
- It's tricky to use command line to install from **package source**. But sometimes one has to.
- Packages will be installed to the **Library**.

# Install a new R package - using the GUI

## On Mac:

- Under “Packages & Data” menu, click “Package Installer” and get a dialog window.
- Choose a repository and specify the location, then install.



# Install a new R package - using the GUI

## On Windows:

For CRAN packages:

- Click “Packages” menu, then select “Install package(s) ...”. If you do this the first time, you will be asked to choose a CRAN mirror set. Select “USA(TN)”.
- Click the package you want to install, then click OK. Again if it’s the first time, you’ll be asked to create a personal library, just click OK.

For packages not on CRAN. You will have to have the package source as a zip or gz file.

- Click “Packages” menu, then select “Install package(s) from local zip files ...”.



# Install a new R package - in R console

- For packages from CRAN, use the “**install.packages**” function to install.

```
> install.packages("ggplot2")  
also installing the dependencies 'digest', 'gtable', 'reshape2', 'proto'
```

```
trying URL 'http://cran.r-  
  project.org/bin/macosx/leopard/contrib/2.15/digest_0.6.3.tgz '  
Content type 'application/x-gzip' length 161113 bytes (157 Kb)  
opened URL
```

```
=====
```

```
downloaded 157 Kb
```

```
trying URL 'http://cran.r-  
  project.org/bin/macosx/leopard/contrib/2.15/gtable_0.1.2.tgz '  
Content type 'application/x-gzip' length 60865 bytes (59 Kb)  
opened URL
```

```
=====
```

# Install a new R package - in R console

- For packages from Bioconductor, use the “biocLite” function to install.

```
> source("http://bioconductor.org/biocLite.R")
Bioconductor version 2.11 (BiocInstaller 1.8.3), ?biocLite for help

> biocLite("GenomicFeatures")
BioC_mirror: http://bioconductor.org
Using Bioconductor version 2.11 (BiocInstaller 1.8.3), R version 2.15.
Installing package(s) 'GenomicFeatures'
also installing the dependencies 'IRanges', 'GenomicRanges'

trying URL
  'http://bioconductor.org/packages/2.11/bioc/bin/macosx/leopard/contrib/2.1
  5/IRanges_1.16.5.tgz'
Content type 'application/x-gzip' length 2052325 bytes (2.0 Mb)
opened URL
=====
downloaded 2.0 Mb
```

# Install a new R package - in command line

- If you are familiar with Linux style command line, you can obtain the package code (usually in a .tar.gz format), and issue the following command to install:

```
R CMD INSTALL pkg_1.0.tar.gz
```

- There's no compelling reason to do it this way unless you have to.

# Setting up personal R library










**This is important if you are using a Linux machine, or a Mac but you don't have root access.**

- The R “root library” is not accessible to you (you have no write permission, so can't install package).
- You can create a personal R library by doing:
  - create a directory somewhere you have access, say, C:\Rlib or ~/Rlib.
  - create a file call “**.Renvi**ron” (start with a dot) containing following line: `R_LIBS=~/Rlib`.
- Every time R start, it'll read the file and know that you have a personal library at ~/Rlib.

# Creating R packages

# The structure of an R package

- The source of an R package is saved in a directory with following files and directories:

Name	
	CHANGES
	DESCRIPTION
	NAMESPACE
	data
	demo
	inst
	man
	R
	src

# The structure of an R package

- Three simple text files:
  - **DESCRIPTION** (required): description of the package.
  - **CHANGES** (optional): a log of changes in the codes.
  - **NAMESPACE** (optional): define the package “namespace”.
- Directories:
  - **R** (required): all R source codes.
  - **man** (optional): help files (in Rd).
  - **src** (optional): source codes in other languages (C/Fortran).
  - **data** (optional): data (in RData) distributed with the package.
  - **inst** (optional): additional instruction files (like software manual).
  - **demo** (optional): demo scripts.

# Create an R package

- The easiest way is to use “`package.skeleton`” function, which creates the “skeleton” of the package, e.g., empty directories and files.
- For example, if we want to create a package called “coolpkg”:

```
> package.skeleton("coolpkg")
Creating directories ...
Creating DESCRIPTION ...
Creating NAMESPACE ...
Creating Read-and-delete-me ...
Saving functions and data ...
Making help files ...
Done.
```

Further steps are described in `'./coolpkg/Read-and-delete-me'`.



## **“Read-and-delete-me” file**

- \* Edit the help file skeletons in 'man', possibly combining help files for multiple functions.
- \* Edit the exports in 'NAMESPACE', and add necessary imports.
- \* Put any C/C++/Fortran code in 'src'.
- \* If you have compiled code, add a `useDynLib()` directive to 'NAMESPACE'.
- \* Run **R CMD build** to build the package tarball.
- \* Run **R CMD check** to check the package tarball.

Read "Writing R Extensions" for more information.

## Steps after package.skeleton

1. Modify DESCRIPTION file to include information about the package.
2. Modify NAMESPACE file for package namespace \*.
3. Put R codes into the R directory.
4. Put C/Fortran codes (if any) into the src directory \*.
5. Go to man directory and create help files.
6. Create the package (as a single zip file).

\* if necessary.

# Package files - DESCRIPTION

The **DESCRIPTION** file contains basic information about the package in the following format:

```
Package: coolpkg
Type: Package
Title: What the package does (short line)
Version: 1.0
Date: 2016-02-04
Author: Who wrote it
Maintainer: Who to complain to <yourfault@somewhere.net>
Description: More about what it does
License: What license is it under?
```

# Packages files - DESCRIPTION

**Package:** `coolpkg`

The mandatory ‘Package’ field gives the name of the package. This should contain only letters, numbers or a dot, have at least two characters and start with a letter and not end in a dot.

**Version:** `1.0`

The mandatory ‘Version’ field gives the version of the package. This is a sequence of at least *two* (and usually three) non-negative integers separated by single ‘.’ or ‘-’ characters.

# Packages files - DESCRIPTION

**License: GPL-2**

The mandatory ‘License’ field should specify the license of the package in a standardized form. Alternatives are indicated *via* vertical bars. Individual specifications must be one of the “standard” short specifications: GPL-2 GPL-3 LGPL-2 LGPL-2.1 LGPL-3 AGPL-3 Artistic-1.0 Artistic-2.0. Some other examples:

**License: GPL (>= 2) | BSD**

**License: LGPL (>= 2.0, < 3)**

# Packages files - DESCRIPTION

## **Description: It does great things**

The mandatory ‘Description’ field should give a comprehensive description of what the package does. One can use several (complete) sentences, but only one paragraph.

## **Title: CoolPkg**

The mandatory ‘Title’ field should give a short description of the package. Some package listings may truncate the title to 65 characters. It should be capitalized, not use any markup, not have any continuation lines, and not end in a period.

# Packages files - DESCRIPTION

**Author: John Doe**

The mandatory ‘Author’ field describes who wrote *the package*. It is a plain text field intended for human readers

**Maintainer: <youremail@somewhere.net>**

The mandatory ‘Maintainer’ field should give a *single* name with a *valid* (RFC 2822) email address in angle brackets (for sending bug reports etc.). It should not end in a period or comma.

# Packages files - DESCRIPTION

**Date: 2014-01-17**

The ‘Date’ field gives the release date of the current version of the package. It is strongly recommended to use the yyyy-mm-dd format conforming to the ISO 8601 standard.

**Depends: R (>= 2.11.0)**

The optional ‘Depends’ field gives a comma-separated list of package names which this package depends on. The package name may be optionally followed by a comment in parentheses.



# Other optional contents in an R package

- Function helps
  - Need to create .Rd files and save under /man directory.
- R data
  - Need Need to save .RData files and save under /data directory.
- R vignette
  - Need to use Sweave.
- We will cover these next lecture.

# Checking the Package

- You can run a comprehensive checking of the package to get detailed information. For that do in command line window:

## **R CMD check coolpkg**

```
$ R CMD check coolpkg
using log directory
'/Users/hwu30/Dropbox/mine/teaching/Rintro/package/test/coolpkg.Rcheck'
...
* checking whether package 'coolpkg' can be installed ... WARNING
Found the following significant warnings:
  Warning: /Users/hwu30/Dropbox/mine/teaching/Rintro/package/test/coolpkg/man/coolpkg-
package.Rd:33: All text must be in a section
See
'/Users/hwu30/Dropbox/mine/teaching/Rintro/package/test/coolpkg.Rcheck/00install.out'
for details.
```

- If there are errors, they need to be corrected. Warnings are okay. However submitting to CRAN requires no warning from R CMD check.

# Building the Package

- After creating all necessary files, we are in place to build up the package.
- This can be done in command line using “R CMD build”. After that you’ll get a file called “coolpkg\_1.0.tar.gz” under current directory, which is the package source.

```
$ R CMD build coolpkg
* checking for file ‘coolpkg/DESCRIPTION’ ... OK
* preparing ‘coolpkg’:
* checking DESCRIPTION meta-information ... OK
* checking for LF line-endings in source and make files
* checking for empty or unneeded directories
* building ‘coolpkg_1.0.tar.gz’
```

Congratulations ! You now have a package that you can distribute to someone else or post on CRAN for sharing with the world.

# Some notes on Windows

It is a little troulber to build package on Windows, because some tools (like Perl) are not installed by default.

There are some instructions online:

- <http://www.biostat.wisc.edu/~kbroman/Rintro/Rwinpack.html>
- <http://www1.appstate.edu/~arnholta/Software/MakingPackagesUnderWindows.pdf>

Basically you need to:

1. Install additional software.
2. Setup environment variables.
3. Use the same command in command window.

# Review

What we've covered today:

- How to find and install R packages.
- How to write R packages.

Next lecture, we will cover:

- Writing R function helps.
- Writing R vignette using Sweave.