

**Prospects and Challenges for CRAN –  
with a glance on Ubuntu binaries**

Uwe Ligges, Kurt Hornik, Duncan Murdoch,  
Brian Ripley, Stefan Theußl  
Firstname.Lastname@R-project.org

The R Project for Statistical Computing

useR! 2010, Gaithersburg

**Prospects and Challenges for CRAN –  
with a glance on 64-bit Windows binaries**

Uwe Ligges, Kurt Hornik, Duncan Murdoch,  
Brian Ripley, Stefan Theußl  
Firstname.Lastname@R-project.org

The R Project for Statistical Computing

useR! 2010, Gaithersburg

# Contents / Introduction

- Cornerstones of R's success:
  - Decentralized and modularized way of creating software
  - Standardized format of packages and package system
  - Accessibility by a broad audience:  
standardized repositories

# Contents / Introduction

- Cornerstones of R's success:
  - Decentralized and modularized way of creating software
  - Standardized format of packages and package system
  - Accessibility by a broad audience:  
standardized repositories
- CRAN repository contains more than 2460 packages
  - Ease of installing packages
  - Checks help to provide at least some kind of quality

# Contents / Introduction

- Cornerstones of R's success:
  - Decentralized and modularized way of creating software
  - Standardized format of packages and package system
  - Accessibility by a broad audience:  
standardized repositories
- CRAN repository contains more than 2460 packages
  - Ease of installing packages
  - Checks help to provide at least some kind of quality
- To support such a loosely coupled development model:  
verification of certain formal quality criteria

# Contents / Introduction

- Established quality assurance systems and collaborative infrastructures typically face several challenges
- More and more has been or has to be automated:  
R package management system has gone a long and successful way to also support repository maintainers, but we have to go further
- Services:
  - Check result summaries
  - Binaries
  - Other support of the loosely coupled development model

# Products

We can think of

- packages as 'products',
- potential users as 'customers',
- package repositories like CRAN: 'warehouse'-like storage areas.

# Products

We can think of

- packages as 'products',
- potential users as 'customers',
- package repositories like CRAN: 'warehouse'-like storage areas.

If the 'right' product is not available:

- Easy for customers to become contributors: creating new package to fill the gap
- Decentralized and modularized way of creating software
- Rather than reinventing the wheel, package authors wisely reuse code of other packages

# Repositories

- CRAN, BioConductor, Omega(hat), R-Forge, and many others
- Publication mechanism for CRAN:
  - Submit contributed source package via ftp upload
  - Notify CRAN maintainers
  - Package checked by a CRAN maintainer
  - If fine, package is included for provision.
  - Binary versions are created and checked
  - Regular checks

# Other OSS Repositories

Other Open Source Software Repositories:

- Debian GNU Linux
- Other Linux distributions
- Perl (CPAN)
- ...

Debian GNU Linux offers a very sophisticated package management system and pursues an approach similar to CRAN:

- Relationships between packages declared in the package's *control* file  
(Depends, Recommends, Suggests, Enhances, etc.)
- Packages are checked for certain formal quality criteria

# Dependencies

- Packages might depend on other packages that again depend on ...
- Hierarchy of dependencies could be broken by a simple bug in one of the packages:  
implications on the interoperability between packages
- Check system that allows for recursive checking
- Check system for huge package repositories should be parallelized:  
deal with thousands of packages while respecting dependency structures

# Dependency levels

Package maintainers specify dependencies in the DESCRIPTION file:

- Depends:** Package *B* depends on functionality in package *A* in such a way that package *A* is loaded in advance of *B*
- Imports:** Package *B* imports (parts of) the namespace of package *A* into its own namespace
- LinkingTo:** Package *B* links to compiled code in package *A*
- Suggests:** Some functionality or examples in the documentation of package *B* depend on package *A*
- Enhances:** Package *B* enhances packages *A* functionality but works without *A* being present

# Dependency levels

- *Depends*, *Imports* and *LinkingTo* must be fulfilled at installation time
- *Suggests* must typically be available when a package is checked
- Usually two different types of dependency graphs have to be calculated
  - Graphs needed for finding the correct installation order
  - Graphs needed for finding what have to be checked

# (Reverse) dependencies

Consider

- package  $B$  depends on package  $A$ ;  
formally denoted  $A \in d(B)$ , where  $d(B)$  entails all dependencies of  $B$
- packages  $C$  and  $D$  depend on package  $B$ ,  
i.e.  $A \in d_R(C), A \in d_R(D)$
- $d_R(D)$  denotes all recursive dependencies of package  $D$ ,  
i.e. the transitive closure of all dependencies of  $D$ .

then once  $A$  is updated, we definitely need to

- re-check packages  $A, B, C$ , and  $D$ ,

because newly introduced features or changes in  $A$  could have broken something somewhere else.

# (Reverse) dependencies

- Reverse dependencies of  $A$  denoted  $d^{-1}(A)$ , i.e., all packages depending on  $A$ .
- Recursive reverse dependencies:  $C \in d_R^{-1}(A)$  has to be considered for re-checking interoperability.
- With growth of CRAN: frequently an update of one package  $P$  breaks code in some dependent package(s)  $d_R^{-1}(P)$ .

# (Reverse) dependencies

- Reverse dependencies of  $A$  denoted  $d^{-1}(A)$ , i.e., all packages depending on  $A$ .
- Recursive reverse dependencies:  $C \in d_R^{-1}(A)$  has to be considered for re-checking interoperability.
- With growth of CRAN: frequently an update of one package  $P$  breaks code in some dependent package(s)  $d_R^{-1}(P)$ .
- Wonder why CRAN worked fairly well until 2008 without these checks

# (Reverse) dependencies

- We even may need – in addition to a new binary for *A* – some updated *binary* packages for *B*, *C* and *D*:  
e.g. if S4 classes and/or saved images are involved
- Find out which other packages are ‘involved’ (inclusive the recursive dependencies) in an update of a given package using

```
tools::dependsOnPkgs(pkgs,  
  dependencies = c("Depends", "Imports", "LinkingTo"),  
  recursive = TRUE, lib.loc = NULL,  
  installed = installed.packages(lib.loc,  
                                fields = "Enhances"))
```

# (Reverse) dependencies

Number of CRAN packages with 0, 1, . . . , and max. number of (recursive / reverse) dependencies:

dependencies	0	1	2	3	4	5	6	max
flat	809	573	365	227	180	97	57	19
recursive	809	265	182	126	93	120	110	33
flat reverse	1614	336	132	83	33	37	20	313
recursive reverse	1614	245	106	66	52	25	20	1141

based on dependency levels

- *Depends, Imports, LinkingTo*
- plus *Suggests* for flat reverse and recursive reverse dependencies

# (Reverse) dependencies

Selected CRAN packages with extreme number of (recursive / reverse) dependencies

dependencies	MASS	survival	Metabonomic	sisus
flat	4	4	19	18
recursive	4	4	33	31
flat reverse	313	127	0	0
recursive reverse	1065	1141	0	0

based on dependency levels

- *Depends, Imports, LinkingTo*
- plus *Suggests* for flat reverse and recursive reverse dependencies

# (Reverse) dependencies

- CRAN's dependency matrix is sparse
- Roughly half ( $> 1000$ ) of all CRAN packages 'depend' (recursively) on packages MASS or survival
- This is a problem given the runtime and many package updates a day — at least without allowing for parallel checks (and installs)

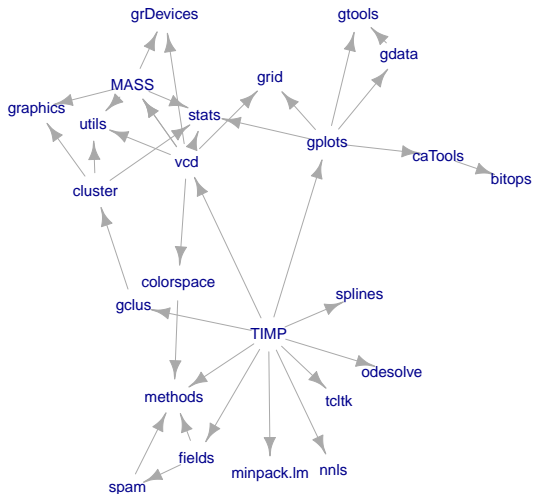
# (Reverse) dependencies

- CRAN's dependency matrix is sparse
- Roughly half ( $> 1000$ ) of all CRAN packages 'depend' (recursively) on packages MASS or survival
- This is a problem given the runtime and many package updates a day — at least without allowing for parallel checks (and installs)
- Let us take look at dependency graphs created with the help of package *igraph*

# (Reverse) dependencies

Dependency graph for package *TIMP*

dependency levels: *Depends*, *Imports* and *LinkingTo*



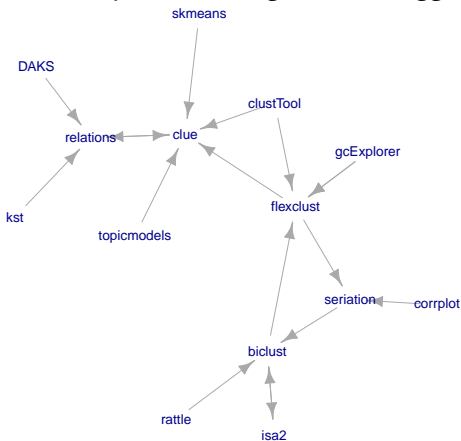
# (Reverse) dependencies

Graph representing the recursive reverse dependencies of package *Matrix* as needed in every new check of the given package;  
dep. levels: *Depends*, *Imports*, *LinkingTo*, and *Suggests*



# (Reverse) dependencies

Graph representing the recursive reverse dependencies of package *clue* as needed in every new check of the given package;  
dep. levels: *Depends*, *Imports*, *LinkingTo*, and *Suggests*



# Resources

- Combinations of different flavors of R:
  - branches: *devel*, *patched*, and *release*
  - platforms: Linux, Mac OS X, Solaris, Windows
  - architectures: SPARC, ix86, x86\_64
- Building and checking of packages on all combinations can become rather time consuming, particularly for checking reverse recursive dependencies
- development version of R changes over time: regular checks (up to daily)

# Computer resources

- Desirable to get check results for development processes early
- Build/check system required that
  - **finished within (at least!) 24 hours**
  - for each flavor of R in order to
  - provide the check results when needed, not thereafter
  - make binaries available in time during an R release cycle (e.g. the day after alpha/beta/rc/release)
- [http://CRAN.R-project.org/web/checks/check\\_timings.html](http://CRAN.R-project.org/web/checks/check_timings.html)

# Computer resources

Why should we 'improve' computer resources?

Tasks of a CRAN auto-build-and-check machine

- Make R-devel (3 times a week?)
- Build and check new and updated packages at least for R-release, R-devel, and R-oldrelease (?):
  - including reverse dependencies
  - each 6 hours
- Notifications for developers (at least in case of ERRORS?)
- Check summaries

# Computer resources

Why should we 'improve' computer resources?

Tasks of a CRAN auto-build-and-check machine

- Re-check all packages for R-devel (and R-patched?) on a regular (weekly?) basis  
**Aim:** make it possible to look out for errors for both R Core developers and package developers
- Provide the ftp check system for package developers as a service

# CRAN Windows Binaries' Package Check

Last updated on 2006-06-13 17:51:39 (**useR! 2006**) (simplified)

No	Package	Version	R-2.3.1	Inst. time	Check time
...	...	...	...	...	...
742	wavelets	0.2-1	OK	26	88
743	waveslim	1.5	OK	58	109
744	wavethresh	2.2-8	OK	30	75
745	wccsom	1.1.0	OK	18	87
746	wle	0.9-2	OK	24	365
747	xgobi	1.2-13	<b>ReadMe</b>		
748	xtable	1.3-2	OK	22	52
749	zicounts	1.1.4	<b>WARNING</b>	26	46
<b>750</b>	<b>zoo</b>	1.1-0	OK	23	60
SUM (in hours) on <b>Xeon 3.06 GHz:</b>				6.34	19.77
				<b>≈ 26 hours</b>	

# CRAN Windows Binaries' Package Check 2010

Last updated on 2010-07-16 19:50:06 (last Friday) (simplified)

No	Package	Version	R-2.11.1	Inst. time	Check time
...	...	...	...	...	...
2458	zic	0.5-3	OK	36	17
2459	zipfR	0.6-5	OK	7	63
2460	zoeppritz	1.0-2	OK	1	16
2461	zoo	1.6-4	OK	4	62
2462	zyp	0.9-1	OK	1	18
Sum (in hours), <b>2x Xeon E5430 Quad:</b>				6.9/8	50.6/8
				<b>≈ 8 hours</b>	

# Parallel installations

Due to the requirements for the CRAN maintenance, parallel package installation has been made possible since R-2.9.0

- Per package locks rather than per library locks (OOLOCK)
- Package dependencies are calculated in R
- Written to a Makefile
- Finally resolved by 'make -jX'

# Parallel installations

Makefile (as used on CRAN):

```
PKG := packageA packageB packageC
PKG_INST := $(PKG:=-install.out)
all: $(PKG_INST)
%-install.out: %
    MAKE=make MAKEFLAGS= R -f install.R\  
        --vanilla --quiet --args Path/to/library\  
        build $< R_default_packages=NULL
packageA-install.out: packageB-install.out
...
[one line for each package]
```

# Parallel installations

## Parallel installations on user level

- within R

```
install.packages(pkgs, lib, . . . . .,  
                 Ncpus = getOption("Ncpus"), ...)
```

- which generates the Makefile and several instances of  
R CMD INSTALL --pkglock . . . . .  
are used (which is also possible on user level).

# Human resources

It is all automated, so what?

## Tasks of a repository maintainer

- Maintaining and adapting the scripts themselves
- Maintaining the hardware of a devoted machine
- Setting up repositories for new versions of R
- Handling errors that were not covered by the scripts
- Answering questions (for developers and users)
- Asking package maintainers to fix their packages

Quality management can be improved by moving as many tasks as possible from human to computational resources.

# Base system vs. packages

- Functionality of the base R system is defined, well tested, and the code base of the current “stable” branch does not change significantly
- New version of R is released twice a year
- Only bugfixes are provided between releases (‘patched’ flavor of R)
- For packages, contributors upload code without a specific schedule
- Release vs. (current) development platforms

# Solutions: R-Forge

R-Forge (<http://R-Forge.R-project.org>), for the distributed development approach of packages, offers:

- Central platform for the development of R packages and other projects
- Organized in 'project'
- Various tools and web-based features for software development, communication and other services
- SVN repository
- CRAN-style repository of R packages built from the committed source code
- Reflecting the development progress made in the repository

# Solutions: R-Forge

- Social networking?

# Solutions: R-Forge

- Social networking?
- Allow developer to check if some package update will break code in other packages before the CRAN release (expensive!)

# Solutions: Build and check systems

Package developers without access to Windows machines for building or testing purposes may use

- special service called *win-builder*
- <http://win-builder.R-project.org>
- Allows to upload source packages and provides corresponding Windows binaries and check results
- Such services may appear for more than just the Windows platform on R-Forge 'soon'

## ... glance on 64-bit Windows binaries

- 64-bit Windows binaries are available since early 2010
- Shortly after a suitable compiler collection (gcc-4.4.x, MinGW beta) was released
- Notification by Arm Gong on January 04, 2010
- First official release with R-2.11.0
- Allows for processes using > 2Gb
- Binaries distributed separately:  
separate installer, separate checks, separate binary repositories:  
CRAN-mirror/bin/windows/contrib/2.11  
CRAN-mirror/bin/windows64/contrib/2.11
- Speed: Seems to be faster than 32-bit – but with a different more recent compiler version

# CRAN checks (July 20, 2010)

[http://cran.r-project.org/web/checks/check\\_summary.html](http://cran.r-project.org/web/checks/check_summary.html):

Flavor	OK	WARN	ERROR	Total
r-devel-linux-ix86	2195	248	17	2460
r-devel-linux-x86_64-gcc-debian	2188	244	28	2460
r-devel-linux-x86_64-gcc-fedora	2197	244	20	2461
r-patched-linux-ix86	2229	220	11	2460
r-patched-linux-x86_64	2218	219	23	2460
r-patched-solaris-sparc	2128	197	113	2438
r-patched-solaris-x86	2128	202	108	2438
r-release-linux-ix86	<b>2230</b>	221	9	2460
r-release-macosx-ix86	2160	208	90	2458
r-release-windows-ix86	<b>2196</b>	216	13	2425
r-release-windows64-x86_64	<b>2170</b>	196	39	2405
r-oldrel-macosx-ix86	2159	172	113	2444
r-oldrel-windows-ix86	2217	168	40	2425

# 64-bit Windows binaries for R-2.12.x

We aim at having bi-arch binaries for R-2.12.x  
(release to be in October?)

- Both 32-bit and 64-bit binary parts in one package
- Many parts of base R and binary packages are independent of 32-bit vs. 64-bit
- Just one repository required
- Install and check time reduced:  
Only parts depending on 'bit' have to be done twice
- Shared libraries (DLLs) in './libs/i386/' , './libs/x64/'
- Same true for directories './bin/.' and './etc/.' in base R

# 64-bit Windows binaries for R-2.12.x

- MinGW-w64 '1.0' changed to gcc pre-4.5.1 (underscore conventions changed)
- Package authors using 'configure.win' or 'Makefile.win' are encouraged to switch to 'Makevars.win', if applicable

# Prospects and Challenges

Prospects of a loosely-coupled development approach are diverse:

- Rapid development
- Diversity
- Alternative approaches facing different aspects of implementations such as speed vs. accuracy

Challenges are diverse, many solutions that have been implemented / are shortly before being implemented:

- (Recursive / reverse) dependency calculations
- Parallelized checks

# Questions?

# Questions?

Questions?

Indeed, there are some open questions!

# Questions?

## Questions?

Indeed, there are some open questions!

Open issues and questions:

- Given a package is updated, all its reverse dependencies are re-built for a binary repository ...
- ... and distributed through all the CRAN mirrors

# Questions?

## Questions?

Indeed, there are some open questions!

Open issues and questions:

- Given a package is updated, all its reverse dependencies are re-built for a binary repository ...
- ... and distributed through all the CRAN mirrors
- Sometimes more than 300 packages on a single day ...
- ... while just a few of them really need to be re-built in binary form

# Questions?

## Questions?

Indeed, there are some open questions!

Open issues and questions:

- Given a package is updated, all its reverse dependencies are re-built for a binary repository ...
- ... and distributed through all the CRAN mirrors
- Sometimes more than 300 packages on a single day ...
- ... while just a few of them really need to be re-built in binary form
- Infrastructure that supports calculation of the necessity of a binary re-built not yet in place
- No mechanism in place to make `update.packages()` collect such (required) rebuilds

# Questions?

## Questions?

Indeed, there are some open questions!

Open issues and questions:

- Some better database like system for each library (no need to parse thousands of DESCRIPTION files) / each repository

# Questions?

## Questions?

Indeed, there are some open questions!

Open issues and questions:

- Some better database like system for each library (no need to parse thousands of DESCRIPTION files) / each repository
- Moving rest of perl code to R (as we have seen a speed gain when porting the INSTALL script from perl to R), 'Rd2 parser' (Duncan Murdoch), and hopefully also for the ported check scripts

# Questions?

## Questions?

Indeed, there are some open questions!

Open issues and questions:

- Some better database like system for each library (no need to parse thousands of DESCRIPTION files) / each repository
- Moving rest of perl code to R (as we have seen a speed gain when porting the INSTALL script from perl to R), 'Rd2 parser' (Duncan Murdoch), and hopefully also for the ported check scripts
- Being much stricter in CRAN maintenance

# References

- Csardi G, Nepusz T (2006): The igraph software package for complex network research. *InterJournal Complex Systems*: 1695.
- Jackson I, Schwarz C, et al. (2010): *Debian Policy Manual*, version 3.8.4.0.
- Theußl S, Ligges U, Hornik K (2010): Prospects and Challenges in R Package Development. *Computational Statistics*, to appear.
- Theußl S, Zeileis A (2009): Collaborative software development using R-Forge. *The R Journal* 1(1): 9–14.