COMO DESENVOLVER PACOTES NO R SEM USO .\src

Tutorial - Versão 2.0
Agosto de 2013

Ben Dêivide de Oliveira Batista

1 Introdução

Em busca de explicações para desenvolver pacotes no R, resolvemos escrever esse tutorial, tentando simplificar os estudos de quem está começando a construção de pacotes estatísticos. Esse tutorial será desenvolvido para o sistema operacional "WINDOWS 32 e 64 bits".

2 Ferramentas necessárias para construir o pacote R

Para desenvolvermos um pacote R no Windows, precisaremos dos seguintes programas:

Programa	Versão	Instalação	Busca
R	R-3.0.1	Obrigatório	⟨http://cran.r-project.org/bin/windows/base/⟩
Rtools	Rtools30	Obrigatório	⟨http://cran.r-project.org/bin/windows/Rtools/⟩
Miktex	2.9	Obrigatório	⟨http://miktex.org/download⟩
CMD	-	Obrigatório	Windows
Rstudio	v0.97	Opcional	⟨http://www.rstudio.com/ide/download/desktop⟩

Após a instalação dos programas é interessante reiniciar o computador para completar a instalação.

2.1 Importância dos programas obrigatórios

O programa **R** é o principal software dentre todos os outros citados acima, já que a finalidade dos pacotes, é gerar rotinas para esse software. O **Rtools** é um software auxiliar ao programa R, que dará as ferramentas necessárias para construir o pacote R. O R foi desenvolvido como ferramenta de desenvolvimento em ambiente Unix, porém para desenvolver aplicações do R em ambientes Windows é necessário emular um ambiente Unix. A ferramenta TOOLS simplesmente faz esta emulação. Usaremos as ferramentas do **Rtools** e as ferramentas do prórpio **R** via prompt de comando do Windows **CMD**. Para que não precisemos via linhas de comando informar aonde as ferramentas do Rtools estão, faremos as seguintes alterações no sistema operacional no Windows: **Menu Iniciar** > **Painel de Controle** > **Sistema** > **Configurações avançadas do sistema** > **Variáveis de Ambiente** > **Variáveis do Sistema** > **Variável** > **path**. Na variável **path**, iremos inserir quais os programas que queremos executar sem que precise informar o local no PC aonde este esteja. Assim, clicando em **EDITAR**, iremos inserir essa linha de comando, para Win 32bits:

```
C:\Rtools\bin;c:\Rtools\gcc-4.6.3\bin;C:\Program files\R\R-2.15.1\bin\i386
```

ou para Win 64bits:

```
C:\Rtools\bin;c:\Rtools\gcc-4.6.3\bin;C:\Program files\R\R-2.15.1\bin\x64
```

e posteriormente, clique em **OK**.

Esse passo é muito importante, pois daqui para frente, tudo funcionará dependendo das linhas de comando que foram inseridas. Assim, observe para que serve cada linha de comando.

```
EXECUTAR AS FERRAMENTAS DO RTOOLS => c:\Rtools\bin;
EXECUTAR AS BIBLIOTECAS DE COMPILAÇÃO=> c:\Rtools\gcc-4.6.3\bin;
EXECUTAR AS FERRAMENTAS DO R (Rcmd.exe) => c:\ProgramFiles\R\R-2.15.1\bin\i386;
```

Pronto, após isso, estamos com tudo pronto para desenvolver o pacote no programa R. A sequência desses passos podem ser vistos nas Figuras (1) e (2).

O **Miktex** dará suporte aos arquivos em .pdf que farão parte da estrutura do pacote, como por exemplo, o manual de como funcionará o pacote.

O **RStudio** facilitará a programação do R, pois torna muito mais fácil sua utilização.

3 Esqueleto do Pacote R

Como exemplo vamos supor que nosso pacote será chamado de "mypkg". Assim, a estrutura mais simples de um pacote R será da seguinte forma:

```
./mypkg
|-- DESCRIPTION
|-- NAMESPACE
|-- Read-and-delete-me
|-- R/
|   |-- opalgb.r
|-- man/
|   |-- mypkg-package.Rd
|   `-- soma.Rd
|   `-- subtra.Rd
|   `-- multi.Rd
|   `-- divisi.Rd
```

Funções:

- **DESCRIPTION**: Esse arquivo contém informações básicas sobre o pacote, no seguinte formato:

```
Package: mypkg
Type: Package
Title: Basic Mathematical Operations
Version: 1.0
Date: 2004-06-27
Author: Ben Deivide
Maintainer: Ben Deivide <ben.deivide@gmail.com>
Description: The Package calculates operations such as addition,
subtraction, multiplication and division.
```

```
License: GPL version 2 or newer
Depends: R (>= 1.6.0)
```

- **NAMESPACE:** Arquivo responsável para exportar o algoritmo das funções desejadas. Quando não se quer restringir nenhum algoritmo dentro do pacote, o próprio R, basta usar o seguinte comando:

```
exportPattern(``^[[:alpha:]]+'')
```

  Caso queira restringir os algoritmos, faz-se um arquivo, "arquivos.R", dentro da pasta **R** do pacote "mypkg", limitando a visualização dos algoritmos do pacote, como veremos mais a frente.

- **Read-and-delete-me:** Esse arquivo contém informações adicionais para construir o pacote. Essas informações são:

```
* 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.
```

  Quando visualizadas essas informações, o arquivo pode ser deletado do pacote.

- **.\R:** Essa pasta contém todos os algoritmos em R. Os algoritmos dentro dessa pasta não precisam está dentro do mesmo arquivo "arquivo.R", podendo está em arquivos separados, e mesmo assim, ainda continuam interligados, isto é, posso ter um algoritmo num "arquivo1.R" dependendo de outro algoritmo no "arquivo2.R". Com a compilação do pacote, todas as rotinas estão prontas para serem executadas. No nosso exemplo, temos apenas o arquivo "opalgb.r" contendo as funções de nossas rotinas.

- .**man**: Essa pasta tem por finalidade de armazenar todos os arquivos de ajuda do pacote. A extensão dos arquivo é ".Rd". Em nosso exemplo, temos cinco arquivos: um sendo o arquivo base para descrição geral do pacote, "mypkg-package.Rd", e os demais, sendo arquivos mais detalhados sobre cada uma das funções. Por exemplo, em nosso pacote, desenvolveremos quatro rotinas e cada rotina contém um arquivo explicando detalhadamente sua função. Esses arquivos são: "soma.Rd", "subtra.Rd", "multi.Rd" e "divisi.Rd". Um modelo de arquivo base é:

```
\name{mypkg}
\alias{mypkg}
\docType{package}
\title{Basic Mathematical Operations}
\description{
The package calculates operations such as addition, subtraction,
multiplication and division.
}
```

```
\details{
\tabular{ll}{
Package: \tab mypkg\cr
Type: \tab Package\cr
Version: \tab 1.0\cr
Date: \tab 2013-28-08\cr
License: \tab GPL/GNU\cr
}
}
\author{Maintainer: Ben Deivide <ben.deivide@gmail.com>}
\references{Venables. S programming. 2000}
\examples{
soma(3,7)
subtra(3,7)
multi(3,7)
divisi(3,7)
}
```

Não precisa se preocupar com o desenvolvimento desses arquivo, pois o próprio R, na construção do pacote, define as informações básicas do arquivo, ficando ao seu critério preencher o que está sendo pedido. Para maiores detalhes, acesse: ⟨http://cran.r-project.org/doc/manuals/R-exts.html⟩.

4 Funções do pacote

O nosso pacote, "mypkg", terá a finalidade de calcular as quatro operações básicas: soma, subtração, divisão e multiplicação. O arquivo "opalgb.r", contém essas rotinas:

```
#Pacote: mypkg

divisi <- function (a,b){
    calculo <-a/b
    return(calculo)}

multi <- function (a,b){
    calculo <-a*b
    return(calculo)}

soma <- function (a,b){
    calculo<-a+b
    return(calculo)}

subtra <- function (a,b){
    calculo <-a-b
    return(calculo)}
```

5 Construindo o pacote R

Inicialmente, iremos criar a estrutura do pacote utilizando o programa R, usando o comando **package.skeleton()** do pacote **utils**, sendo que este já é da base do R, não precisando carregá-

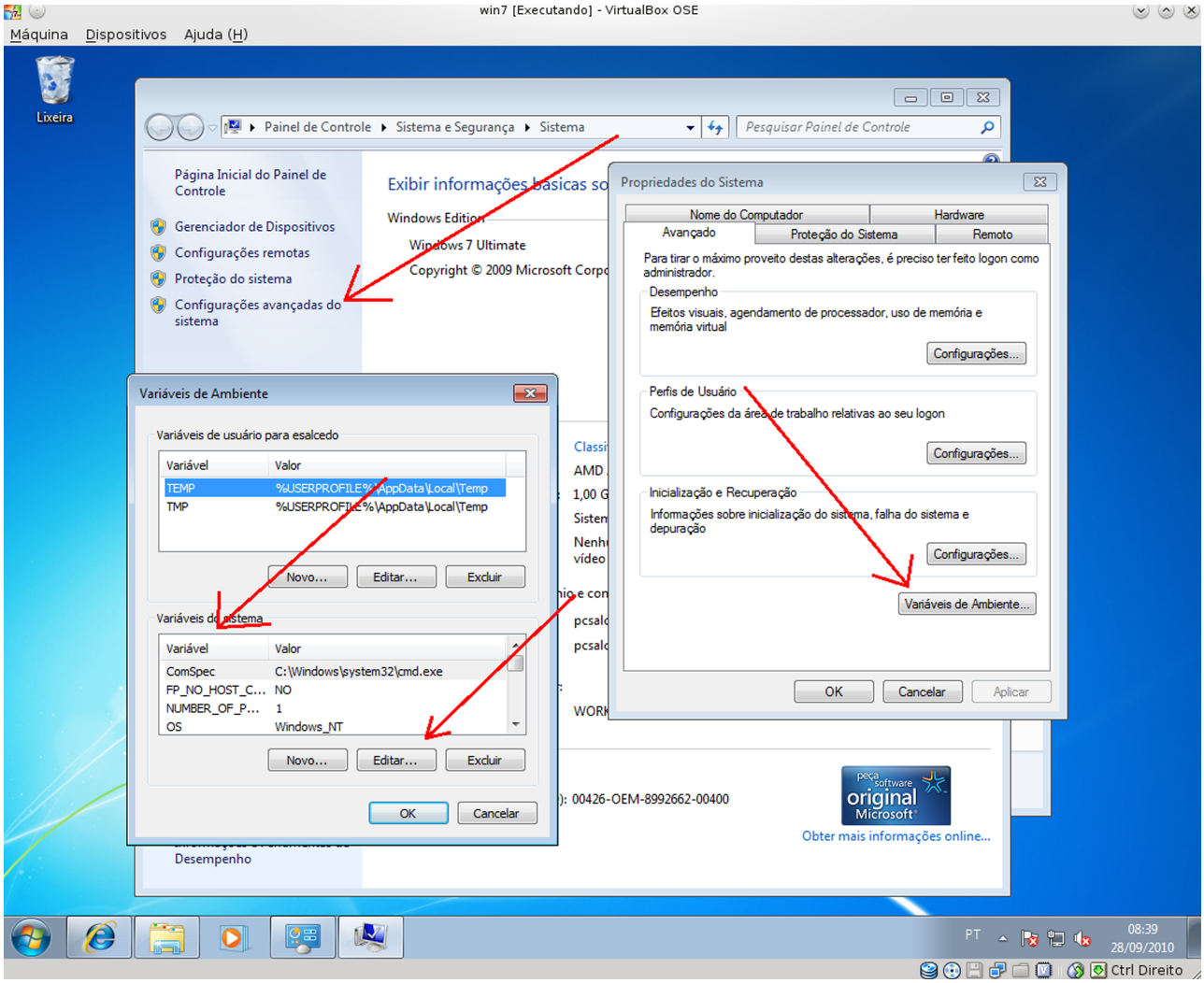

Figura 1: Configuração no Path.

lo. Assim, antes de executarmos o comando, iremos escolher o diretório ao qual faremos o pacote. No exemplo, escolheremos o seguinte diretório no console do R:

```
setwd(``c:/package'')
```

Em seguida, iremos rodar todas as funções desejáveis. Posteriormente, criaremos a estrutura do pacote, utilizando o comando no console do R,

```
package.skeleton(name = ``nome.pacote'', list , environment = .GlobalEnv,
path = ``.'', force = FALSE)
```

onde os argumentos:

- **name**: nome do pacote;
- **list**: vetor listando todos os objetos (funções) do R que irão fazer parte do pacote.
- **environment**: Se o argumento "list"for omitido, os índices deste ambiente serão empacotados, ou seja, todas as funções criadas na atual seção do R irão compor o pacote;

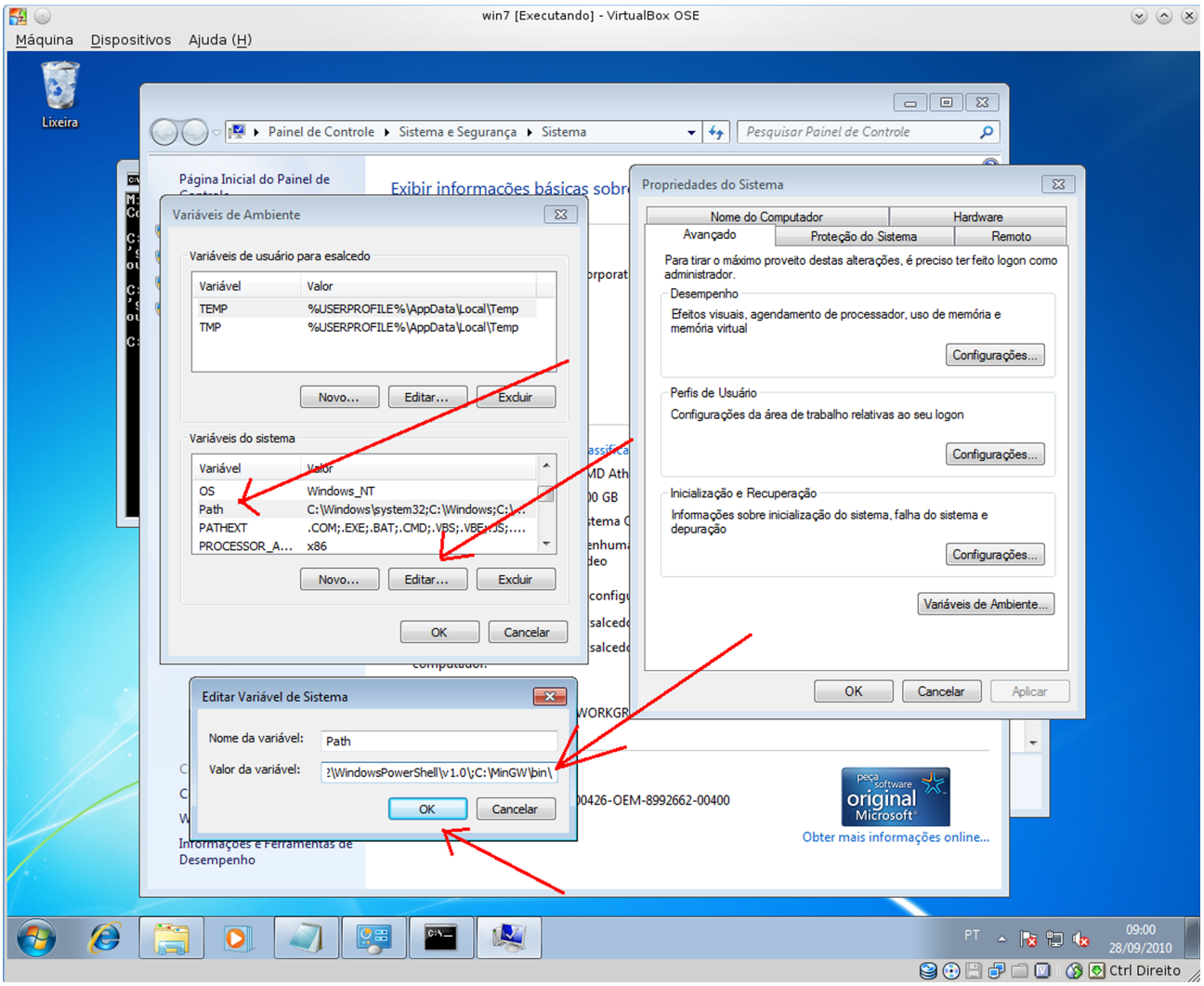

Figura 2: Configuração no Path.

- **path**: Caminho para colocar os diretórios dentro do pacote;
- **force**: Se for "FALSE"não sobrescreverá um pacote já existente.

sendo que se aparacer as seguintes observações,

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

implica que o comando gerou um diretório **mypkg** e os sub-diretórios, como já mencionado.

Outra forma mais simples para construir o pacote é usando o **RStudio**. Ao abri-lo, segue: Project > Create Project...; ou Project: (None)>Create Project..., como pode ser observado

nas Figuras (3) e (4).

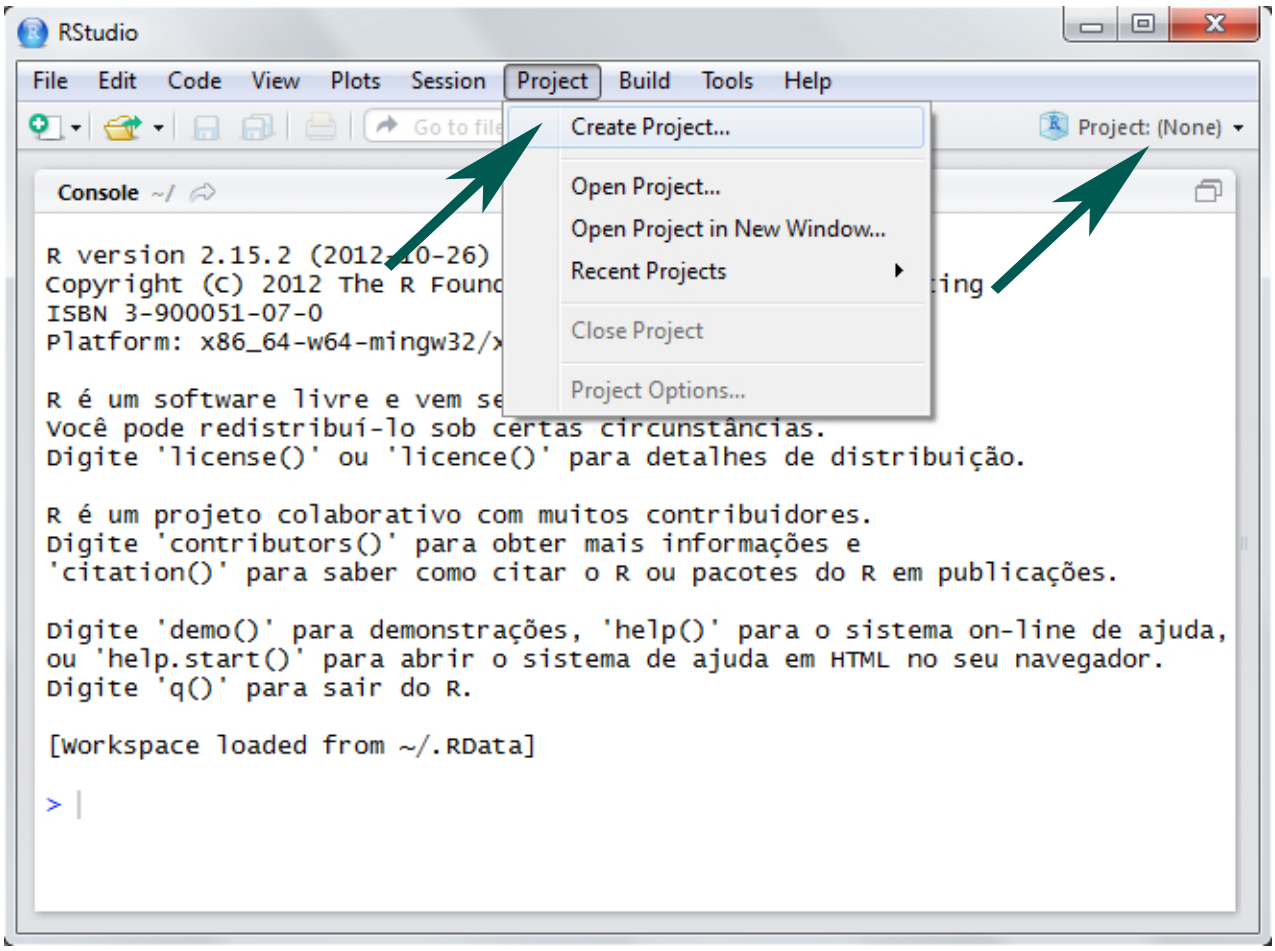

Figura 3: Construção do pacote mypkg pelo RStudio.

Na Figura 4, temos as opções:

- **Type**: escolheremos a opção "package";
- **Package name**: escolheremos o nome do pacote, que no nosso caso, "mypkg";
- **Create package based on source files**: nesse local, adicionaremos (botão Add) o arquivo "opalgb.r";
- **Create project as subdirectory of**: local aonde queremos a criação do pacote, que no nosso caso, "c:/package";
- por fim, clicar em "Create Project".

5.1 Editando os arquivos do pacote

Para editarmos observe que no RStudio na opção Files, todos os arquivos da **seção 3** foram criados, sendo que há um acréscimo dos arquivos: "mykpg.Rproj" e ".Rbuildignore", que quando

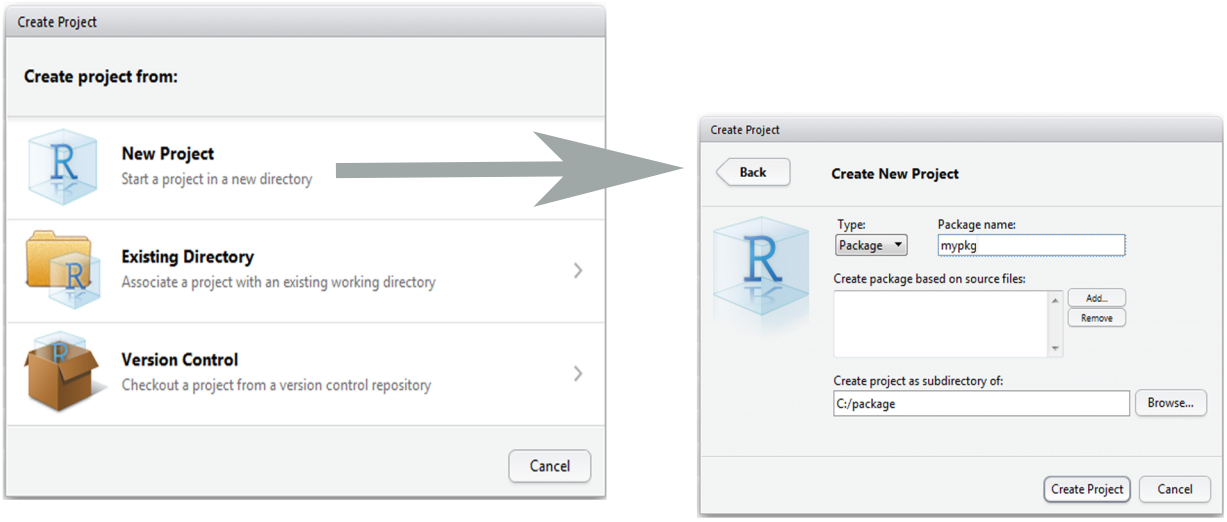

Figura 4: Construção do pacote mypkg pelo RStudio.

o pacote for compilado, estes devem ser deletados. Para editarmos os arquivos: **DESCRIPTION**, **NAMESPACE** e a pasta **man**, basta um clique no arquivo desejado, que o RStudio permitirá sua edição, como observado na Figura (6).

6 Checando o pacote

Depois de todos os arquivos editados, usaremos o seguinte comando no CMD:

```
c:\Users\Benallanna> cd c:\package
c:\package>Rcmd check mypkg
```

para checar se todos os arquivos estão no padrão do R. Ao executar o comando, será criado uma pasta, chamada "< nome do pacote >.Rcheck", que no nosso caso, "mypkg.Rcheck", em que dentro do diretório, o arquivo "00check.log"(pode ser aberto pelo bloco de notas) irá mostrar todos os detalhes da checagem do pacote, inclusive mostrando possíveis erros, se houver. Caso haja, tem que averiguar quais os problemas.

Uma checagem mais profunda, já específico para enviar o pacote para o CRAN do R, irá verificar informações diretamente do CRAN (por exemplo: saber se já não existe um pacote com o nome "mypkg"), por isso, é interessante que esteja conectado à internet. Assim, o comando no CMD:

```
c:\Users\Benallanna> cd c:\package
c:\package>Rcmd check --as-cran mypkg
```

Nesse ponto da construção do pacote, estamos prontos para compilar o pacote.

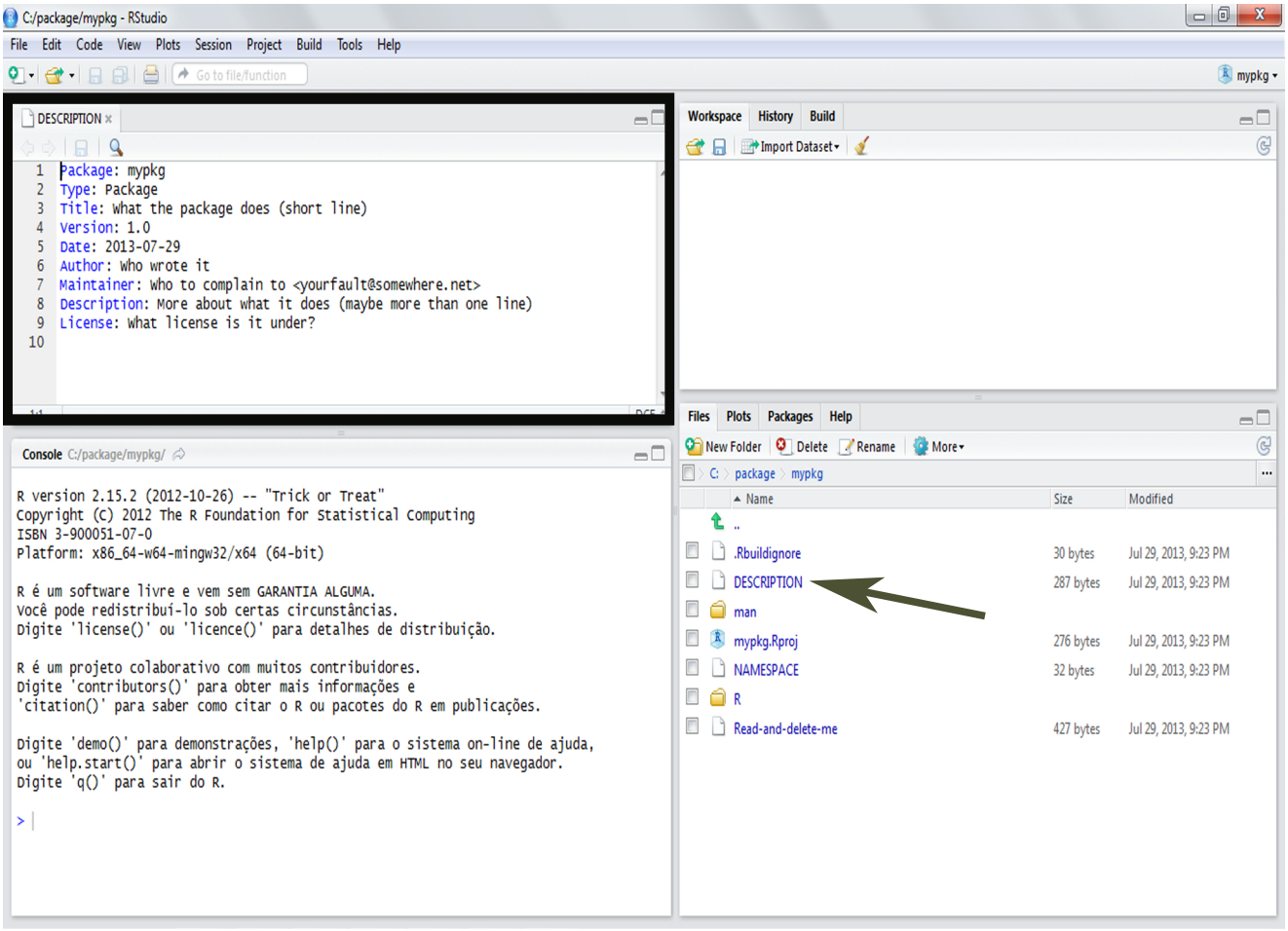

Figura 5: Construção do pacote mypkg pelo RStudio.

7 Compilando o pacote para o Windows

Para compilar e instalar o pacote no Windows, utilizaremos o seguinte comando no CMD:

```
Rcmd INSTALL --build mypkg
```

OBS.: Antes do build, se for usado apenas um "-", haverá apenas a instalação do pacote. Usando "- -", dois traços, haverá a instalação e a exportação do arquivo ".zip".

Esse comando gerará no diretório "c://package", um arquivo com a extensão ".zip", chamado "mypkg.zip".

Para compilar nos demais sistemas operacionais, utilizaremos no CMD, o seguinte comando:

```
Rcmd build mypkg
```

Esse comando gerará no diretório "c://package", um arquivo com a extensão ".tar.gz", chamado "mypkg.tar.gz".

8 Submissão para o CRAN

Pelo Windows, entraremos pelo Internet Explorer no site:

⟨ftp://cran.R-project.org/incoming/⟩

Posteriormente, **Internet Explorer > Exibir > Abrir Site FTP no Windows Explorer**. Abrirar uma pasta chamada "incoming", em que você deverá copiar e colar o arquivo nessa pasta. Assim, enviando um email para o cran (⟨cran@r-project.org⟩), é só aguardar a publicação.

Podemos também, abrindo o **menu iniciar do Windows > Windows Explorer >** ⟨**ftp://cran.R-project.org/incoming/**⟩.

Outra forma de enviou do pacote ao CRAN será seguindo os passos pelo site:

⟨http://cran.r-project.org/submit.html⟩

9 Resgatando pacotes desativados no R

Muitos pacotes no R são desativados devido a atualização que os autores não fazem no pacote. Na busca desses pacotes, podemos usar o link ⟨http://cran.r-project.org/src/contrib/Archive⟩, ou o próprio Google. Posteriormente, encontrado o pacote desejado com a extensão "<arquivo.tar.gz>", devendo extraí-lo. Como exemplo, vamos buscar o pacote "zmatrix_1.1.tar.gz" em que sua última versão foi feita em 14-Maio-2001. Vamos dizer que precisamos desse pacote. Assim, iremos no nosso diretório "c:\package" colocar o arquivo "zmatrix_1.1.tar.gz" e extraí-lo. Após isso, faz-se o seguinte comando:

```
Rcmd INSTALL --build zmatrix
```

O que pode acontecer com outros pacotes, é que estes por serem muito antigos, poderão compilar apenas em versões anteriores ao seu software R, ou problemas de atualização na estrutura do pacote, já que o R está em constante atualização. Caso isso aconteça, basta editar os arquivos da pasta "zmatrix" extraída de "zmatrix_1.1.tar.gz".

10 Buscando rotinas da base do R

Muitas vezes gostaríamos de saber como é o algoritmo de algumas rotinas da base do R, pois a partir destes gostaríamos de implementar algo similar. Por exemplo, o comando "qtukey( )" retorna o quantil da distribuição da amplitude estudentizada externamente da normal, e gostaria de averiguar o algoritmo dessa rotina. Uma forma usual seria inserir no R, apenas o nome:

```
> qtukey
function (p, nmeans, df, nranges = 1, lower.tail = TRUE, log.p = FALSE)
.Internal(qtukey(p, nranges, nmeans, df, lower.tail, log.p))
<bytecode: 0x000000000772d900>
<environment: namespace:stats>
```

Porém, observamos que a saída não é o algoritmo, pois essa rotina está implementada na base do R, geralmente em código Fortran/C/C++. Assim, para encontrarmos, basta acessar: ⟨http://fossies.org/dox/R-3.0.1/files.html⟩ como observado na Figura 6.

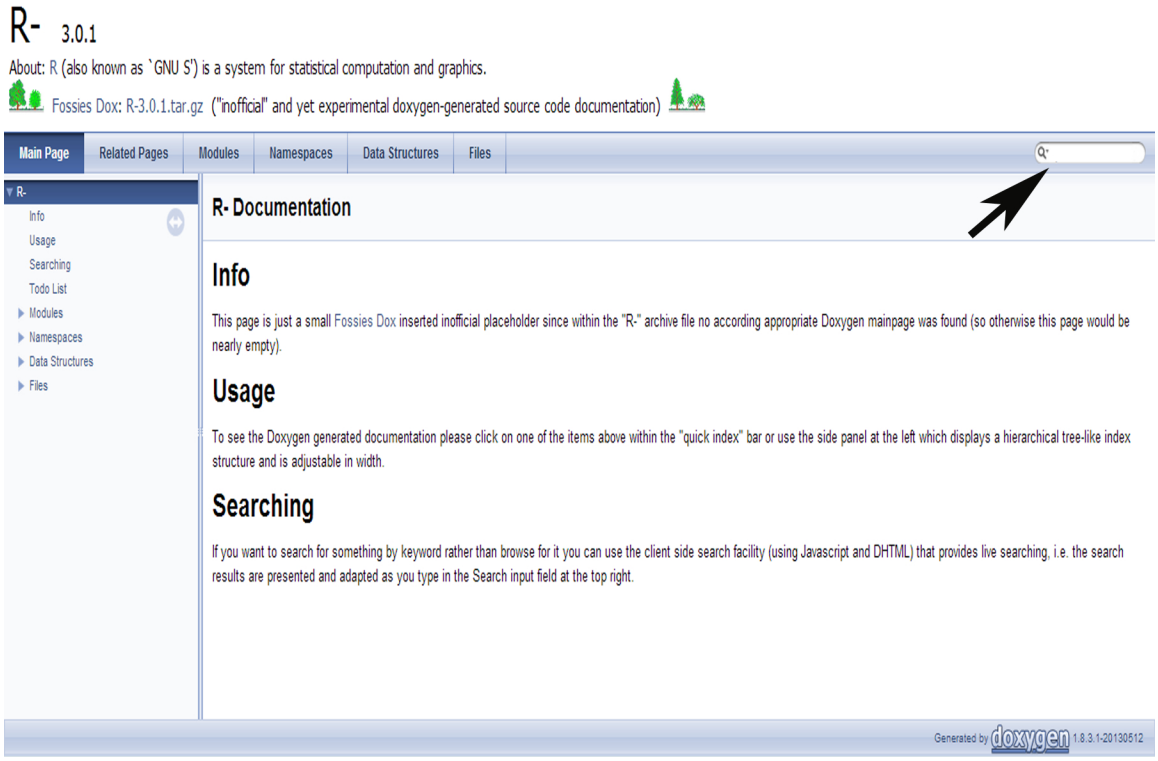

Figura 6: Rotinas da base do R.

Na Busca, escreva a rotina desejada, por exemplo: qtukey, e verificar todo o algoritmo.

11 Exportando funções desejadas

No R, digitando apenas o nome da função, após o carregamento do pacote, a rotina pode ser visualizada. Por exemplo:

```
> library(mypkg)
> #exemplo com a funcao ''soma'':
> soma
function (a, b)
{
    calculo <- a + b
    return(calculo)
}
<environment: namespace:mypkg>
```

Porém, poderemos "esconder" a exportação dessa função. Basta configurar o arquivo **NAMESPACE** no pacote. Vamos criar uma rotina, "auxi.r", na pasta R do pacote, com as seguintes funções, para evitar a exportação das funções originais do pacote "mypkg":

```
#rotina auxiliar

razao <-
  function (a,b){
    x <-divisi(a,b)
    return(x)}

prodi <-
  function (a,b){
  x <-multi(a,b)
  return(x)}

adi <-
  function (a,b){
    x<-soma(a,b)
    return(x)}

dif <-
  function (a,b){
    x <- subtra(a,b)
    return(x)}
```

No arquivo **NAMESPACE**, apagaremos a rotina anterior, e reescrevemos as seguintes linhas de comando:

```
export(razao,prodi,adi,dif)
```

Minhas funções passarão a ser chamadas serão agora: **adi**, **dif**, **razao** e **prodi**, representando, soma, subtração, razão e produto, respectivamente. Assim, ao pedir a função soma, com essa nova configuração:

```
> library("mypkg", lib.loc="C:/Program Files/R/R-2.15.2/library")
> adi
function (a, b)
{
    x <- soma(a, b)
    return(x)
}
<environment: namespace:mypkg>
```

o R não mostrará mais a função primitiva da soma. Como o nome das funções exportadas mudaram, temos que também modificar os ".Rd" para os nomes das funções alteradas.