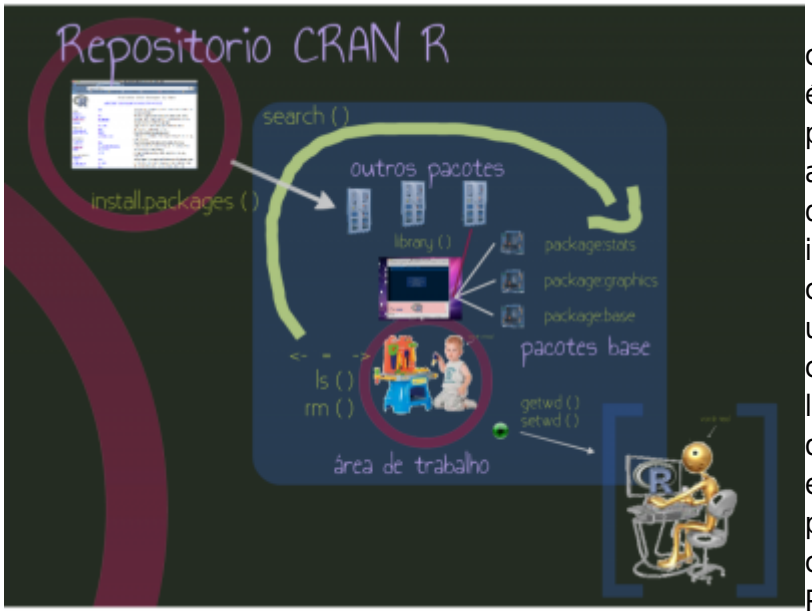
- Tutorial
- Exercícios
- Apostila

1b. Introdução ao R: ambiente de programação

O Ambiente de Programação



Agora que apresentamos as bases conceituais e da sintaxe da linguagem R é o momento de entender o ambiente de programação a que a linguagem está associada. O R não é só o interpretador da linguagem. Existe toda a gestão da informação que transita nessa nossa conversa com o R. Para ser um bom usuário do R é importante que entenda como essa interação se dá e é preciso se localizar nesse ambiente que é nossa oficina virtual de criação. Até aqui já entendemos que nesse ambiente de programação há dois espaços distintos: o código ou script e o interpretador do R! Pois bem, esses são os limites do nosso ambiente, nossa ligação com o R que é o script e a conexão com a linguagem de máquina que é o interpretador!

Entre esses extremos existem espaços virtuais [1)] que organizam o fluxo da informação e seu armazenamento. Por exemplo, criamos objetos e acessamos aquilo que foi atribuído a ele e também manipulamos esse objeto com outros objetos da classe função. Mas onde o objeto criado fica armazenado? De onde vem a função que utilizamos? Como faço para acessar meu script de uma sessão do R?

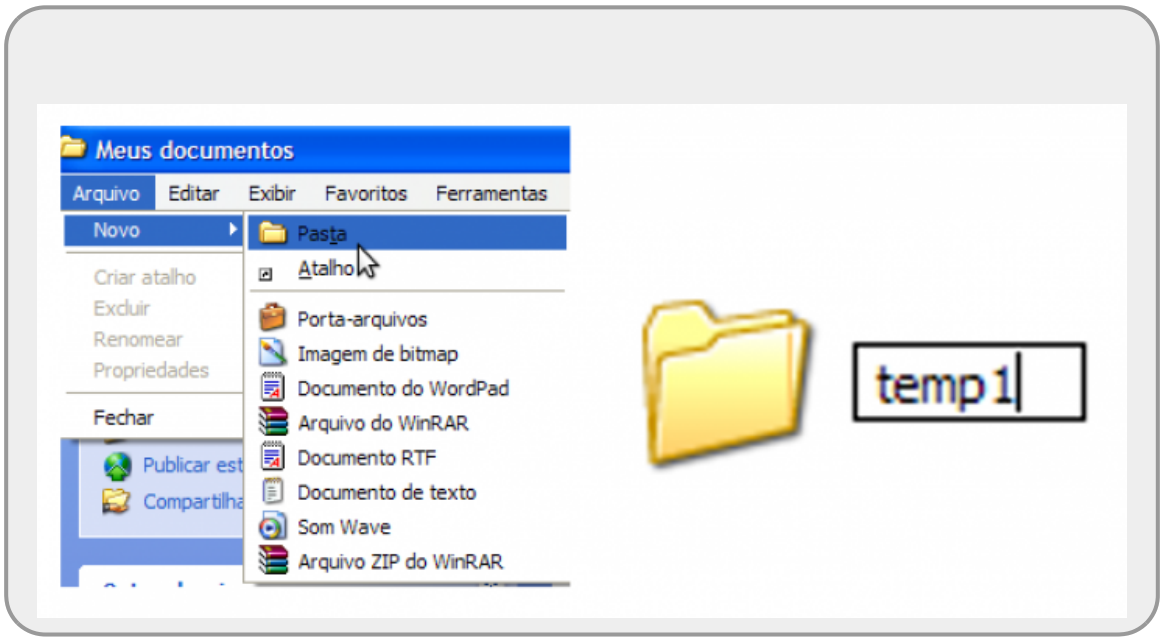
O Mapa do R

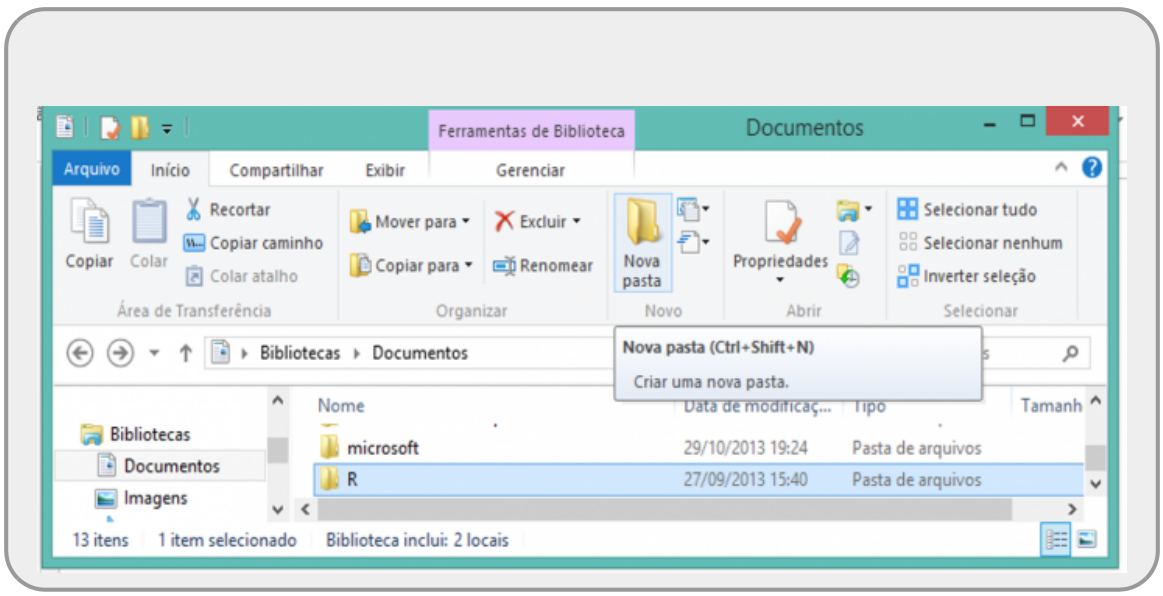
Nosso exemplo de como se organizar no R aqui, será com o ruWindows. Não porque gostamos do sistema operacional, mas simplesmente porque é o sistema mais utilizado. A lógica é a mesma para outros sistemas operacionais, com macOS e Linux, com pequenas variações.

Organize suas pastas

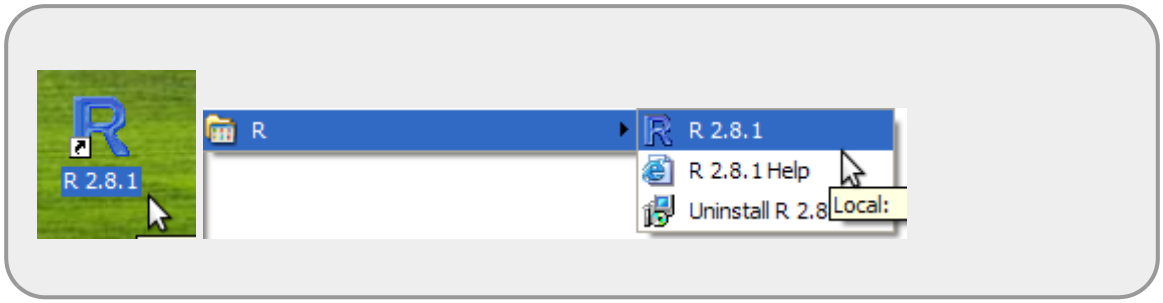
Antes de começar um novo projeto, uma nova análise de dados por exemplo, crie uma pasta[2)] de

trabalho específico para o projeto, com o menu ***Arquivo>Novo>Pasta*** do ruWindows explorer. Por exemplo, para esse curso você pode criar uma pasta `cursoR` e dentro dela uma outra como `aula01Intro` e dentro dessa pasta, colocar o script que fizemos na aula anterior `tutorial01.r` .





Após criar as pastas, execute o R a partir do atalho do ruWindows ou pelo menu de programas no computador ou na barra de ferramentas.



Diretório de Trabalho

A primeira coisa que deve fazer ao iniciar uma sessão do R é verificar qual é o diretório[3] de trabalho que o R está vinculado. Essa pasta é a conexão da sua sessão do R com a memória de disco do seu computador. Podemos acessar o endereço da pasta no computador, com o comando:

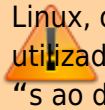
```
getwd()
```

Ao iniciar o R pelo atalho da área de trabalho do ruWindows o diretório de trabalho será sempre o mesmo, possivelmente em “Meus Documentos”, e.g.:

```
[1] "C:/Documents and Settings/Administrador/Meus documentos"
```

Para mudar o diretório usamos a função `setwd`, de *set work directory*:

```
setwd("C:/Documents and Settings/Administrador/Meus
documentos/cursoR/aulaIntro")
```



IMPORTANTE: as barras de endereço devem ser no padrão Linux, ou seja, barras simples e não dupla invertida como a utilizada em ruWindows. Um erro comum no R é esquecer as “s ao digitar o endereço do diretório de trabalho, veja Os 10 Mandamentos do R.

Verifique se mudança funcionou, com um novo comando `getwd`:

```
getwd()
```

Caso esteja no diretório de trabalho desejado é possível listar todos os arquivos da pasta utilizando uma das funções abaixo:

```
dir()
list.files()
```

Caso não tenha colocado o script da aula anterior nessa pasta, coloque e rode novamente o código acima para verificar se está tudo certo.

Rodando o script

Vimos no tutorial anterior como rodar o script a partir do código aberto no GUI do R. Podemos também rodar o script inteiro, a partir de um arquivo, utilizando a função `source` que apresenta um argumento chamado `file` [4].

```
source(file = "tutorial01.r")
```

Workspace do R

Todo objeto criado em uma sessão do R fica armazenado, durante a sessão, na área de trabalho [5)]. Para não confundir a área de trabalho do R com a área de trabalho do ruWindows, vamos aqui utilizar o termo **workspace** para designar a área de trabalho do R. Na nossa metáfora da oficina, podemos dizer que a *área de trabalho do R* é a nossa bancada, onde temos os objetos que estamos trabalhando.

Podemos listar os objetos do *workspace* do R com a função `ls` ou `objects`. Antes de usar veja a documentação da função. A documentação do `help` no R é padronizada e na descrição é sempre fornecida, entre outras, a informação sobre o que é retornado e qual o formato desse output. Vamos encontrar essa informação na documentação:

```
help(ls)
```

A documentação reporta que "`ls` and `objects` return a vector of character strings...". Como acabamos de carregar o script da aula passada nessa sessão do R, o workspace deve conter, entre outros, os objetos que criamos, como `dez`, `cem`, `contadez`, etc... Vamos verificar:

```
ls()
```

Caso tenha muitos objetos na sua área de trabalho, uma forma de selecionar os objetos é usar o argumento `pattern`:

```
ls(pattern="co")
```

O código acima pede para listar os objetos do workspace que tenham `co` em algum lugar do nome, o resultado deve ser:

```
[1] "contadez"    "copa70"      "copa94"      "descontadez"
```

Um conceito básico do R é que o resultado retornado por uma função pode ser atribuído a um objeto:

```
objetosCo <- ls(pattern = "co")
objetosCo
class(objetosCo)
```

Note que o `objetosCo` é um vetor da classe `characters` e contém os nomes dos objetos que apresentam `co` no nome.

Os objetos que estão no workspace podem ser salvos no seu diretório de trabalho utilizando a função `save`. A função `save.image` é um atalho da função `save` para salvar todos os objetos do workspace.

O arquivo salvo, em geral, tem a extensão `.RData`[6]. Vamos salvar os objetos que tem o padrão `co` no nome no nosso diretório de trabalho e em seguida certifique-se de que o arquivo RData foi criado:

```
save(list=ls(pattern="co"), file="coObjetos.RData")
list.files()
```

Agora, vamos salvar todas os objetos do **workspace** e verificar se foi criado o arquivo!

```
save.image()
list.files(all.files=TRUE, pattern=".RData")
```

Como não fornecemos nenhum nome de arquivo [7] para a função `save.image`, ela por padrão cria um arquivo com o nome `.RData`. Arquivos começando por `.` em geral, são arquivos que ficam oculto no sistema operacional, por isso usamos o argumento `all.files = TRUE` na função `list.files`. Dessa forma podemos ver os arquivos ocultos no diretório de trabalho.

Vamos criar agora um objeto que contenha todos os nomes dos objetos dessa sessão e em seguida vamos fechar a sessão do R, tomando o cuidado de respoder **sim** à pergunta se quer salvar a imagem da sua sessão!.

```
allobj <- ls()
allobj
```

Arquivo .RData

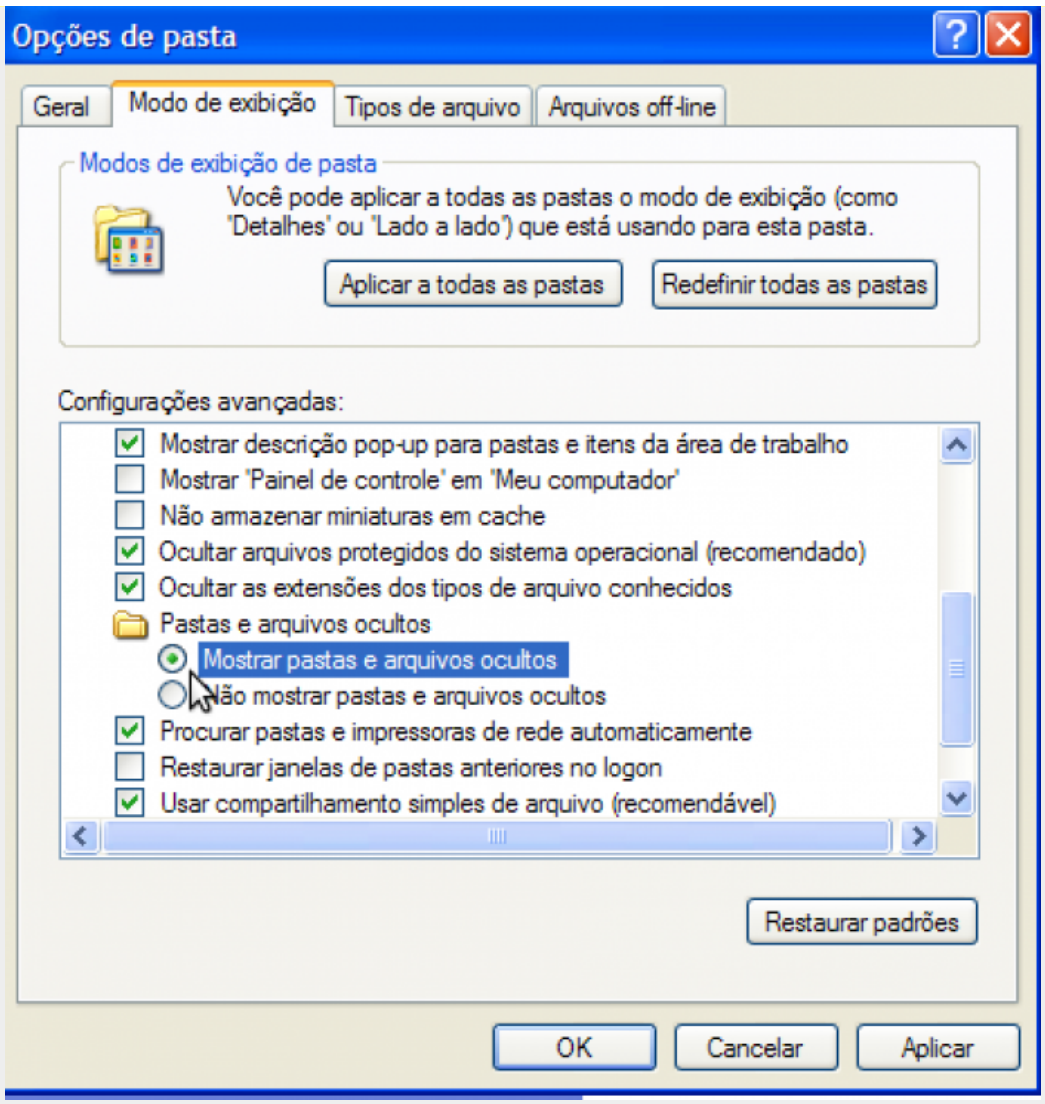
Agora com a sessão do R desligada, abra o gerenciador de arquivos, Windows Explorer, na pasta criada no inicio desse tutorial, nossa sugestão foi algo como:

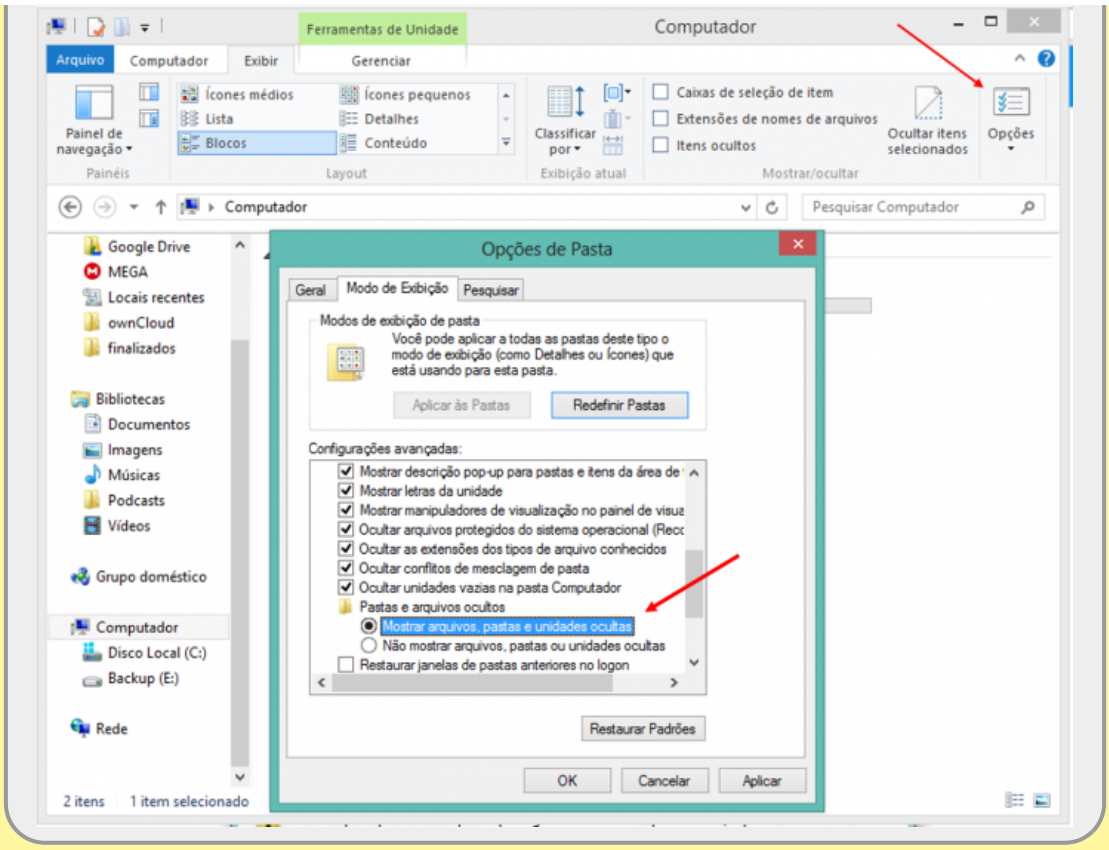
```
"C:/Documents and Settings/Administrador/Meus documentos/cursoR/aulaIntro"
```

Deve existir um arquivo com o nome `.RData` que listamos na nossa sessão do R. Entretanto, como se trata de arquivo oculto, normalmente o sistema operacional não mostra!

Mostrando Arquivos Ocultos

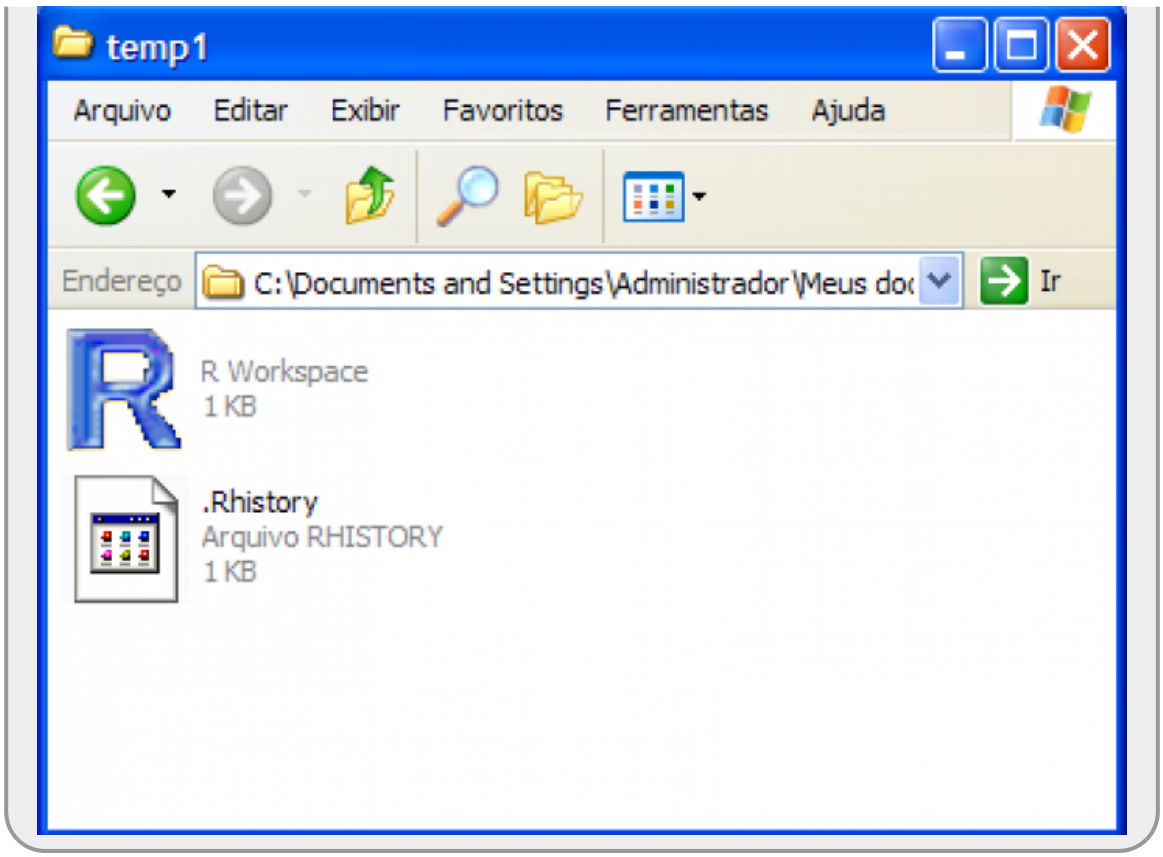
Veja como configurar essa opção no ruWindows:

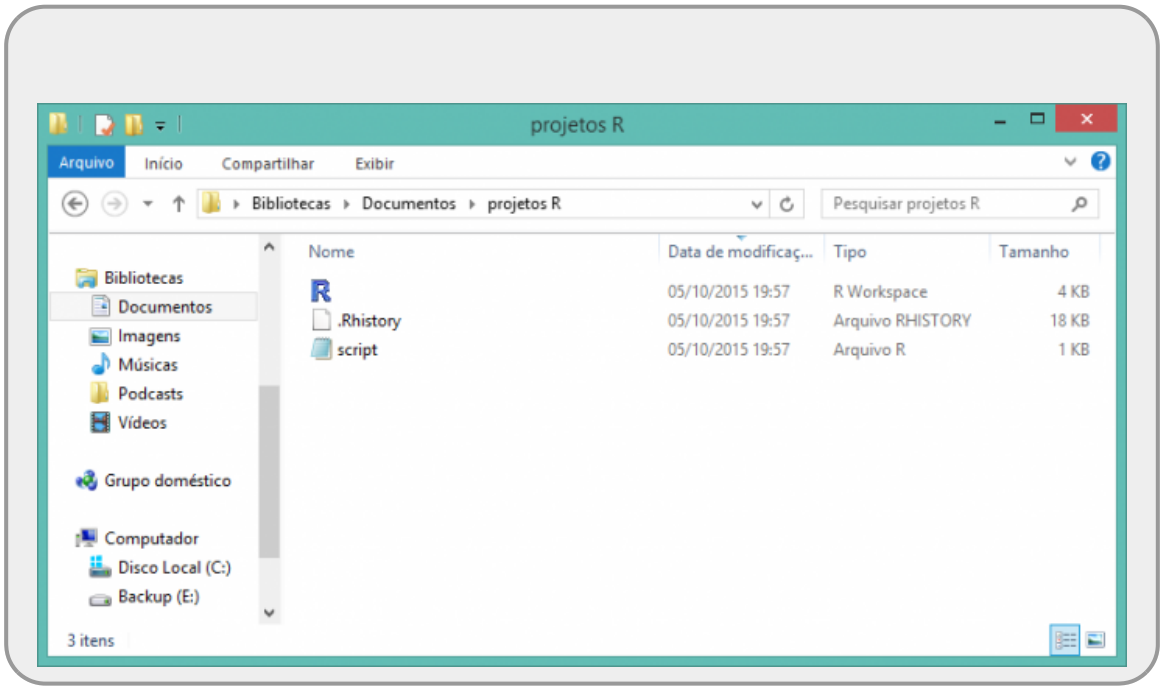

Opções de pasta
Geral
Modo de exibição
Tipos de arquivo
Arquivos off-line
Modos de exibição de pasta
Você pode aplicar a todas as pastas o modo de exibição (como 'Detalhes' ou 'Lado a lado') que está usando para esta pasta.
Aplicar a todas as pastas
Redefinir todas as pastas
Configurações avançadas:
Mostrar descrição pop-up para pastas e itens da área de trabalho
Mostrar 'Painel de controle' em 'Meu computador'
Não armazenar miniaturas em cache
Ocultar arquivos protegidos do sistema operacional (recomendado)
Ocultar as extensões dos tipos de arquivo conhecidos
Pastas e arquivos ocultos
Mostrar pastas e arquivos ocultos
Não mostrar pastas e arquivos ocultos
Procurar pastas e impressoras de rede automaticamente
Restaurar janelas de pastas anteriores no logon
Usar compartilhamento simples de arquivo (recomendável)
Restaurar padrões
OK
Cancelar
Aplicar



.RData: iniciando uma sessão

Caso o gerenciador de arquivos já esteja configurado para mostrar arquivos ocultos, você verá o arquivo `.RData` na pasta.





Clique nesse arquivo `.RData`, descrito como `R Workspace`. Uma sessão do R deve-se abrir automaticamente.

Verifique quais objetos estão no *workspace* da sessão aberta e qual o diretório de trabalho que está conectado a essa sessão:

```
ls()
getwd()
ls(pattern = "obj")
```

Acabamos de ensinar como abrir uma sessão do R com todos os objetos criados na última sessão aberta do R e apontando para o diretório de trabalho desejado. Isso é uma forma eficiente de reiniciar uma sessão do R que foi interrompida!

Note que o `allobj` está entre os objetos que foram carregados pelo `.RData`. Ele foi criado depois que utilizamos a função `save.image`. Isso acontece porque ao fechar a sessão concordamos em salvá-la e por padrão o R executa `save.image()` antes de fechar. O arquivo `.RData` salvo anteriormente foi sobrescrito pelo novo `.RData` que contem o `allobj`.

.RData: um problema!

Quando iniciamos uma sessão do R pelo menu ou por um atalho da área de trabalho do Windows, ele automaticamente busca um arquivo `.RData` no diretório de trabalho a que está vinculado e carrega automaticamente esse `.RData`. Caso o usuário tenha o hábito de abrir o R sempre do mesmo atalho e sempre salvar ao fechar a sessão, o arquivo `.RData` rapidamente se torna um monstro devorador de memória, com todos os objetos criados em todas as sessões do R que foram trabalhadas. Como o R armazena o **workspace** da sessão na memória RAM do computador, logo seu computador irá parecer uma carroça velha, mesmo se for um Mac de 50 mil reais!

Uma nova sessão

Quando iniciamos um novo projeto é desejável que a nossa *bancada de trabalho* ou ***workspace*** do R esteja limpo. Não é aconselhável trabalhar em uma oficina bagunçada e cheia de material que não será utilizado na tarefa a ser executada, não é muito eficiente.

Para remover objetos do workspace usamos o comando `rm` que é um atalho para a função `remove`. Primeiro, como sempre devemos fazer, vamos consultar a documentação:

```
help(rm)
```

Removendo todos os Objetos

A documentação do `rm` nos diz que o argumento `list` deve receber uma vetor de caracteres com o nome dos objetos que devem ser removidos. Como queremos remover todos os objetos, podemos usar a sintaxe de aninhamento de funções do R. Veja como ficaria um comando para listar todos os objetos do workspace do R e ao mesmo tempo removê-los

```
rm(list = ls())
```

Note que estamos fornecendo ao argumento `list` da função `rm` o que a função `ls()` nos retorna: a lista do nome de todos os objetos do workspace, na forma de um vetor de caracteres.

Verifique novamente o workspace para garantir que esteja vazio e salve o workspace!

```
ls()
save.image()
```

Agora vamos simular a perda dos objetos: saia do R, respondendo "NÃO" à pergunta "Salvar Área de Trabalho"[8].

Abra o R de novo a partir do `.RData`. Tudo perdido? Não! Com o código salvo (script) você pode executá-lo novamente, e recuperar todo o trabalho. Repita o procedimento novamente de abrir o arquivo de script e rodá-lo 😉.

```
source(file = "tutorial01.r")
ls()
```

Iniciando um novo projeto

- *Garanta que a sessão do R esteja conectada ao diretório de trabalho, que é a pasta onde deve estar os dados que será usado e onde iremos salvar os resultados e gráficos realizados*
- *Verifique se o workspace está limpo, e caso não esteja, remova todos os objetos que não serão utilizados no projeto. Preferencialmente ele deve estar limpo.*
- *Não se preocupe com o `.RData`, concentre-se no script, salve o arquivo com frequência e deixe ele bem documentado. Caso tenha problema na sessão do R, precisa apenas rodar o script novamente e continuar o trabalho pelo script.*

Onde estão as Funções?

O nosso **workspace** do R armazena todos os nossos objetos em uma sessão. Entretanto, estamos usando objetos de funções o tempo inteiro. Onde estão as funções? Para responder essa pergunta

vamos executar a função abaixo:

```
search()
```

O resultado deve ser algo como:

```
 [1] ".GlobalEnv"        "package:stats"     "package:graphics"
 [4] "package:grDevices" "package:utils"     "package:datasets"
 [7] "package:methods"   "package:base"
```

O `search()` retorna os pacotes que estão carregados na sua sessão do R. Pacotes são conjuntos de ferramentas associadas a algum conjunto de tarefas. Na nossa metáfora da oficina os pacotes são os armários onde guardamos nossas ferramentas classificadas pelo tipo de uso. Por padrão o R abre alguns desses armários quando iniciamos uma sessão, os armários com as ferramentas básicas: eles que contem as funções que estamos usando.

Além desses pacotes, existem outros que são distribuídos junto ao programa R, mas que não são abertos no início de uma sessão. São pacotes importantes, mas um pouco mais específicos ou especializados do que as ferramentas básicas. São como armários que estão trancados, com ferramentas que usamos menos frequentemente, precisamos destrancar e abrir para poder usar as ferramentas que estão neles. Para acessar esses pacotes utilizamos a função `library` que nos mostra os pacotes que estão instalados no nosso computador:

```
 library()
```

Por exemplo, temos o pacote `lattice` descrito como: “A powerful and elegant high-level data visualization…”. Uma das funções desse pacote é `bwplot` para a construção de boxplots. Se tentarmos consultar a documentação dessa função:

```
help(bwplot)
```

A resposta é que não há documentação associada.

Carregando Pacotes

Para utilizar as funções de um pacote, não basta o pacote estar instalado no computador, é necessário carregá-lo, ou seja abrir o armário contém as ferramentas! Para isso usamos a própria função `library` :

```
library("lattice")
search()
```

Agora o pacote `lattice` aparece na segunda posição do nosso `search`. O que isso significa? Cada uma das posições que o search apresenta, representam o caminho de busca do R na memória. São como compartimentos [9] de memória. Quando executamos um comando o R busca os objetos na memória seguindo trajeto apresentado pelo `search`. O primeiro compartimento que é procurado se chama `.GlobalEnv`. Esse é o compartimento de memória do nosso **workspace** e o pacote que foi carregado recentemente é colocado na posição seguinte. O último compartimento é dedicado ao pacote `base` onde estão os objetos e funções básicas do R.

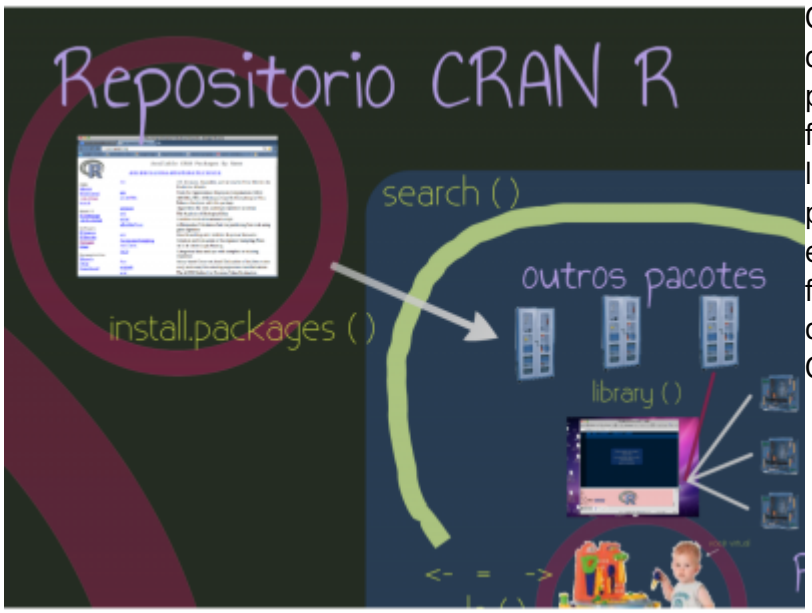
Agora a função `bwplot` está disponível para ser utilizada. Como ainda não ensinamos a fazer gráficos, é possível pedir auxílio para a própria função mostrar o que ela é capaz de fazer. Na documentação da função há uma sessão dedicada a exemplificar o uso da função. É possível copiar e colar esse código para executar os exemplos, ou usar a função `example` que extrai e roda os exemplos:

```
help(bwplot)
example(bwplot)
```

Quais funções estão disponíveis em um pacote instalado? Qual documentação está disponível junto ao pacote? Essas informações podem ser encontradas em um outro tipo de ajuda de documentação do R, o `help.start`. Que abre toda a documentação geral do R que foi instalada no computador. Apesar de ser um hipertexto, ela está localmente instalada e não precisa de internet para ser aberta. Para encontrar a documentação específica de um pacote instalado entre no link `Packages`. Navegue nessa documentação para se habituar a ela.

```
help.start()
```

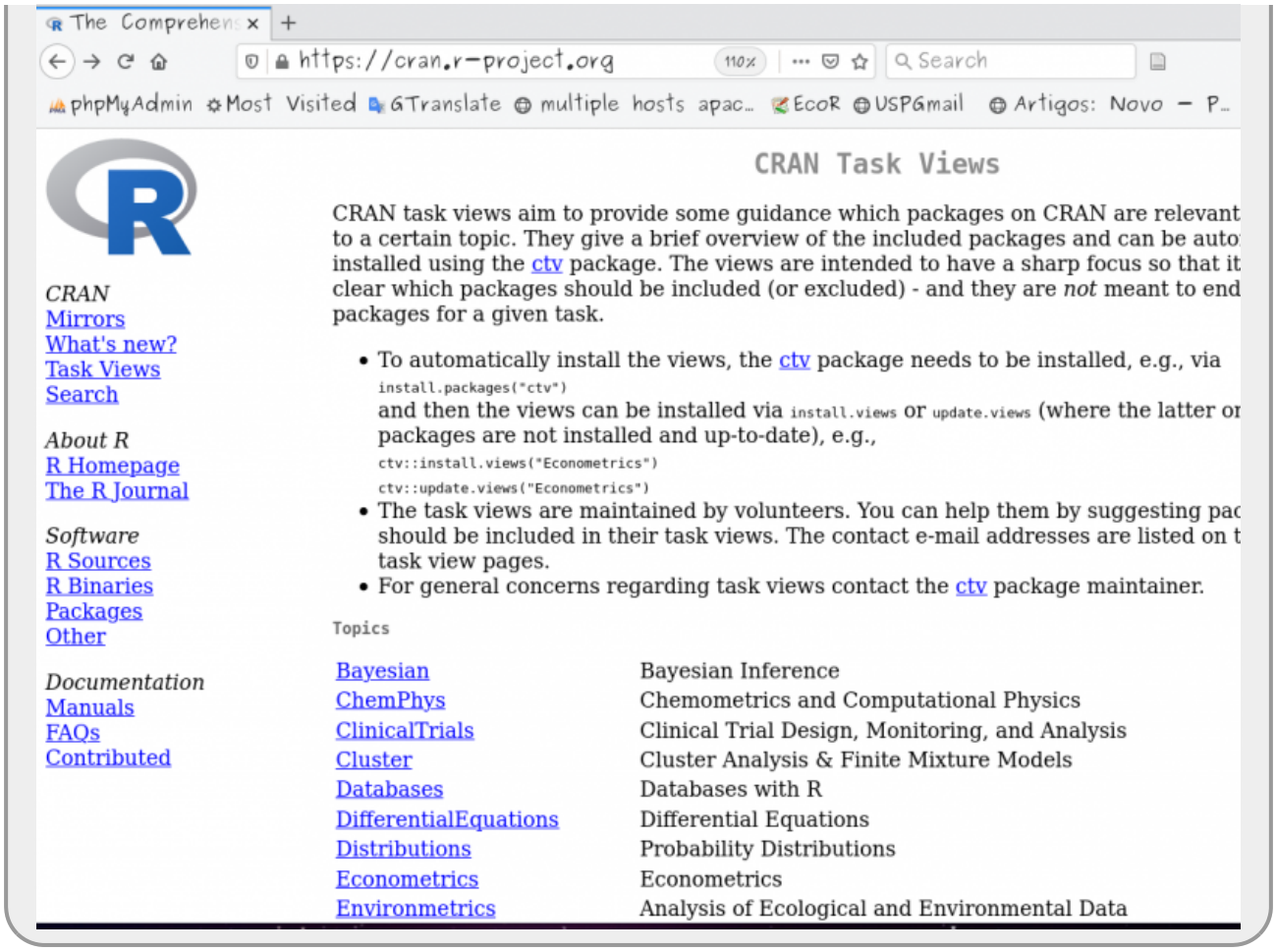
Instalando Pacotes



Quando estamos trabalhando em uma oficina normalmente chegamos a um ponto em que precisamos de uma ferramenta que não está disponível localmente. Nesses momentos precisamos nos deslocar até uma loja especializada para adquirir as ferramentas necessárias. As loja do R são os repositórios de pacotes, oficial é o CRAN https://cran.r-project.org/.

Entre os três principais repositórios [10], em consulta feita em 31 de julho de 2020, existiam 20.205 pacotes contendo mais de **3,1 milhões de funções** disponíveis para o R[11]!

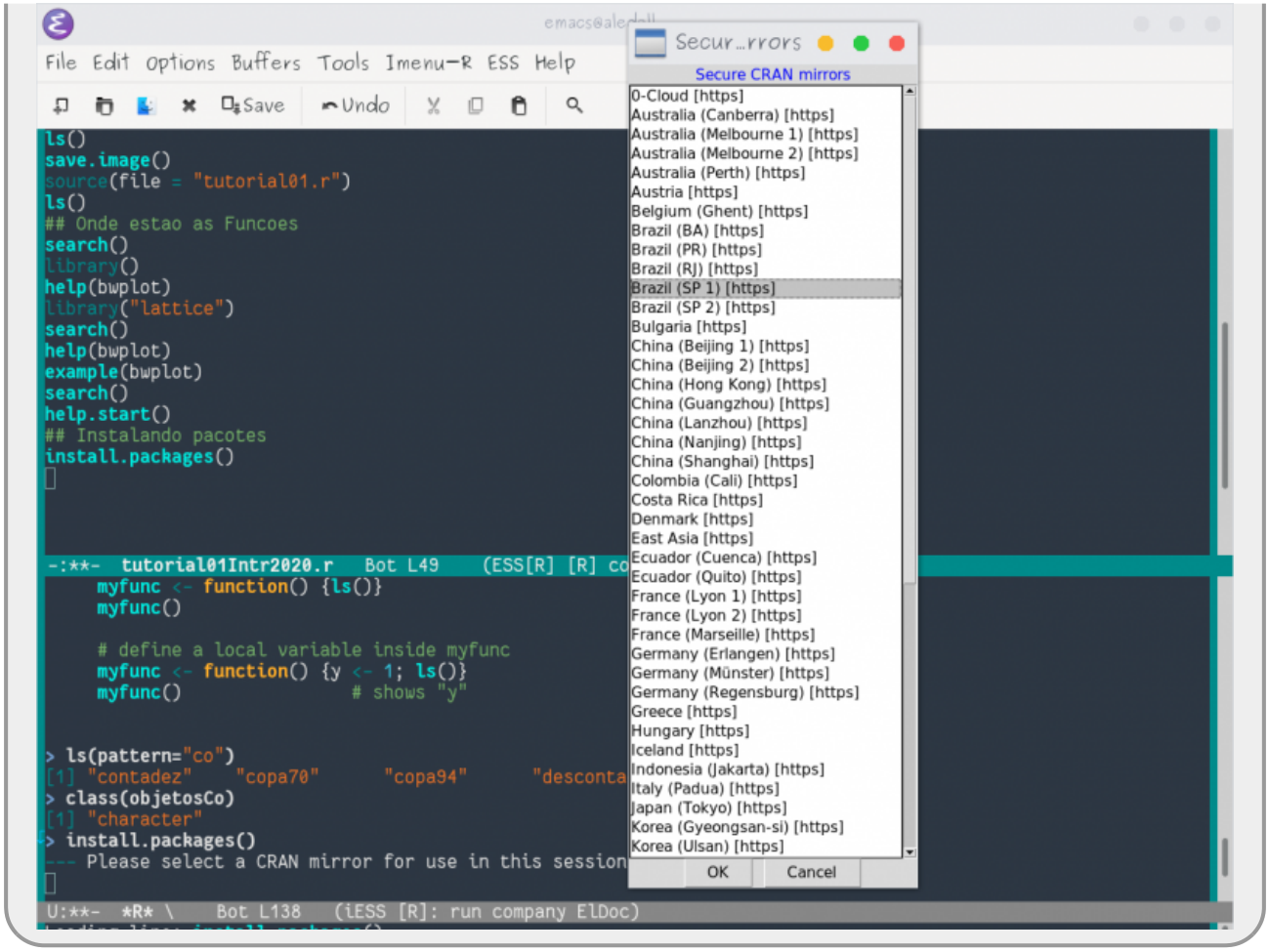
Nesse universo de itens disponíveis fica difícil encontrar a melhor ferramenta para a tarefa que queremos executar. Existem várias ferramentas para auxiliar nessa tarefa. Aconselhamos fortemente a usar e inciar a sua busca pelo `Task Views` disponível no site oficial do CRAN. Que são revisões de pacotes disponíveis separadas por temas, normalmente áreas de pesquisa (ecologia, genética, economia...) ou tipos de análises (bayesiana, multivariada, espacial...).



Entre no site do CRAN e navegue para **Task Views** > **Environmetrics**. Nele irá encontrar uma revisão sobre as ferramentas disponíveis para análises de dados na área de ecologia, dividida em tópicos. É muito útil, inclusive para saber quanto confiável e estável é um pacote. Na documentação está a descrição de um dos pacotes mais importantes para análise de dados multivariados em ecologia de comunidades chamado `vegan`. Um pacote com cerca de 20 anos, atualizado frequentemente com incorporação de novas metodologia e que já foi muito testado, um pacote clássico para a ecologia. A forma mais prática para instalar um pacote do CRAN é utilizar a função interativa `install.packages()`. Essa função irá abrir uma janela para que o usuário escolha qual o espelho do CRAN de onde o pacote virá e em seguida lista todos os pacotes que estão no repositório. Como são muitos é aconselhável usar o argumento `pkgs` que recebe um vetor de caracteres com os nomes dos pacotes a serem instalados [12)]

```
install.packages(pkgs = "vegan")
```

Selecione o repositório mais próximo fisicamente, no nosso caso, é o `Brazil(SP 1)[https]`:



Agora vamos verificar se o pacote está instalado consultando o `help.start()` e virificando se aparece na na lista de pacotes instalados. Vá no link abaixo do `Descrition File` denominado `User guides, package vignettes and other documentation` e abra o documento com o tema `Diversity analysis in vegan`.

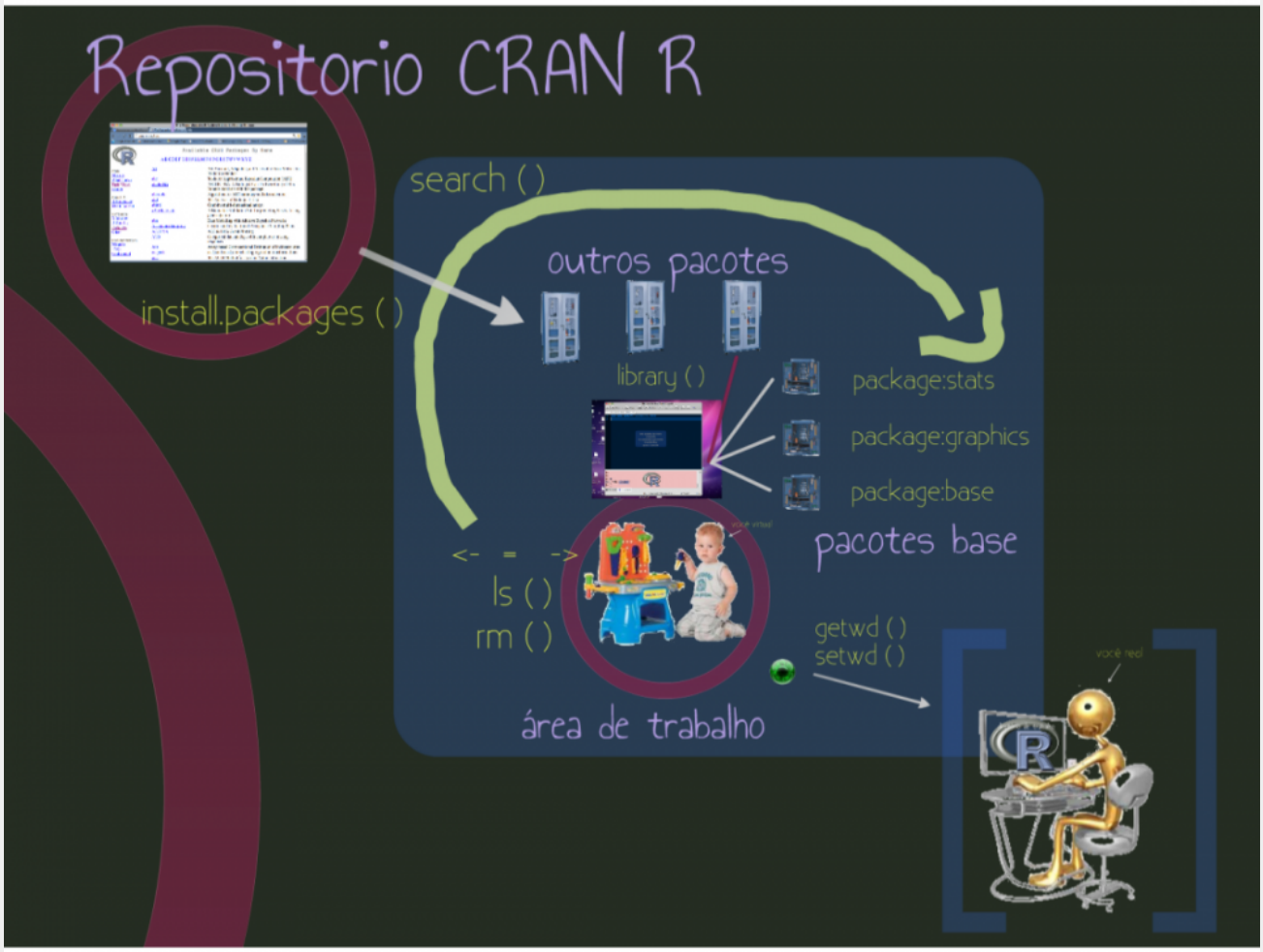
```
help.start()
```

Vá no link `Packages` e selecione o pacote `vegan` na lista de pacotes. A página inicial do pacote com o título **Community Ecology Package** conterá o link para a documentação de cada uma das funções do pacote. São muitas, mais de 500! Felizmente a documentação do pacote é muito completa e tem alguns tutoriais incluídos que são chamados de `Vignettes`. Essas vinhetas são de grande ajuda para quem precisa estudar um pacote. Abra a vinheta sobre `Diversity analysis in vegan`, navegue no documento. São 12 páginas condensadas sobre análises de diversidade, mostrando como faze-las no `vegan` e com exemplos muito ilustrativos.

Esquema do Mapa do R

Esse tutorial é uma base muito importante para que o usuário se torne fluente na linguagem e se posicione no ambiente de programação. Uma boa sugestão é retornar a esse tutorial depois que tiver alguma experiência no R para sedimentar os conceitos aqui apresentados. Abaixo o esquema que apresentamos em aula e no inicio desse tutorial. Garanta que consegue se localizar nele.

MapeaR



Siga para a aba de exercícios para fazer os exercícios associados a esse tutorial.

1)

tecnicamente chamados no R de `environments`

2)

diretório é o termo técnico

3)

o nome técnico das pastas do computador

4)

sempre consulte a documentação antes de usar uma nova função

5)

tecnicamente chamado de *Global Environment* ou *Workspace*

6)

pode encontrar também como sendo `.Rda`, existe outro formato de arquivo de dados no R que é o `.Rds` que armazena apenas um objeto

7)

argumento `file =`

8)

detalhes na apostila

9)

chamados de “Environments”

10)

CRAN, GitHub e Bioconductor
[11)]

https://www.rdocumentation.org/
[12)]

VEJA A DOCUMENTAÇÃO!!!