Where We Work…

Susan Holmes (c)

Paths and directory names:

We are going to make sure we know where we are and how to save our work.

In the windows system, our folder would have as its address:
`C:\Users\susan\RWork\`,

R inherits its file and folder naming conventions from unix and we will call this:
`C:/Users/susan/RWork/` with front slashes instead of backslashes.

The Mac OSX already uses the unix standards, the address is usually of the form `/Users/susan/RWork/`.

Notes:
- 1 In R, we use forward slashes for the directories because backslashes are actually used for a different purpose (as what we call escape characters to isolate special characters and stop them from being immediately interpreted.)

Actually in windows: `("C:\\Path\\To\\A\\File")` is the same as `("C:/Path/To/A/File")`

- 2 As a rule, to avoid problems, we should also make sure not to have directories and folders with spaces and special characters.

When having issues with complicated path names, it is work knowing that you can have autocompletion on paths using the tab, in the same way we have it for functions and arguments, the backslash serves to escape and is a prefix for non standard characters in paths.

Question: Try writing `setwd("/U")` on a Mac or `setwd("C:/U")` on Windows and then using tab to fill in a possible directory to change to. What happens in you just use `setwd()` with tab? Let's look more into how functions are documented.

Question: Use `?setwd` ,what do you see.

Comments on a help page.

```
getwd                    package:base                    R Documentation
```

Get or Set Working Directory

Description: `getwd` returns an absolute filepath representing the current working directory of the R process; `setwd(dir)` is used to set the working directory to `dir`.

Usage:
`getwd() setwd(dir)`

Arguments:

```
 `dir`: A character string: tilde expansion will be done.
```

Value:

```
`getwd` returns a character string or `NULL` if the working
directory is not available.  On Windows the path returned will use
`/` as the path separator and be encoded in UTF-8.
The path will not have a trailing `/` unless it is the root directory
(of  a drive or share on Windows).

`setwd` returns the current directory before the change, invisibly
and with the same conventions as 'getwd'.  It will give an error
if it does not succeed (including if it is not implemented).
```

Note:
Note that the return value is said to be *an* absolute filepath: there can be more than one representation of the path to a directory and on some OSes the value returned can differ after changing directories and changing back to the same directory (for example if symbolic links have been traversed).

See Also: `list.files` for the _`contents`_ of a directory. `normalizePath` for a *canonical* path name.

Examples:
`(WD <- getwd()) if (!is.null(WD)) setwd(WD)`

General format of help pages

All help pages are structured in the same way, first the actual name of the function and the package it belongs to.

Then the usual way it is used (**Usage**), in this example we can see that `getwd()` doesn't take an argument and that `setwd(dir)` does.

The **Arguments** section tells us what input is expected, here we are told `dir: a character string`, this means that if I enter:
`setwd(/User/susan/RWork)` will generate an error.
`Error: unexpected '/' in "setwd(/"`

A character string is always enclosed in quotes (of either kind, " or '"'). Sometimes you can be tricked into thinking that a character string is not being used for instance in some code you might see `setwd(filepath)`, what has happened is that in previous code, the user has defined the variable filepath with something like:
`filepath <- "/Users/susan/RWork"`

Question: Look at the help file for the function `list.files` and use it to give the list of files in your home directory.

Making vectors

We are now going to do a small exercise, to show how R reads and writes data on your local machine.

If you type `2:17` then return, you will see:

```
##  [1]  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17
```

R has made a vector of 16 values from 2 to 17. The `[1]` in front of the first element denotes the first index. If we had a very long vector, you'll see many lines of output with an index in square brackets at the beginning of every line giving that first element's index.

```
2:171
```

```
##   [1]   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
##  [18]  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35
##  [35]  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52
##  [52]  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69
##  [69]  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86
##  [86]  87  88  89  90  91  92  93  94  95  96  97  98  99 100 101 102 103
## [103] 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120
## [120] 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137
## [137] 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154
## [154] 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171
```

Look at what you have in your R workspace currently with

```
ls()
```

Now name your vector, we say we assign the contents to `vectint` by typing

```
vectint <- (2:17)
```

You see no output. It has been stored.
In order to see the data we have to type its name:

```
vectint
```

```
##  [1]  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17
```

Now we create a different vector

```
sqrs <- vectint^2
```

If you type `sqrs`, what will you see?

Now let's take a look at the workspace again.

```
ls()
```

```
## [1] "sqrs"    "vectint"
```

On the right in the Environment pane you may have noticed, something was added, `sqrs` now exists in our R environment. We can also check that by typing:

```
ls()
```

```
## [1] "sqrs"    "vectint"
```

Upper case and lower case

R, Mac and unix make the difference between UPPER CASE and lower case, so if we write:

```
c(45,32,11)
```

```
## [1] 45 32 11
```

```
VECA = c(45,32,11)
```

What happens when you type:

```
Veca
```

The names are different, this true of function and variable names as well as files and directories.

This can be confusing because it is not the case in Windows.

Functions we have seen

Try to name three functions we have used.

How can you tell what a function is?

In fact, we started our first session with the `getwd()` function, we went on to use `c()` and `ls()`.

You'll note that all these are used with a name followed by two brackets.

Using functions to save our work

The most common use of a function is to have arguments inside the brackets. Let's explore a very useful one, with the help of the prompt and the the tab completion, suppose we want to save our `sqrs` vector.

Start to type `sav` in the console, if autocompletion doesn't come up, type the tab : a small window comes up with proposals of functions you might to look, let's look at the first one.

If you type just `save()` in R what does it say?

This tells us that save unlike some of the other functions we have seen such as `getwd()`, `q()` needs to have at least one argument, it actually needs several: the first is the name of the object we want to dave, without quotes, the second is a file and we have to give quotes to tell the system that this is a filename.

```
save(sqrs,file="sqrs.RData")
```

If you want to save everything that you have in our workspace, not just one object, you can use the `Save choice` (a floppy disk icon in the Environment pane) then you will be asked to give a file name, choose something explicit like the project and date, without spaces or special characters.

When you enter the filename and save, a command appears in the console pane, for instance:

```
save.image("~/Dropbox/Stat32/Slides_ABCofR/Stats32Example1.RData")
```

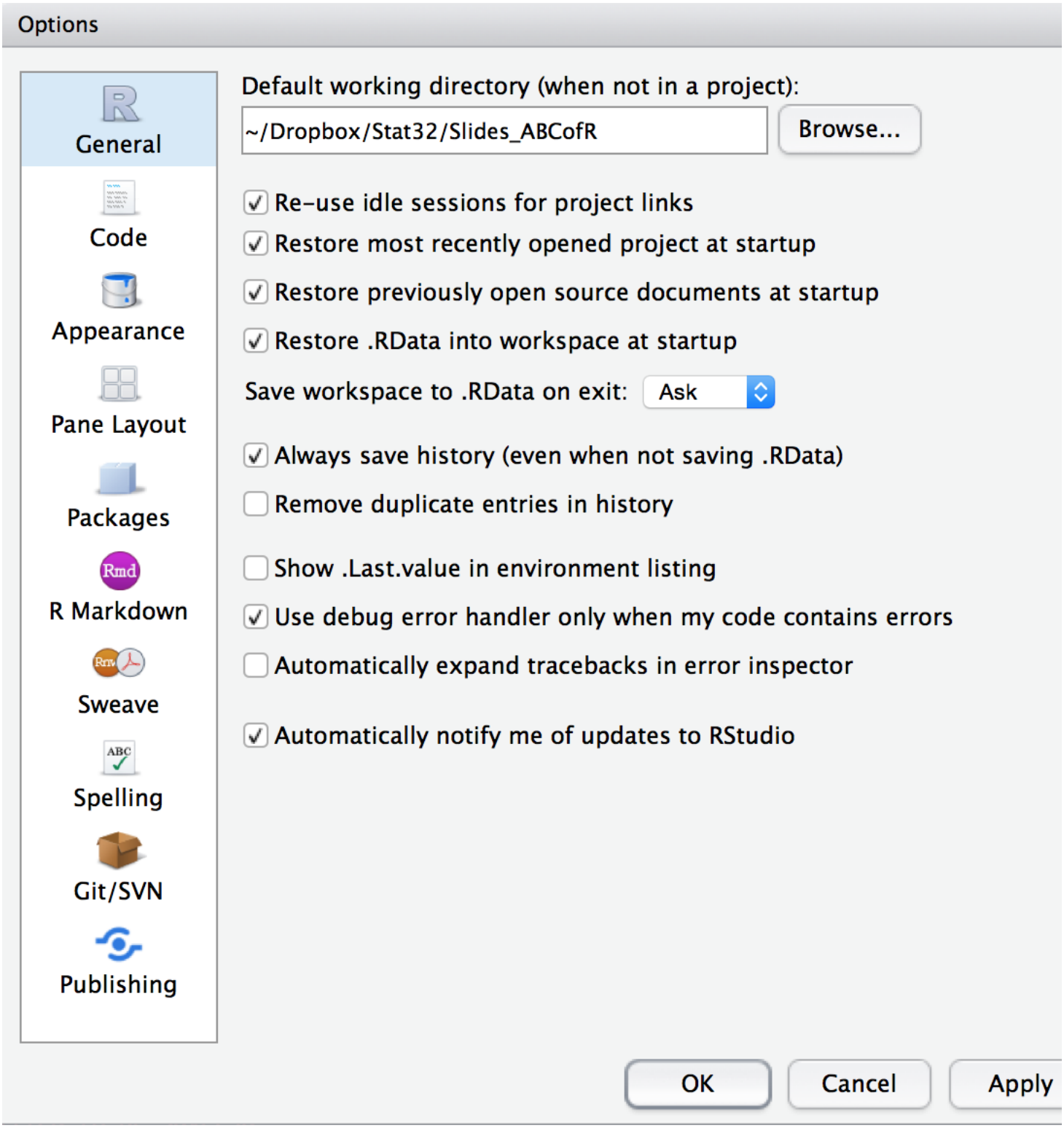
This shows you the command R used to save my workspace. The ~ stands for my own directory and this is equivalent to the complete path: `"/Users/susan/Dropbox/Stat32/Slides_ABCofR/Stats32Example1.RData"`

If you simply type

```
save.image()
```

everything in your current workspace will be written into a file called ".Data". Next time you start R, it will load that data if there is a file of that type in the driectory you started from.

You can tell RStudio where to work from by bringing up the Global Options choice in the Tools menu in Windows (Preferences menu under the RStudio tab on the Mac)



Options and Preferences for RStudio

If you list the files in the directory now:

```
list.files()
```

you will see these new files that have been created. These create a permanent record of your work and are important. If you do not save when you quit R, all the work you did will be lost. It is also beneficial to keep the history of your commands and this is also a choice you can make in the preferences and options pane.

Type

```
history()
```

and you will see the record in the right hand pane, you can save it by clicking on the diskette icon or using the command line:

```
savehistory("~/Dropbox/Stat32/Slides_ABCofR/introsession.Rhistory")
```

Now quit

```
q()
```

and here you understand that R is asking whether to save your workspace for next time, this time you can say *y* if you like.

Summary of this session:

- We learnt about how to read a help pane carefully: the output from `?getwd`.
- What a file path and working directory were: `getwd()` and `setwd()`.
- How to create a vector in your workspace, using `<-` and `c()`.
- How and where your data are saved in R (.RData and .Rhistory).

Question Now start RStudio again and look at what data has been loaded by typing:

```
ls()
```

So before even starting a new session you can easily pick up where you left off.

Try creating another vector, saving it and saving your history before quitting RStudio. Now start again, make sure your workspace is as you want it.

If you need to cleanup, after saving what you need to save you can type:

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

Check using `ls()` that your workspace is clear.