Data analysis and visualization in R

GROW UC Merced 2020

Andrea Sánchez-Tapia

25th August 2020

outline

- Overview to R and RStudio
- Introduction to R
- Starting with Data
- Manipulating Data Frames with **dplyr**
- Data visualisation

overview of R and RStudio

why learn R?

- ***libre* software**: free and free-to-be-used-and-modified for any means -> one of the pillars of open science
- **script-based**: reproducibility, easy to scale up your analyses, transparency (track errors), great way to learn about methods.
- **interdisciplinary and modular**: lots of code written by area specialists. At the core of its philosophy is a smooth transition from user to programmer.
- **communication** with other tools: manuscripts, presentations, apps and dashboards

why learn R?

- communication with **other programming languages** (ex. **reticulate** to run python scripts)
- great **graphic capabilities**!
- **official support**: help in documentation, mailing lists
- **an active and welcoming community**: email lists, Stack Overflow, RStudio community, useR groups, R-Ladies+ chapters, Slack communities, `#rstats`

R has a modular structure: packages

- R **base** installation includes base packages developed and maintained by the **R Core Development Team**
- other packages are created by the community
- hosted in **CRAN** (The Comprehensive R Archive Network) or Bioconductor, GitHub, rOpenSci.org
- a package is a collection of functions, it must be loaded to be used (ex. `library(dplyr)`)
- the whole package is loaded, not some functions or parts of it. if you want to use one function you can use `package::function()`

how to run R

- from the R program in Windows ~~but don't~~
- directly in the **terminal** in Linux and Mac (just type `R`). this is important in **HPC** environments like the UC Merced cluster (R scripts can also be run from outside R)
- many GUIs and text editors: rgedit, **emacs ESS**, Atom, Sublime Text etc.
- **RStudio**: an Integrated Development Environment (IDE) - Desktop Version but also Server and Cloud versions

install, load, and cite packages

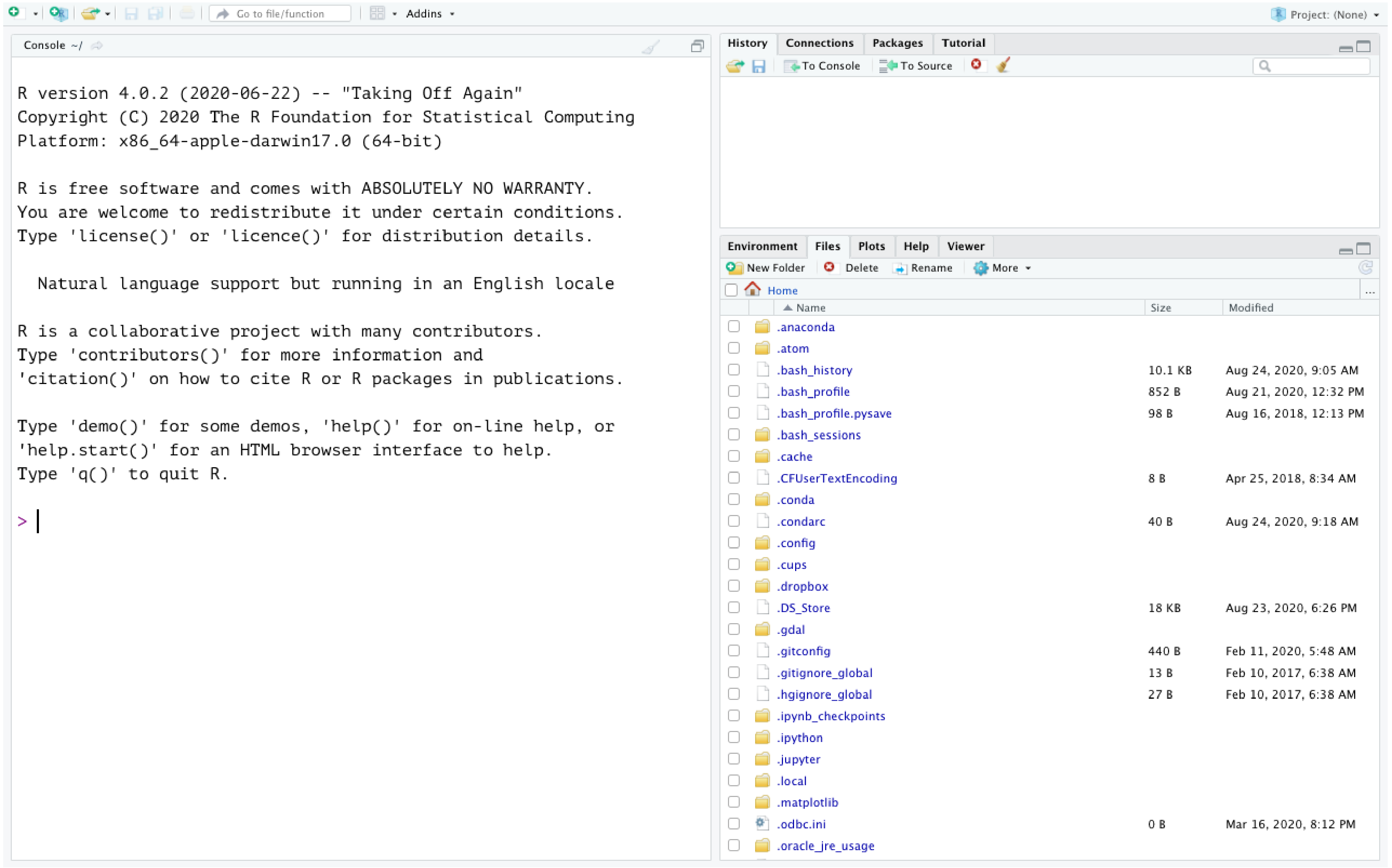
```r
install.packages("dplyr")
install.packages("ggplot2")
install.packages("tidyverse") # just an umbrella package
```

- if you get an error about a missing **dependency** copy the name of the missing package(s) and execute `install.packages()`

```r
library("dplyr")
citation("dplyr")
```

about notation

- **packages** are collections of **functions**
- **functions** have **arguments** or **parameters** (options)
- To designate them:
  - package name: **base** (in bold letters)
  - function name: `help()` (in fixed width font and with parentheses at the end)
  - objects and arguments: `data`, `na.rm` (in fixed width font)
  - sometimes you'll see `stats::median()` this is correct syntaxis `program::function()` and helps distinguish **functions with the same name** or calling one function only (from an **installed** package)



```
R version 4.0.2 (2020-06-22) -- "Taking Off Again"
Copyright (C) 2020 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin17.0 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> 
```

Name	Size	Modified
.anaconda		
.atom		
.bash_history	10.1 KB	Aug 24, 2020, 9:05 AM
.bash_profile	852 B	Aug 21, 2020, 12:32 PM
.bash_profile.pysave	98 B	Aug 16, 2018, 12:13 PM
.bash_sessions		
.cache		
.CFUserTextEncoding	8 B	Apr 25, 2018, 8:34 AM
.conda		
.condarc	40 B	Aug 24, 2020, 9:18 AM
.config		
.cups		
.dropbox		
.DS_Store	18 KB	Aug 23, 2020, 6:26 PM
.gdal		
.gitconfig	440 B	Feb 11, 2020, 5:48 AM
.gitignore_global	13 B	Feb 10, 2017, 6:38 AM
.hgignore_global	27 B	Feb 10, 2017, 6:38 AM
.ipynb_checkpoints		
.ipython		
.jupyter		
.local		
.matplotlib		
.odbc.ini	0 B	Mar 16, 2020, 8:12 PM
.oracle_jre_usage		

- Check for the following panes:
  - Environment -> Objects in the **workspace**
  - Files
  - Plots
  - Help
  - Console

Some other that might be useful *later*:

- **Terminal**
- **Viewer** (for presentations and documents)
- **git** (only when working inside an RStudio project)

working directory

- the **files** pane is showing one folder location:
    - `Home/Documents`
    - `~/Documents`
    - `"/Users/andreasancheztapia/Documents"`
- in `Global options > General > R Sessions` OR `cmd` + `,` "default working directory when not in a project"
- `getwd()` in the console
- we have to tell R where we are working -> change the working directory

project organization

- projects are better organized if we use **one folder per project** and **subfolders** within our working directory
- take care of data **provenance**: we shouldn't modify **raw data files** but **save processed data** (and the corresponding scripts)

In practice:

```
project/
    ├── data/
    ├── raw_data/
    ├── docs/
    ├── figs/
    ├── R/
    ├── output/
    └── README.md
```

Hands on:

1. Select our folder for this project
2. Create a subfolder structure: `/figs`, `/data_raw`, `/data`

RStudio projects

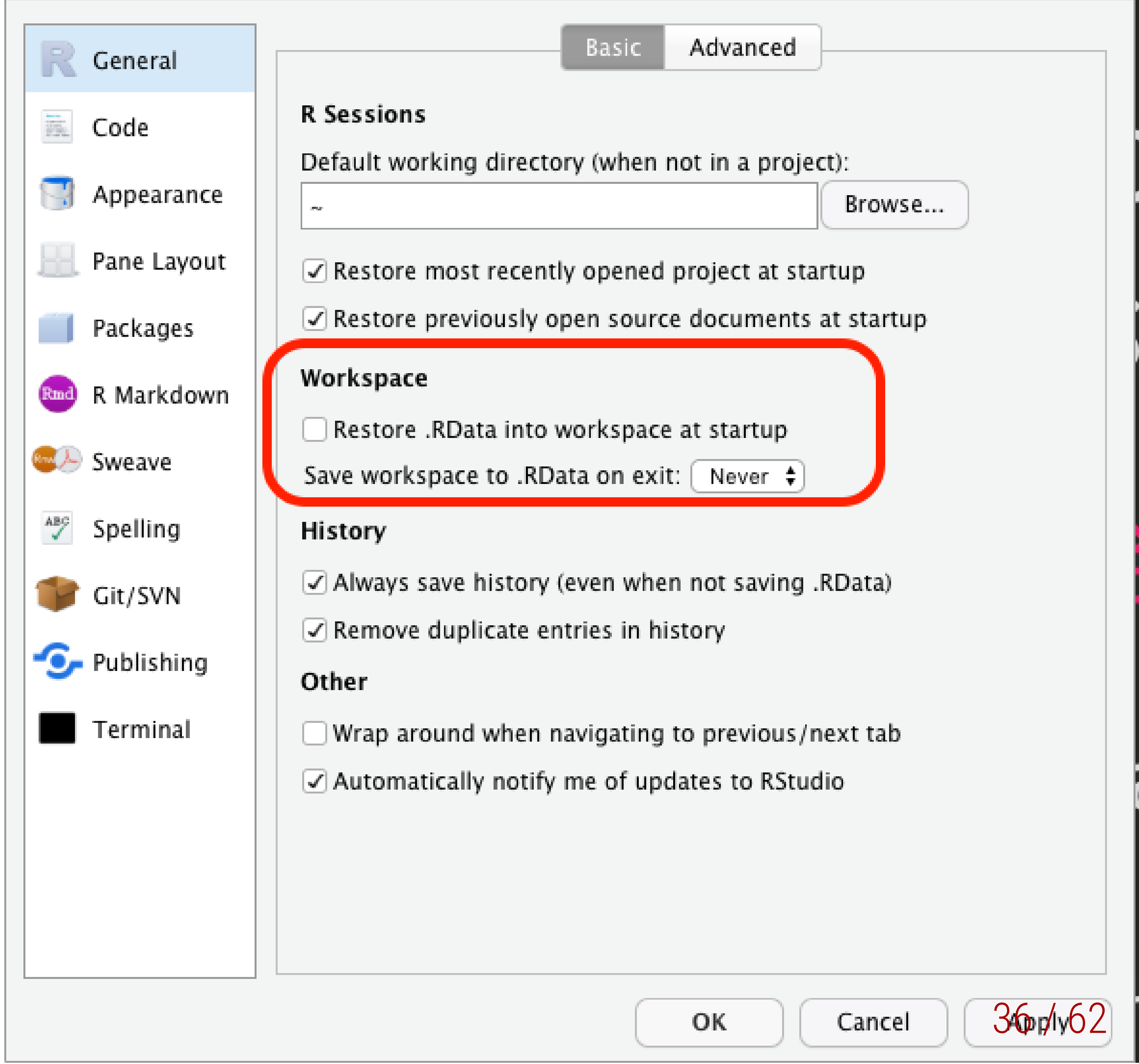
RStudio projects create a .Rproj file in your folder that acts as a shortcut for your projects

- recognize the location
- respect some project-specific preferences
- reopen files
- **git** pane available

about the workspace

- R creates **objects** that occupy RAM memory: the **workspace**
- the **workspace** can be saved and loaded between sessions BUT
- **you can lose track of how you created the objects in the workspace**
- `#goodpractices` don't save the workspace

in the general options



General
Code
Appearance
Pane Layout
Packages
R Markdown
Sweave
Spelling
Git/SVN
Publishing
Terminal

Basic | Advanced

R Sessions

Default working directory (when not in a project):
~ Browse...

[✓] Restore most recently opened project at startup

[✓] Restore previously open source documents at startup

Workspace

[ ] Restore .RData into workspace at startup

Save workspace to .RData on exit: Never

History

[✓] Always save history (even when not saving .RData)

[✓] Remove duplicate entries in history

Other

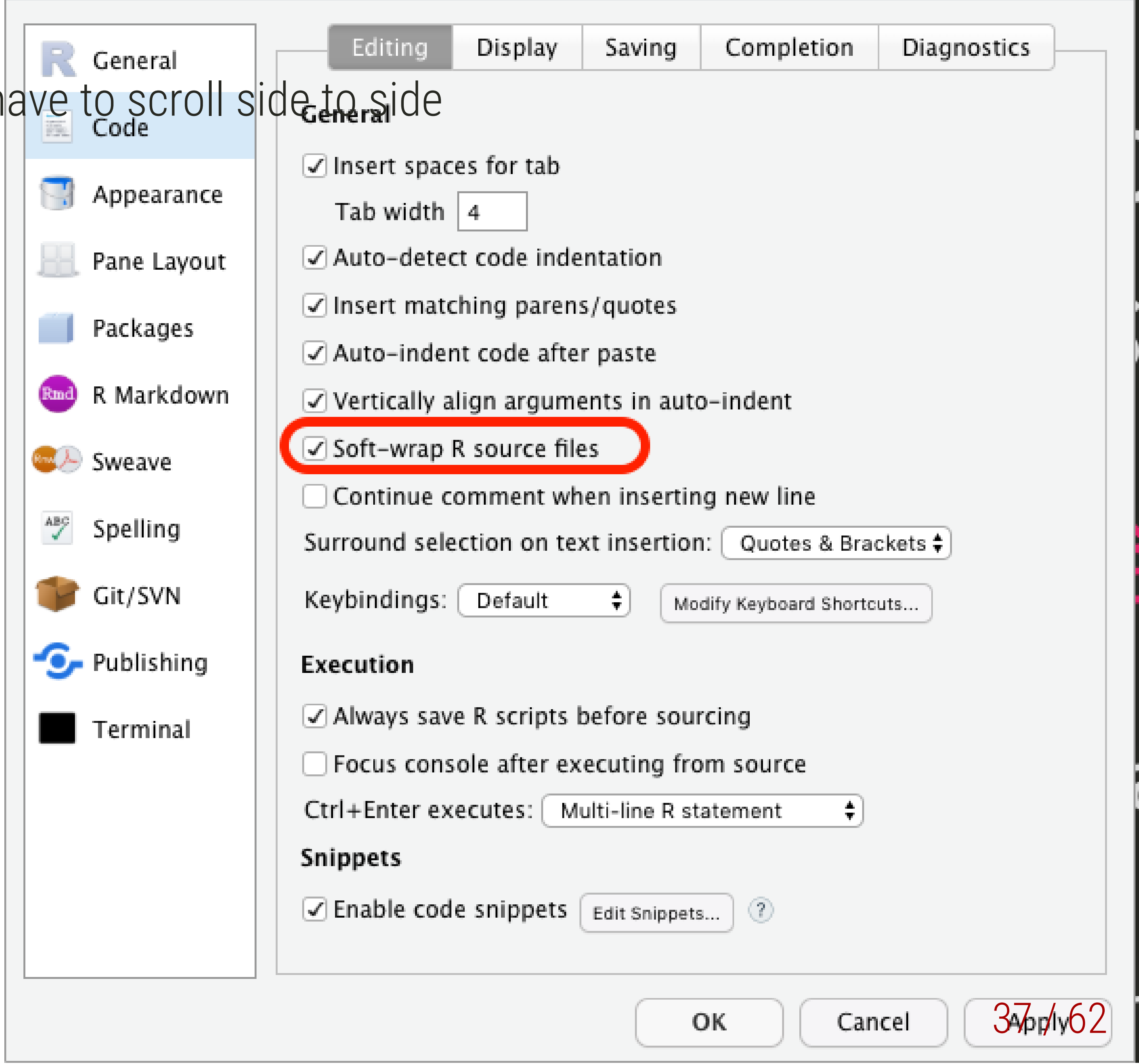
[ ] Wrap around when navigating to previous/next tab

[✓] Automatically notify me of updates to RStudio

OK Cancel Apply

soft wrap your scripts so you don't have to scroll side to side

soft wrap your scripts so you don't have to scroll side to side



General

- ✓ Insert spaces for tab
- Tab width 4
- ✓ Auto-detect code indentation
- ✓ Insert matching parens/quotes
- ✓ Auto-indent code after paste
- ✓ Vertically align arguments in auto-indent
- ✓ Soft-wrap R source files
- ☐ Continue comment when inserting new line
- Surround selection on text insertion: Quotes & Brackets
- Keybindings: Default   Modify Keyboard Shortcuts...

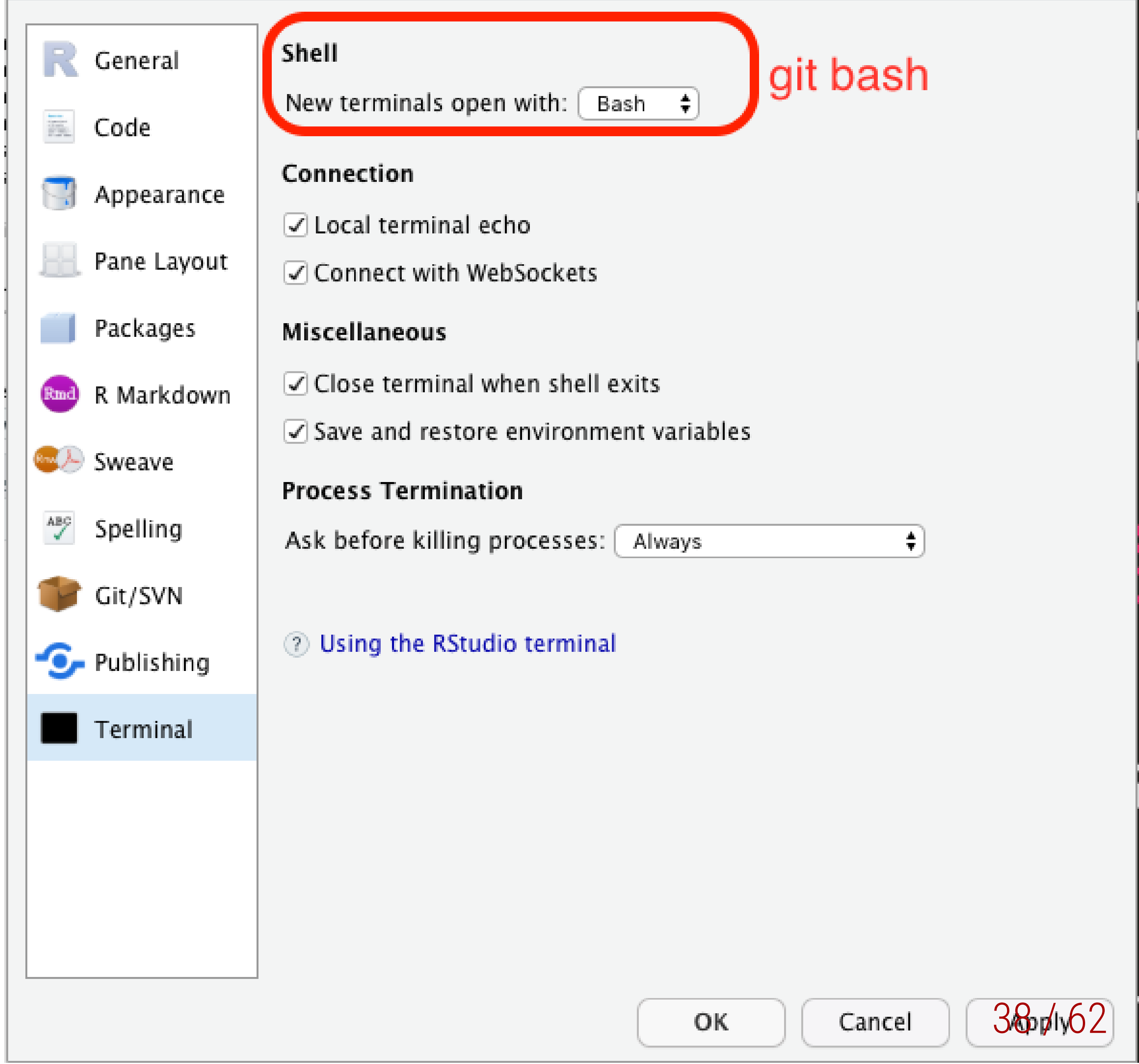
Execution

- ✓ Always save R scripts before sourcing
- ☐ Focus console after executing from source
- Ctrl+Enter executes: Multi-line R statement

Snippets

- ✓ Enable code snippets   Edit Snippets...

check your terminal



Shell

New terminals open with: Bash

git bash

Connection

☑ Local terminal echo

☑ Connect with WebSockets

Miscellaneous

☑ Close terminal when shell exits

☑ Save and restore environment variables

Process Termination

Ask before killing processes: Always

Using the RStudio terminal

- we have an **RStudio project** in the correct **working directory**, with a nice file structure and RStudio is configured
- how did package installation go?

introduction to R

introduction to R

- `<-` is the assignment operation in R and it does not return output
- overwriting objects **does not affect other objects**
- **naming things**: don't begin with a number or symbol. be consistent with your **coding style**!
- separators can be anything and (you could use `.` but be nice).

data types in R

```r
animals  <- c("mouse", "rat", "dog")
weight_g <- c(50, 60, 65, 82)
```

```r
class(animals)
```

```
## [1] "character"
```

```r
class(weight_g)
```

```
## [1] "numeric"
```

`character` and `numeric` but also `logical` and `integer` ("whole" numbers, with no decimal component, in N), `complex`, and others.

subsetting vectors

- R is **1-indexed** and intervals are closed (not half-open)

```r
animals <- c("mouse", "rat", "dog", "cat")
animals[2]
```

```
## [1] "rat"
```

- Subsetting is done with brackets `[]`

```r
animals[c(3, 2)]
```

```
## [1] "dog" "rat"
```

conditional subsetting

```r
weight_g <- c(21, 34, 39, 54, 55)
weight_g[c(TRUE, FALSE, FALSE, TRUE, TRUE)]
```

```
## [1] 21 54 55
```

Nobody works like this, instead we use **logical clauses** to **generate** these logical vectors

logical clauses

- equality or not: `==`, `!=`
- inequalities: `<`. `>`, `<=`, `>=`
- union (OR) `|`
- intersection (AND) `&`
- belonging `%in%`
- differences between sets: `setdiff()`
- negation works `!`: "not in" `!a %in% b`

comparing vectors

```r
animals      <- c("mouse", "rat", "dog", "cat")
more_animals <- c("rat", "cat", "dog", "duck", "goat")

animals %in% more_animals
```

```
## [1] FALSE  TRUE  TRUE  TRUE
```

comparing vectors

```r
animals      <- c("mouse", "rat", "dog", "cat")
more_animals <- c("rat", "cat", "dog", "duck", "goat")

animals == more_animals
```

```
## Warning in animals == more_animals: longer object length is not a multipl
```

```
## [1] FALSE FALSE  TRUE FALSE FALSE
```

- Vectors are compared **one by one AND recycled** when one of them is shorter, so use `%in%` when you want to check **belonging to a set**

missing data

```r
heights <- c(2, 4, 4, NA, 6)
mean(heights)
```

```
## [1] NA
```

```r
max(heights)
```

```
## [1] NA
```

```r
mean(heights, na.rm = TRUE)
```

```
## [1] 4
```

```r
max(heights, na.rm = TRUE)
```

data structures

- **vector**: lineal arrays (one dimension: only length)
- **factors**: vectors (one-dimensional) representing **categorical variables** and thus having **levels**
- **matrices**: arrays of vectors -> the same type (all numeric or all character, for instance) (two dimensions: width and length)
- **data frames**: two-dimensional arrays but might be of combined types (i.e., column 1 with names, column 2 with numbers)
- **arrays** are similar to matrices and dataframes but may be three-dimensional ("layered" data frames)
- **list**: literally a list of anything (a list of data frames, or different objects)