Transitioning from R Markdown to Quarto

A One-Week Boot Camp for R Markdown Users

The rgtlab Curriculum Project

2026-11-13

Transitioning from R Markdown to Quarto

A One-Week Boot Camp

The rgtlab Curriculum Project

Welcome

This is the online version of **Transitioning from R Markdown to Quarto: A One-Week Boot Camp** by The rgtlab Curriculum Project. We wrote it for analysts who already write R Markdown and wish to move their work to Quarto.

We cover what an experienced R Markdown user needs in order to become fluent in Quarto over one week. That is, installing Quarto and rendering an existing `.Rmd` file unchanged; the document anatomy that differs from R Markdown, including the `#|` cell-option syntax and the execution model; output formats, figures, tables, and native cross-references; projects at scale, from websites to books to manuscripts; and reproducible publishing, with a systematic checklist for migrating a real project. The book is designed for **five consecutive days** of work: one hour of lecture content each day, two hours of homework, no examinations.

Our organizing principle is that Quarto is a superset of what R Markdown already does. The transition is therefore a matter of translation and extension rather than of starting over. An analyst who can write an `.Rmd` file today already knows most of what Quarto asks. We map the familiar constructs onto their Quarto equivalents and then show what Quarto adds beyond them. Each chapter opens with a concrete authoring problem, develops the Quarto mechanism that addresses it, and returns to the problem to judge what the mechanism did and did not solve.

The workshop is a companion to the rgtlab curriculum sequence:

- **R for Biostatistics: A One-Week Boot Camp** (https://r-bootcamp.rgtlab.org), the preparatory volume for R.
- *Git and GitHub for Biostatistics*, the preparatory volume for version control, on which the publishing chapter of this workshop draws.
- *Biostatistics Practicum*, workflow infrastructure (Git in depth, Docker, renv, Quarto, CDISC) that builds directly on this workshop.

- *Reproducible Research for the Health Sciences*, the reproducibility volume that extends the freeze, caching, and publishing material introduced here.

A reader who has completed *R for Biostatistics* and *Git and GitHub for Biostatistics* arrives at this workshop ready to place a Quarto project under version control and publish it reproducibly by the end of the week.

We refer the reader to the Preface for the design rationale and to the Conventions page for the visual cues used throughout.

License

This book is licensed under Creative Commons Attribution 4.0 International.

The code samples printed in this book are licensed under the MIT License. The repository code (package functions and build scripts) is licensed under GPL-3.

Transitioning from R Markdown to Quarto

A One-Week Boot Camp for R Markdown Users.

Draft, November 13, 2026. Content may change.

Copyright

Transitioning from R Markdown to Quarto: A One-Week Boot Camp by rgtlab curriculum team is licensed under a Creative Commons Attribution 4.0 International License.

The code samples printed in this book are licensed under the MIT License. The repository code (package functions and build scripts) is licensed under GPL-3.

To cite this book, please use:

> rgtlab curriculum team (2026). *Transitioning from R Markdown to Quarto: A One-Week Boot Camp*. Available at https://rmd-to-quarto.rgtlab.org.

Table of contents

License 2

Copyright **5**

Preface **11**
Why move from R Markdown to Quarto 11
What this book covers 12
What this book does not cover 13
How this book is meant to be used 13
Prerequisites 14
Acknowledgments 14

Conventions **15**
Commands and output 15
Callouts 16
Comparisons with R Markdown 16
Cross-references 17
Chapter structure 17

How to use this book **19**
The daily cadence 19
Chapter template 19
A note on the two tools 20
Conventions 20
Companion volumes 21

I. Five Days **23**

1. Day 1: From R Markdown to Quarto, Your First Render **25**
1.1. Learning objectives 25

1.2. Lecture 25
1.2.1. Why transition, and what carries over 25
1.2.2. Installing Quarto and the command-line interface . . 26
1.2.3. The .qmd format compared with .Rmd 26
1.2.4. Rendering an existing .Rmd unchanged 27
1.2.5. RStudio, Positron, VS Code, Vim, and terminal workflows 27
1.2.6. Project anatomy 28
1.3. Worked example: rendering a colleague's report under Quarto 28
1.4. Homework 29
1.5. Solutions 29
1.6. What's next 30

2. Day 2: Document Anatomy: YAML, Chunks, and the Execution Model 31
2.1. Learning objectives 31
2.2. Lecture 31
2.2.1. YAML front matter differences 31
2.2.2. The `#|` cell-option syntax 32
2.2.3. Execution options in place of opts_chunk 32
2.2.4. Controlling what runs and what shows 33
2.2.5. The knitr and jupyter engines 33
2.2.6. Inline code 33
2.3. Worked example: modernizing a setup chunk 34
2.4. Homework 35
2.5. Solutions 35
2.6. What's next 36

3. Day 3: Output Formats, Figures, Tables, and Cross-References 37
3.1. Learning objectives 37
3.2. Lecture 37
3.2.1. One source, several formats 37
3.2.2. Figures and captions 38
3.2.3. Tables and captions 38
3.2.4. Native cross-references 39
3.2.5. Citations and CSL 39
3.2.6. Callouts and margin content 40
3.2.7. Theming 40

Draft, November 13, 2026. Content may change.

3.3. Worked example: a cross-referenced results section 40
3.4. Homework 41
3.5. Solutions 42
3.6. What's next 42

4. Day 4: Projects at Scale: Websites, Books, and Manuscripts 43
4.1. Learning objectives 43
4.2. Lecture 43
4.2.1. The project and `_quarto.yml` 43
4.2.2. Websites 44
4.2.3. Books 44
4.2.4. The manuscript project type 45
4.2.5. Parameterized reports 45
4.2.6. Includes and shortcodes 45
4.2.7. Profiles 45
4.3. Worked example: a two-page analysis website 46
4.4. Homework 47
4.5. Solutions 47
4.6. What's next 48

5. Day 5: Reproducibility, Publishing, and Migrating a Real Project 49
5.1. Learning objectives 49
5.2. Lecture 49
5.2.1. Freeze and caching 49
5.2.2. renv integration 50
5.2.3. Publishing 50
5.2.4. Continuous integration with GitHub Actions 51
5.2.5. A migration checklist 52
5.3. Worked example: migrating a small project end to end . . . 52
5.4. Homework 53
5.5. Solutions 54
5.6. What's next 54

References 55

Appendices 57

Credits 57

Colophon **59**

Draft, November 13, 2026. Content may change.

Preface

This volume is a one-week workshop for analysts who already write R Markdown and wish to move their work to Quarto. We assume fluency with the R Markdown they use every day: knitr chunks, YAML front matter, and rendering to HTML, PDF, and Word. A beginner's book would teach literate programming from nothing. This one teaches the translation from a tool the reader knows to one they do not yet know. Our goal is a working migration, not a first exposure.

The design constraint is deliberate. Five lectures and ten hours of homework should leave the reader able to render, extend, and publish Quarto documents, and able to migrate an existing R Markdown project. We meet this constraint by building on what the reader already knows. Quarto shares R Markdown's Pandoc foundation and its knitr engine, so most of an `.Rmd` file transfers unchanged. We spend our time on the differences that matter and on the capabilities that R Markdown lacked.

Why move from R Markdown to Quarto

The case for the transition is not that R Markdown is broken. R Markdown continues to work, and existing documents will render for years. Rather, the case is that Quarto consolidates a scattered ecosystem into one tool. Consider three frictions that an R Markdown user has met:

- Producing a slide deck, a website, and a PDF from related content meant learning `xaringan`, `blogdown`, and `bookdown` separately, each with its own conventions.
- Cross-referencing a figure or a table required a package extension and a syntax that differed by output format.
- Sharing a document with a colleague who works in Python meant that the R Markdown toolchain did not apply.

Quarto addresses each directly. One command-line tool renders slides, sites, books, and articles. Cross-references are native and uniform across formats, and the same document model serves R, Python, and Julia. An analyst who internalizes Quarto writes once and targets many outputs, which is what R Markdown promised and Quarto more fully delivers.

What this book covers

Five days, one chapter each:

1. **From R Markdown to Quarto, your first render.** Why transition; installing Quarto and the command-line interface; the `.qmd` format compared with `.Rmd`; rendering an existing `.Rmd` unchanged; the RStudio, VS Code, and terminal workflows; project anatomy.
2. **Document anatomy: YAML, chunks, and the execution model.** How the YAML front matter differs from R Markdown; the `#|` cell-option syntax compared with `knitr::opts_chunk`; execution options (`echo`, `eval`, `warning`, `message`, `freeze`); the knitr and jupyter engines; inline code.
3. **Output formats, figures, tables, and cross-references.** HTML, PDF, Word, and revealjs; figure and table handling and captions; native cross-references (`@fig-`, `@tbl-`, `@sec-`); citations and CSL; callouts and margin content; theming.
4. **Projects at scale: websites, books, and manuscripts.** Quarto projects; `_quarto.yml`; websites and books; the manuscript project type; parameterized reports; includes and shortcodes; profiles.
5. **Reproducibility, publishing, and migrating a real project.** Freeze and caching; renv integration; publishing with `quarto publish`, Netlify, GitHub Pages, and Quarto Pub; continuous integration with GitHub Actions; a systematic `.Rmd`-to-`.qmd` migration checklist applied to a whole project.

Each chapter is approximately 1 hour of reading and worked examples plus 2 hours of homework problems, with worked solutions provided.

Draft, November 13, 2026. Content may change.

What this book does not cover

We deliberately omit, and point elsewhere for, material that surrounds the transition but is not the transition itself:

- Learning R itself, or R Markdown from scratch. We assume the reader arrives with both. See *R for Biostatistics*.
- Version control with Git and GitHub. The publishing chapter uses Git, but does not teach it. See *Git and GitHub for Biostatistics*.
- Containerized and dependency-pinned reproducibility in depth (Docker, renv at scale, validated pipelines). See the *Biostatistics Practicum* and *Reproducible Research for the Health Sciences*.
- Extension and filter authoring (Lua filters, custom formats). We use extensions but do not write them.

The workshop is intentionally narrow. The reader finishes Day 5 able to work fluently in Quarto and to migrate a real project. This leaves the follow-on volumes free to teach the infrastructure that surrounds it.

How this book is meant to be used

The five-day cadence is the spine. A reader starting the week having never run `quarto render` should be able to finish the week publishing a Quarto site. The cadence assumes roughly 3 hours of focused work per day:

- **Hour 1:** Read the chapter, running every command in your own terminal and rendering every document as you reach it. Quarto is learned by rendering, not by reading.
- **Hours 2-3:** Complete the homework problems. Solutions are at the end of each chapter; check yourself only after attempting each problem.

The commands in this book are shown for a Unix-style shell (macOS Terminal, Linux, or Git Bash on Windows). Where a step differs on Windows, the difference is noted.

Prerequisites

We assume:

- A working laptop running macOS, Windows, or Linux, with a recent R installation.
- Practical familiarity with R Markdown: writing knitr chunks, editing YAML front matter, and rendering to HTML, PDF, or Word.
- Basic familiarity with the command line: opening a terminal, changing directories with `cd`, listing files with `ls`.

Prior completion of *R for Biostatistics* and *Git and GitHub for Biostatistics* is helpful, since the worked examples version Quarto projects and publish them. Neither is strictly required, however, since the Quarto mechanics are independent of how the reader learned R or Git.

Acknowledgments

This workshop consolidates patterns we developed while migrating our own teaching and research materials from R Markdown to Quarto. The Quarto documentation at https://quarto.org and the writings of the R Markdown authors provide the technical scaffolding. The workshop pedagogy itself reflects our experience with what an R Markdown user actually needs in order to make the transition, as distinct from the full capabilities of the tool.

Conventions

This page summarizes the visual conventions used throughout the book.

Commands and output

This book mixes terminal commands with document source. Shell commands appear in syntax-highlighted blocks. A command the reader types is shown after a `$` prompt; the prompt itself is not typed. Output appears on the following lines, without a prompt:

```
$ quarto --version
1.5.57
```

When a sequence of commands is shown without output, the prompts are omitted for readability:

```
quarto render report.qmd
quarto preview report.qmd
```

Inline commands, file names, and Quarto options are set in `monospace`. Quarto subcommands are named in full (`quarto render`, not merely `render`) so that they are unambiguous. Document source, including YAML front matter and executable cells, is shown in fenced blocks labeled by language:

```
title: 'Baseline summary'
format: html
```

Placeholders that the reader must replace with their own values are written in angle brackets, for example `quarto render <file>` or `quarto publish <target>`. The angle brackets are not typed.

Callouts

Three callout types appear:

> **Tip**
>
> A small practical recommendation.

> **Check your understanding: example**
>
> A short question testing comprehension of the just-read material. Click to expand the answer.

> **Warning**
>
> A pitfall the reader may otherwise hit, most often a difference from R Markdown that silently changes behavior.

Comparisons with R Markdown

Because this book is a transition, we frequently place an R Markdown construct beside its Quarto equivalent. Such pairs are shown as two adjacent code blocks, the `.Rmd` form first and the `.qmd` form second, so that the reader can read the translation directly:

```
R Markdown:  ```{r, echo=FALSE}
Quarto:      ```{r}
             #| echo: false
```

These comparisons are the spine of the book; where a construct transfers unchanged, we say so explicitly.

Draft, November 13, 2026. Content may change.

Cross-references

Within this book, sections are referenced by their Quarto label (`@sec-day1`, `@sec-crossrefs`). These resolve to clickable links in HTML and section numbers in PDF. Figures and tables use the `@fig-` and `@tbl-` prefixes introduced on Day 3.

References to the companion volumes *R for Biostatistics*, *Git and GitHub for Biostatistics*, and the *Biostatistics Practicum* use **prose pointers** rather than Quarto cross-references, because cross-references do not resolve across separate books. For example: 'see the publishing chapter of the companion *Git and GitHub for Biostatistics* volume'.

Chapter structure

Every content chapter follows the same five-section template:

1. **Learning objectives.** What you will be able to do after the day.
2. **Lecture.** The substantive content, with commands to run and documents to render in your own environment as you read.
3. **Worked example.** A small but realistic authoring scenario that uses the day's content end to end.
4. **Homework.** Problems organized from easier to harder.
5. **Solutions.** Worked solutions to all homework problems. Read only after attempting the problem.

Each chapter closes with a short **What's next** pointer. The pattern repeats deliberately; by the third chapter you know where to find each component.

How to use this book

We begin with a short orientation. In this chapter we set out the daily cadence, the chapter template, and the conventions used throughout, so that the reader knows what to expect.

The daily cadence

Each day has the same shape:

- **Lecture content (~1 hour).** Read the chapter, running each command and rendering each document in your own environment as you encounter it. Do not skip the renders; the chapter is designed to be read with a terminal open and a scratch project at hand.
- **Homework (~2 hours).** Five to eight problems, with worked solutions at the end of the chapter. Attempt each problem before checking the solution.

The cadence assumes one chapter per day for five consecutive days. The workshop may be compressed (two chapters per day across two-and-a-half days) or expanded (one chapter per week across a month) depending on the reader's other commitments; we leave the choice to the reader.

Chapter template

Each chapter follows a five-section template, and we shall keep to it consistently:

1. **Learning objectives.** A bulleted list of what the reader should be able to do by the end of the day.

2. **Lecture.** The substantive content for the day, with commands to run and documents to render.
3. **Worked example.** A small but realistic authoring scenario that uses the day's content end to end, most often translating or extending a familiar R Markdown document.
4. **Homework.** Problems organized from easier to harder.
5. **Solutions.** Worked solutions to all homework problems. Read only after attempting the problem.

There are no quizzes, no examinations, and no tests in this book. Self-assessment proceeds entirely through the homework solutions.

A note on the two tools

This book concerns two related but distinct things:

- **R Markdown** is the authoring system the reader already knows. It couples the knitr engine to Pandoc and renders `.Rmd` documents. We treat it as the starting point, not the subject.
- **Quarto** is the newer system the reader is moving to. It is a standalone command-line tool that renders `.qmd` documents, shares R Markdown's knitr engine, and adds a uniform model for many output formats and project types. Quarto is the subject of every chapter.

Keeping the distinction clear from the outset prevents a common confusion in which the two are treated as one, or in which Quarto is assumed to be an R package rather than an independent tool.

Conventions

The visual cues used throughout the book are described on the Conventions page. In brief:

- Commands you type follow a `$` prompt; output has no prompt.
- Placeholders you replace are written in `<angle-brackets>`.
- Document source is shown in fenced blocks labeled by language.
- Command lines and file listings are wrapped at 78 characters.

Draft, November 13, 2026. Content may change.

Companion volumes

This workshop sits within the rgtlab curriculum sequence. The two preparatory boot camps precede it, and two follow-on volumes extend it:

- *R for Biostatistics* (https://r-bootcamp.rgtlab.org)
- *Git and GitHub for Biostatistics* (https://git-bootcamp.rgtlab.org)
- *Biostatistics Practicum* (https://rgtlab.org/practicum)
- *Reproducible Research for the Health Sciences* (https://reproducibility.rgtlab.org)

The end of Chapter 5 describes how the publishing material here connects to *Git and GitHub for Biostatistics* and how the reproducibility thread continues into *Reproducible Research for the Health Sciences*.

Part I.

Five Days

Draft, November 13, 2026. Content may change.

1. Day 1: From R Markdown to Quarto, Your First Render

1.1. Learning objectives

By the end of this day you should be able to:

- Explain why an R Markdown user might move to Quarto, and what does and does not transfer.
- Install Quarto and confirm the installation with `quarto --version` and `quarto check`.
- Describe how a `.qmd` file relates to a `.Rmd` file at the level of syntax and engine.
- Render an existing `.Rmd` document with Quarto, unchanged.
- Choose among the RStudio, Positron, VS Code, Vim, and terminal workflows for editing and rendering.
- Recognize the anatomy of a single-document Quarto project.

1.2. Lecture

We begin with the question that motivates the week. Why should an analyst who is already productive in R Markdown move to Quarto at all? We answer it, install the tool, and render a document we already have.

1.2.1. Why transition, and what carries over

Quarto is not a rewrite of R Markdown so much as a generalization of it. It keeps the knitr engine and the Pandoc back end, so the mental model the reader already holds still applies. That is, a `.qmd` document is read by the

Draft, November 13, 2026. Content may change.

same knitr that reads an `.Rmd`, and the same Pandoc that produced the reader's HTML and PDF still produces Quarto's. We note that most of an existing `.Rmd` renders under Quarto with no change at all. What Quarto adds is a uniform command-line interface, native cross-references, and one project model for many output types.

1.2.2. Installing Quarto and the command-line interface

Quarto is a standalone program, not an R package. We install it once from https://quarto.org and drive it from a terminal.

```
# macOS with Homebrew; or download the installer from
# https://quarto.org/docs/get-started
brew install --cask quarto
```

Confirm the installation and inspect the environment Quarto sees:

```
quarto --version
quarto check
```

We note that `quarto check` reports the R, knitr, and (if present) Python and Jupyter installations Quarto will use. It is the first command to run when a render misbehaves.

1.2.3. The .qmd format compared with .Rmd

A `.qmd` file looks almost identical to a `.Rmd` file: YAML front matter, prose in Markdown, and executable code chunks. The visible differences are small on Day 1 and are the subject of Day 2. Recall that in R Markdown, chunk options sit on the chunk header line; in Quarto they move inside the chunk on `#|` comment lines.

```
---
title: 'Baseline summary'
format: html
---
```


Draft, November 13, 2026. Content may change.

```
summary(cars)
```

1.2.4. Rendering an existing .Rmd unchanged

The most reassuring first exercise is to render a document the reader already owns. Quarto renders `.Rmd` files directly.

```
quarto render existing-report.Rmd
```

We observe that the output appears beside the source, as knitr would have produced it. That is the point of Day 1: the reader's existing work is already most of the way to Quarto.

1.2.5. RStudio, Positron, VS Code, Vim, and terminal workflows

Quarto is editor-agnostic. We describe five workflows and let the reader choose:

- **RStudio** renders on the Knit or Render button and offers a visual editor for `.qmd`.
- **Positron**, Posit's next-generation data science IDE, renders `.qmd` with a Render button and visual editor much as RStudio does, and adds first-class support for R and Python in a single session. Now that it has reached general availability, it is a first-class option, and it suits readers who expect to work across both languages (the subject of the companion volume *Python for R Programmers*).
- **VS Code** with the Quarto extension gives a live preview and is a common choice for Python-inclusive projects.
- **Vim**, through the `zzvim-R` plugin (part of the rgtlab software suite, https://github.com/rgt47/zzvim-r), sends lines and code chunks from a `.qmd` buffer to a live R console, which suits readers who already edit in the terminal.
- **The terminal** with `quarto render` and `quarto preview` is editor-independent and is what this book shows.

Draft, November 13, 2026. Content may change.

```
quarto preview report.qmd
```

1.2.6. Project anatomy

A single document needs no project, but understanding the minimal project anatomy now pays off on Day 4. The pieces are the `.qmd` source, an optional `_quarto.yml` configuration, and the rendered output directory.

```
report/
  report.qmd        # source
  _quarto.yml       # optional project configuration
  report.html       # rendered output
```

1.3. Worked example: rendering a colleague's report under Quarto

We consider a concrete scenario. A colleague hands us `enrollment-report.Rmd`, a knitr document that reads a CSV, prints a summary table, and draws one figure. Our task is to confirm it renders under Quarto and then save it as a `.qmd` so that later days can extend it.

We begin by rendering the file as received:

```
quarto render enrollment-report.Rmd
```

The render succeeds and writes `enrollment-report.html`. We note that no source change was required, which confirms the claim from the lecture. Next we copy the file to a `.qmd` extension and open it:

```
cp enrollment-report.Rmd enrollment-report.qmd
quarto preview enrollment-report.qmd
```

The preview opens in the browser and refreshes as we save. We have not yet changed the chunk-option syntax, and we do not need to; the `.Rmd`-style options still work. That translation is Day 2's work. For now, the

Draft, November 13, 2026. Content may change.

colleague's report is a Quarto document that renders and previews, and we have lost nothing in the move.

1.4. Homework

Attempt each problem in your own environment before checking the solution. Work in a scratch directory so that you can experiment freely.

1. **Confirm your installation.** Run the commands that print the Quarto version and check the environment. Record the Quarto version and the R and knitr versions Quarto reports.

2. **Render a minimal document.** Create a `hello.qmd` with a title, one sentence of prose, and one R chunk that prints `summary(cars)`. Render it to HTML and confirm the output file appears.

3. **Render an existing .Rmd.** Take any `.Rmd` you already have (or write a two-chunk one) and render it with `quarto render`. Did it require any change? Note anything that differed from a knitr render.

4. **Preview and live reload.** Run `quarto preview` on your `hello.qmd`, edit the prose, and save. Describe what happens in the browser.

5. **Compare the formats.** Render `hello.qmd` to both HTML and, if you have a LaTeX installation, PDF. Name the two output files that result.

6. **Read quarto check.** Run `quarto check` and read its output. Which engines does it report as available, and which (if any) does it report as missing?

1.5. Solutions

Problem 1. Placeholder. Show `quarto --version` and `quarto check` output and identify the reported R and knitr versions.

Draft, November 13, 2026. Content may change.

Problem 2. Placeholder. Give the minimal `hello.qmd` source and the `quarto render hello.qmd` invocation, and note that `hello.html` results.

Problem 3. Placeholder. Confirm the `.Rmd` rendered with no change and comment on the near-identical output.

Problem 4. Placeholder. Describe the automatic browser refresh produced by `quarto preview` on save.

Problem 5. Placeholder. Show rendering to `html` and `pdf` and name `hello.html` and `hello.pdf`.

Problem 6. Placeholder. Interpret the `quarto check` report of available and missing engines.

1.6. What's next

Day 2 goes inside the document. We shall translate the R Markdown constructs that do differ under Quarto: the YAML front matter, the `#|` cell-option syntax that replaces `knitr::opts_chunk`, the execution options that control what runs and what shows, and the choice between the knitr and jupyter engines. Keep the `enrollment-report.qmd` from the worked example; Day 2 begins by tidying its chunk options.

Draft, November 13, 2026. Content may change.

2. Day 2: Document Anatomy: YAML, Chunks, and the Execution Model

2.1. Learning objectives

By the end of this day you should be able to:

- Translate an R Markdown YAML header into its Quarto form.
- Rewrite chunk options from the header line into the `#|` cell-option syntax.
- Set document-wide execution defaults in place of `knitr::opts_chunk$set`.
- Control execution with `echo`, `eval`, `warning`, `message`, and `freeze`.
- Distinguish the knitr and jupyter engines and state when each applies.
- Use inline code in the Quarto style.

2.2. Lecture

We begin inside the document. Day 1 showed that an `.Rmd` renders unchanged; today we make the small but consequential changes that turn an `.Rmd` into an idiomatic `.qmd`. That is, we translate the anatomy.

2.2.1. YAML front matter differences

The Quarto YAML header is close to the R Markdown one, with a few renamings. Recall that R Markdown uses `output:` with a package-format key; Quarto uses `format:` with a short name.

Draft, November 13, 2026. Content may change.

```yaml
# R Markdown
output:
  html_document:
    toc: true
```

```yaml
# Quarto
format:
  html:
    toc: true
```

We note that many sub-options keep their names, so the translation is often mechanical.

2.2.2. The #| cell-option syntax

The most visible change is where chunk options live. In R Markdown they sit on the header line; in Quarto they move onto `#|` comment lines inside the chunk, written as YAML.

We observe that each option is a YAML key, lowercased and hyphenated (`fig-cap`, not `fig.cap`). This is the single most common edit in a migration, and Day 5 automates it.

2.2.3. Execution options in place of opts_chunk

Where R Markdown sets global chunk defaults with `knitr::opts_chunk$set(...)` in a setup chunk, Quarto sets them once in the YAML under `execute:`.

```yaml
execute:
  echo: true
  warning: false
  message: false
```

We note that a per-chunk `#|` option overrides the document-wide default, exactly as a per-chunk option overrode `opts_chunk` in R Markdown.

Draft, November 13, 2026. Content may change.

2.2.4. Controlling what runs and what shows

Five options carry most of the daily load. We describe each briefly:

- `eval` decides whether the code runs.
- `echo` decides whether the code is shown.
- `warning` and `message` decide whether those streams appear.
- `freeze` decides whether Quarto re-executes a cell at all, which we develop fully on Day 5.

```
# shown but not run
fit <- lm(dist ~ speed, data = cars)
```

2.2.5. The knitr and jupyter engines

Quarto chooses an engine per document. For R content it uses knitr, the same engine R Markdown used, so behavior matches. For Python content it can use jupyter. We set the engine explicitly when needed.

```
engine: knitr
```

We note that mixing R and Python in one document is possible. The everyday R Markdown user will stay on knitr, however, and nothing changes for them.

2.2.6. Inline code

Inline code carries over directly. Recall the R Markdown form `50`; Quarto uses the identical syntax, so inline expressions need no translation.

```
The dataset has 50 rows.
```


Draft, November 13, 2026. Content may change.

2.3. Worked example: modernizing a setup chunk

We return to `enrollment-report.qmd` from Day 1. It still carries an R Markdown setup chunk and header-line options. Our task is to translate them into idiomatic Quarto without changing the rendered result.

The document opens with a familiar setup chunk:

```
knitr::opts_chunk$set(echo = FALSE, warning = FALSE,
                      message = FALSE)
```

We lift those defaults into the YAML and delete the setup chunk:

```
execute:
  echo: false
  warning: false
  message: false
```

Next we find a figure chunk written the R Markdown way:

````
::: {.cell}

:::


and rewrite it in the Quarto way:


::: {.cell}

```{.r .cell-code}
barplot(table(c('A', 'B', 'A')))
```

:::
````

We render again and compare against Day 1's output. The result is identical, which is the goal: the translation changes the source style, not the rendered document. We have now removed the last R Markdown idioms from the report, and it is ready for the output features of Day 3.

Draft, November 13, 2026. Content may change.

2.4. Homework

Attempt each problem in your own environment before checking the solution.

1. **Translate a header.** Given an R Markdown header that uses `output: html_document` with a table of contents, write the equivalent Quarto `format:` block.

2. **Move options inside.** Take a chunk written as ```` ```{r, echo=FALSE, fig.width=6} ```` and rewrite it with the `#|` cell-option syntax.

3. **Global defaults.** Replace a `knitr::opts_chunk$set(echo = FALSE, message = FALSE)` setup chunk with the equivalent `execute:` YAML.

4. **eval versus echo.** Write two chunks: one shown but not run, and one run but not shown. State which option controls each.

5. **Override a default.** With document-wide `echo: false`, write a single chunk that nonetheless shows its code.

6. **Inline code.** Write a sentence that reports the number of rows of `cars` using inline code.

7. **Name the engine.** State which engine Quarto uses for an R-only document, and where you would set it explicitly.

2.5. Solutions

Problem 1. Placeholder. Give the `format: html` block with `toc: true`.

Problem 2. Placeholder. Show the chunk rewritten with `#| echo: false` and `#| fig-width: 6`.

Problem 3. Placeholder. Show the `execute:` block with `echo: false` and `message: false`.

Draft, November 13, 2026. Content may change.

Problem 4. Placeholder. Identify `eval: false` for shown but not run, and `echo: false` for run but not shown.

Problem 5. Placeholder. Show a chunk carrying `#| echo: true` that overrides the global default.

Problem 6. Placeholder. Show the inline `50` sentence.

Problem 7. Placeholder. State knitr for R-only content and the `engine: knitr` YAML key.

2.6. What's next

Day 3 turns from anatomy to output. We shall render the same content to HTML, PDF, Word, and revealjs slides; handle figures and tables with captions; use Quarto's native cross-references for figures, tables, and sections; add citations with a CSL style; and place content in callouts and the margin. Keep `enrollment-report.qmd`; Day 3 adds cross-referenced figures and tables to it.

Draft, November 13, 2026. Content may change.

3. Day 3: Output Formats, Figures, Tables, and Cross-References

3.1. Learning objectives

By the end of this day you should be able to:

- Render one source document to HTML, PDF, Word, and revealjs.
- Attach captions to figures and tables and control their placement.
- Cross-reference figures, tables, and sections natively with `@fig-`, `@tbl-`, and `@sec-`.
- Add citations from a `.bib` file and select a CSL style.
- Place content in callouts and in the margin.
- Apply a theme to an HTML document.

3.2. Lecture

We begin with what R Markdown made awkward and Quarto makes uniform: producing many output formats from one source, and cross-referencing within them. That is, we write once and target several outputs.

3.2.1. One source, several formats

Quarto renders to a format named in the YAML or on the command line. We list several under `format:` and select one at render time.

Draft, November 13, 2026. Content may change.

Table 3.1.: Summary of the cars dataset

```
knitr::kable(summary(cars))
```

```
format:
  html: default
  pdf: default
  docx: default
  revealjs: default
```

```
quarto render report.qmd --to pdf
quarto render report.qmd --to revealjs
```

We note that `revealjs` produces HTML slides from the same prose and chunks, replacing the separate `xaringan` toolchain an R Markdown user may have learned.

3.2.2. Figures and captions

A figure gets a caption and a cross-reference label through chunk options. Recall that the label must begin with `fig-` for Quarto to treat it as a figure.

```
plot(cars)
```

3.2.3. Tables and captions

Tables work the same way with a `tbl-` label. We produce a table with any R table package and add a caption.

Draft, November 13, 2026. Content may change.

rgtlab tools for tables and figures

Two packages in the rgtlab software suite address exactly this output-format problem. `zztable1` (https://github.com/rgt47/zztable1) produces publication-quality summary tables and detects the output format automatically, emitting LaTeX under the PDF format and HTML under the web format from a single call. `zzobj2fig` (https://github.com/rgt47/zzobj2fig) turns data frames and fitted model objects into journal-styled LaTeX tables and cropped PDF figures. Both drop into a `.qmd` code chunk unchanged, so the one-source-several-formats promise of this chapter extends to the tables as well as the prose.

3.2.4. Native cross-references

Here is the capability that most repays the transition. We reference a labeled figure, table, or section by an `@` prefix, and Quarto resolves it to a numbered, clickable link in every format.

```
As @fig-speed shows, distance rises with speed; the
values appear in @tbl-summary. See also @sec-day3.
```

We observe that the same syntax works in HTML and PDF, which in R Markdown required a package and format-specific care.

3.2.5. Citations and CSL

Citations carry over from R Markdown with the same `@key` syntax against a `.bib` file. We select a citation style with a CSL file.

```
bibliography: references.bib
csl: https://www.zotero.org/styles/apa
```

```
Quarto builds on Pandoc and knitr [@xie2015knitr].
```


Draft, November 13, 2026. Content may change.

3.2.6. Callouts and margin content

Quarto adds semantic callouts and margin placement that R Markdown lacked. We note the callout block syntax and the margin-column attribute.

```
::: {.callout-note}
A short aside.
:::

::: {.column-margin}
A note set in the margin.
:::
```

3.2.7. Theming

An HTML document takes a Bootswatch theme and an optional `.scss` file, the same mechanism this book uses.

```
format:
  html:
    theme: cosmo
```

3.3. Worked example: a cross-referenced results section

We extend `enrollment-report.qmd` into a short results section with one figure and one table, each cross-referenced, and one citation. Our task is to produce a document whose figure and table numbers and references stay consistent across HTML and PDF.

We label the figure and table:

```
barplot(table(c('A', 'A', 'B')))
```

We then write prose that references both and cites a source:

Draft, November 13, 2026. Content may change.

Table 3.2.: Baseline characteristics by arm

```
knitr::kable(head(cars, 3))
```

```
@fig-enroll-arm shows enrollment by arm, and
@tbl-baseline reports baseline characteristics. The
reporting approach follows established practice
[@xie2020cookbook].
```

We render to HTML and to PDF:

```
quarto render enrollment-report.qmd --to html
quarto render enrollment-report.qmd --to pdf
```

We observe that the figure is numbered and the reference reads ‘Figure 1’ in both outputs, with the numbering managed by Quarto rather than by hand. The citation resolves against `references.bib` and appears in the reference list in the selected APA style. The report now carries the output features that R Markdown provided only through several separate packages.

3.4. Homework

Attempt each problem in your own environment before checking the solution.

1. **Two formats.** Render a document to both HTML and Word. Name the two output files and note any content that rendered differently.
2. **Caption a figure.** Add a labeled, captioned figure to a document and confirm the caption appears beneath it.
3. **Cross-reference it.** Reference the figure from Problem 2 in prose with `@fig-...` and confirm the rendered link reads ‘Figure 1’.
4. **Caption and reference a table.** Add a captioned table with a `tbl-` label and reference it in prose.

5. **Add a citation.** Create a two-entry `references.bib`, cite one entry with `@key`, and render. Confirm the reference list appears.
6. **Make a callout.** Add a `callout-tip` block and a margin note to a document and render both.
7. **Slides from prose.** Render an existing short document to `revealjs` and describe how the section headings map to slides.

3.5. Solutions

Problem 1. Placeholder. Name `report.html` and `report.docx` and comment on format differences.

Problem 2. Placeholder. Show the `#| label: fig-...` and `#| fig-cap:` options and confirm the caption.

Problem 3. Placeholder. Show the `@fig-...` reference resolving to 'Figure 1'.

Problem 4. Placeholder. Show the `tbl-` label, `tbl-cap`, and the `@tbl-...` reference.

Problem 5. Placeholder. Give the two-entry `.bib`, the `@key` citation, and confirm the reference list.

Problem 6. Placeholder. Show the `callout-tip` and `column-margin` blocks.

Problem 7. Placeholder. Describe headings becoming slide boundaries under `revealjs`.

3.6. What's next

Day 4 scales up from one document to a project. We shall configure a project with `_quarto.yml`, build a website and a book, use the manuscript project type, parameterize a report, and factor shared content with includes, shortcodes, and profiles. Keep `enrollment-report.qmd`; Day 4 folds it into a small project.

Draft, November 13, 2026. Content may change.

4. Day 4: Projects at Scale: Websites, Books, and Manuscripts

4.1. Learning objectives

By the end of this day you should be able to:

- Configure a multi-document Quarto project with `_quarto.yml`.
- Build a website project and a book project from a set of `.qmd` files.
- Use the manuscript project type for a scholarly article.
- Parameterize a report and render it with different parameters.
- Factor shared content with includes and shortcodes.
- Select build variants with profiles.

4.2. Lecture

We begin where R Markdown's separate packages diverged. Producing a website meant `blogdown`, a book meant `bookdown`, and an article meant a journal template, each with its own configuration. Quarto unifies these under one project model. That is, one `_quarto.yml` and one command build any of them.

4.2.1. The project and `_quarto.yml`

A project is a directory with a `_quarto.yml` file that names a type and collects shared settings. We note that render then operates on the whole directory rather than a single file.

Draft, November 13, 2026. Content may change.

```
project:
  type: website
```

```
quarto render        # builds the whole project
quarto preview       # serves it with live reload
```

4.2.2. Websites

A website project lists navigation and pages. Recall that `blogdown` required Hugo; Quarto needs no external site generator.

```
project:
  type: website
website:
  title: 'Analysis notes'
  navbar:
    left:
      - href: index.qmd
        text: Home
      - about.qmd
```

4.2.3. Books

A book project collects chapters and produces HTML and PDF, as this volume does. We note that the structure matches what the reader is reading.

```
project:
  type: book
book:
  title: 'A short book'
  chapters:
    - index.qmd
    - chapter1.qmd
    - chapter2.qmd
```


Draft, November 13, 2026. Content may change.

4.2.4. The manuscript project type

For a scholarly article, the manuscript project type pairs a readable article with its computational notebook. We note that it targets journal submission with figures, tables, and cross-references in one source.

```yaml
project:
  type: manuscript
manuscript:
  article: article.qmd
```

4.2.5. Parameterized reports

Quarto parameterizes a report through YAML `params`, the same idea R Markdown offered, and renders with overrides on the command line.

```yaml
params:
  site: 'overall'
```

```
quarto render report.qmd -P site:north
```

4.2.6. Includes and shortcodes

Shared prose is factored into a file and pulled in with an include shortcode, so that boilerplate lives in one place.

```
{{< include _disclaimer.qmd >}}

{{< meta title >}}
```

4.2.7. Profiles

A profile selects a named set of options, so that one project can build a draft and a final variant without editing the YAML.

```
quarto render --profile draft
```

4.3. Worked example: a two-page analysis website

We fold `enrollment-report.qmd` into a small website with a landing page and the report, sharing one disclaimer across both. Our task is to build and preview a project rather than a single file.

We create a `_quarto.yml` that declares a website and its navigation:

```
project:
  type: website
website:
  title: 'Enrollment analysis'
  navbar:
    left:
      - href: index.qmd
        text: Home
      - href: enrollment-report.qmd
        text: Report
```

We factor the shared disclaimer into `_disclaimer.qmd` and include it on both pages:

```
{{< include _disclaimer.qmd >}}
```

We then build and preview the whole project:

```
quarto preview
```

We observe that both pages share the navbar and the disclaimer, and that editing the disclaimer once updates both. The report we wrote across Days 1 through 3 is now one page of a coherent site, built by the same `quarto` command that rendered it as a single file. We have moved from a document to a project without changing the report itself.

Draft, November 13, 2026. Content may change.

4.4. Homework

Attempt each problem in your own environment before checking the solution.

1. **A minimal project.** Create a directory with a `_quarto.yml` declaring a website and two `.qmd` pages. Render the project and name the output directory.
2. **Navigation.** Add a navbar with links to both pages and confirm the links work in the preview.
3. **A book.** Convert the same two pages into a book project with a chapter list. What changes in `_quarto.yml`?
4. **A parameter.** Add a `params:` entry to a report and render it twice with different values using `-P`. Describe what differs in the output.
5. **An include.** Factor a shared paragraph into a partial and include it on two pages with the include shortcode.
6. **A profile.** Define a `draft` profile and render with `--profile draft`. State one option you varied between default and draft.
7. **Manuscript.** Initialize a manuscript project and name the file that holds the article.

4.5. Solutions

Problem 1. Placeholder. Show the `_quarto.yml` website block and name the `_site` output directory.

Problem 2. Placeholder. Show the navbar block and confirm the links.

Problem 3. Placeholder. Show `type: book` with a `chapters:` list replacing the website block.

Problem 4. Placeholder. Show `params:` and two `-P` renders and describe the differing output.

Problem 5. Placeholder. Show the partial file and the `{{< include >}}` shortcode.

Draft, November 13, 2026. Content may change.

Problem 6. Placeholder. Show a `draft` profile file and the option it varies.

Problem 7. Placeholder. Show `type: manuscript` and name `article.qmd`.

4.6. What's next

Day 5 closes the loop from source to published, reproducible output. We shall use freeze and caching to stabilize a build, integrate renv for dependency pinning, publish with `quarto publish` to Netlify, GitHub Pages, and Quarto Pub, wire continuous integration with GitHub Actions, and apply a systematic `.Rmd`-to-`.qmd` migration checklist to a whole project. Keep the website from the worked example; Day 5 publishes it.

Draft, November 13, 2026. Content may change.

5. Day 5: Reproducibility, Publishing, and Migrating a Real Project

5.1. Learning objectives

By the end of this day you should be able to:

- Use freeze to avoid re-executing unchanged computations.
- Integrate renv so that a project's package versions are pinned.
- Publish a project with `quarto publish` to Netlify, GitHub Pages, and Quarto Pub.
- Configure continuous integration with GitHub Actions.
- Apply a systematic checklist to migrate an `.Rmd` project to `.qmd`.
- Judge when a migration is complete and reproducible.

5.2. Lecture

We begin at the end of the pipeline: making a build stable, publishing it, and keeping it reproducible. That is, we turn a project that renders on our machine into one that renders the same way elsewhere and appears on the web.

5.2.1. Freeze and caching

Freeze records the results of computations so that Quarto does not re-execute unchanged cells on every render. Recall that R Markdown offered per-chunk `cache = TRUE`; Quarto adds project-level `freeze`, which is what a website or book needs for a fast, deterministic build.

Draft, November 13, 2026. Content may change.

```
execute:
  freeze: auto
```

We note that `freeze: auto` re-executes a document only when its source changes, and that the frozen results are committed to version control so that a continuous-integration build need not run R at all.

5.2.2. renv integration

Reproducibility of results requires reproducibility of packages. We pin package versions with renv and restore them in any environment.

```
renv::init()
renv::snapshot()
renv::restore()
```

We observe that `renv.lock` records exact versions, so a collaborator or a CI runner rebuilds the same environment. The *Reproducible Research for the Health Sciences* volume develops this in depth.

> rgtlab tools for reproducible environments
>
> Two rgtlab tools formalize this step. `zzcollab` (https://github.com/rgt47/zzcollab) wraps Docker and renv into a reproducible research compendium. The `zzc` command builds the container in which the book renders, and `make check-renv` validates the lockfile against the source. `zzrenvcheck` (https://github.com/rgt47/zzrenvcheck) checks that every package the source actually uses is both declared and locked. This catches the silent dependency that renders on the author's machine but fails in continuous integration.

5.2.3. Publishing

Quarto publishes to several targets with one command. We show the three the reader is most likely to use.

Draft, November 13, 2026. Content may change.

```
quarto publish quarto-pub
quarto publish gh-pages
quarto publish netlify
```

We note that `quarto publish` renders, uploads, and records the target in `_publish.yml` so that a later publish reuses the same destination.

5.2.4. Continuous integration with GitHub Actions

For a project under version control, a GitHub Actions workflow renders and deploys on every push, which is how this book is built. We note that with freeze committed, the workflow need not install R packages.

```
# .github/workflows/publish.yml
on:
  push:
    branches: [main]
jobs:
  publish:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: quarto-dev/quarto-actions/setup@v2
      - run: quarto render
```

We note that the Git and GitHub mechanics this assumes are the subject of the companion *Git and GitHub for Biostatistics* volume. Here we use them without teaching them.

When the reader commits and publishes by hand, the `zzgit` tool (https://github.com/rgt47/zzgit) offers an interactive add, commit, and push. It first scans the staged diff for secrets with gitleaks. This guards against publishing a credential by accident along with a rendered site.

Draft, November 13, 2026. Content may change.

5.2.5. A migration checklist

We close the week with the systematic procedure for migrating an existing R Markdown project. We apply it as a checklist rather than an ad hoc edit:

1. Rename `.Rmd` files to `.qmd`.
2. Convert `output:` YAML to `format:`.
3. Move header-line chunk options to `#|` cell options.
4. Replace `knitr::opts_chunk$set` with an `execute:` block.
5. Convert `bookdown`-style cross-references to `@fig-`, `@tbl-`, and `@sec-`.
6. Add a `_quarto.yml` if the project has more than one document.
7. Set `freeze: auto` and pin packages with renv.
8. Render, compare against the old output, and publish.

5.3. Worked example: migrating a small project end to end

We migrate a two-document R Markdown project, an analysis and a README site, into a published Quarto website, applying the checklist in order. Our task is to end with a reproducible, published site whose output matches the original.

We rename the sources and convert their headers:

```
git mv analysis.Rmd analysis.qmd
git mv index.Rmd index.qmd
```

We move each chunk's options inside with the `#|` syntax and lift the shared setup chunk into `_quarto.yml`:

```
execute:
  echo: false
  freeze: auto
```

We pin the environment and render:

Draft, November 13, 2026. Content may change.

```
Rscript -e 'renv::snapshot()'
quarto render
```

We compare the rendered pages against the original HTML and confirm that the tables and figures match. Finally we publish:

```
quarto publish netlify
```

We observe that the site is now live, its computations are frozen and committed, and its package versions are pinned in `renv.lock`. A CI runner or a colleague can reproduce the same site from the repository alone. The project that began as two `.Rmd` files is now a reproducible, published Quarto site, and the migration followed a checklist rather than guesswork.

5.4. Homework

Attempt each problem in your own environment before checking the solution.

1. **Turn on freeze.** Add `freeze: auto` to a project, render twice, and describe what the second render skips.
2. **Pin packages.** Run `renv::init()` and `renv::snapshot()` in a project and name the file that records the versions.
3. **Publish to Quarto Pub.** Publish a single document with `quarto publish quarto-pub` and record the resulting URL.
4. **Publish a site.** Publish the Day 4 website to a target of your choice and confirm it loads.
5. **A CI workflow.** Write a minimal GitHub Actions workflow that renders the project on push. Which two actions does it use?
6. **Apply the checklist.** Take a real one-file `.Rmd` and migrate it to `.qmd` following the eight-step checklist. Record which steps applied and which did not.

Draft, November 13, 2026. Content may change.

7. **Verify reproducibility.** Delete the output directory, render from a clean state, and confirm the output matches. What role did freeze and renv play?

5.5. Solutions

Problem 1. Placeholder. Describe the second render reusing frozen results.

Problem 2. Placeholder. Name `renv.lock`.

Problem 3. Placeholder. Show `quarto publish quarto-pub` and record the URL.

Problem 4. Placeholder. Show the publish command and confirm the site loads.

Problem 5. Placeholder. Identify the checkout and quarto-actions setup steps.

Problem 6. Placeholder. Record which of the eight steps applied to the chosen document.

Problem 7. Placeholder. Confirm the clean render matches and attribute it to freeze and renv.

5.6. What's next

The week is complete. The reader can render, extend, and publish Quarto documents and migrate an existing R Markdown project. From here, the reproducibility thread continues into the companion *Reproducible Research for the Health Sciences* volume, which develops renv, Docker, and validated pipelines. The version-control mechanics this chapter assumed are, in turn, the subject of *Git and GitHub for Biostatistics*. We began the week rendering a colleague's `.Rmd` unchanged. We end it publishing a reproducible Quarto project of our own.

Draft, November 13, 2026. Content may change.

References

Credits

The chapter list was constructed by surveying publicly available Quarto migration guides and teaching materials, including the official Quarto documentation at https://quarto.org, the R Markdown to Quarto transition notes maintained by the Quarto project, and course modules that adopt Quarto for reproducible reporting, and adopting the topics common to the majority of them.

Cover artwork generated procedurally by the R script at `images/build-cover.R`. Watercolor palette: deep slate through steel and sky to pale mist, anchored on a brand blue `#39729e` chosen to echo the Quarto project's signature blue. Typography: Avenir family (Apple system font); body text in Source Serif 4.

Colophon

This book was produced with Quarto, typeset in Source Serif 4, with code blocks set in JetBrains Mono. The command examples target Quarto 1.5 or later and a recent R installation with the `rmarkdown` and `knitr` packages.

The book is hosted at https://rmd-to-quarto.rgtlab.org on Netlify, with continuous deployment via GitHub Actions from the `rgt47/rmd-to-quarto` repository on every push to `main`.

The cover is generated procedurally by `images/build-cover.R` from a watercolor gradient anchored on `#39729e` (the volume's brand blue) and overlaid with Avenir typography. Re-running the script regenerates the cover; the procedural approach makes the cover reproducible and editable in version control rather than dependent on a one-shot image-generation step. That the cover of a book about reproducible authoring is itself a versioned, scripted artifact is deliberate.

Last rendered: see the build timestamp on the homepage.