Workbench Developer's Guide

Zhian N. Kamvar

2023-12-31

Table of contents

Preface

The Carpentries Workbench is a open-source and portable lesson infrastructure built with the R programming language. Despite it being built in R, contributors do not need to know any R in order to use it to build reliable, stylish, and accessible lessons.

This book serves as development documentation for The Carpentries Workbench. It was written between June and December 2023 primarily to orient new developers and contributors to The Workbench ecosystem.

> **Prerequisite**
>
> This book assumes familiarity with R Package Development. If you are unfamiliar, please read R Packages (2e) (Wickham and Bryan 2023).

1 Introduction

The core of The Carpentries Workbench consists of three packages:

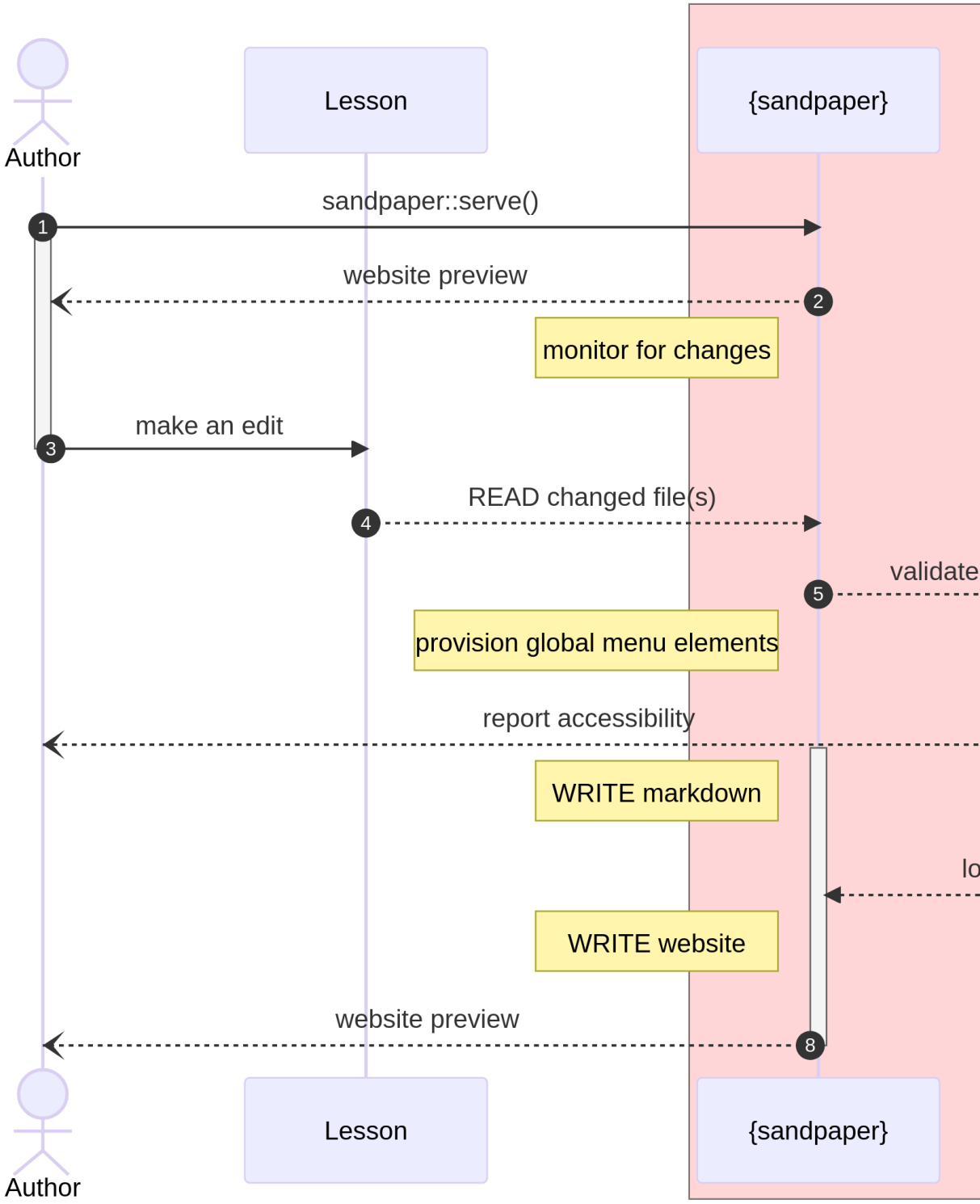
- {sandpaper}: user interface and workflow engine
- {pegboard}: parsing and validation engine
- {varnish}: HTML templates, CSS, and JS elements

These packages are all available and released to the Carpentries R-Universe, which checks for updates to the source packages hourly.

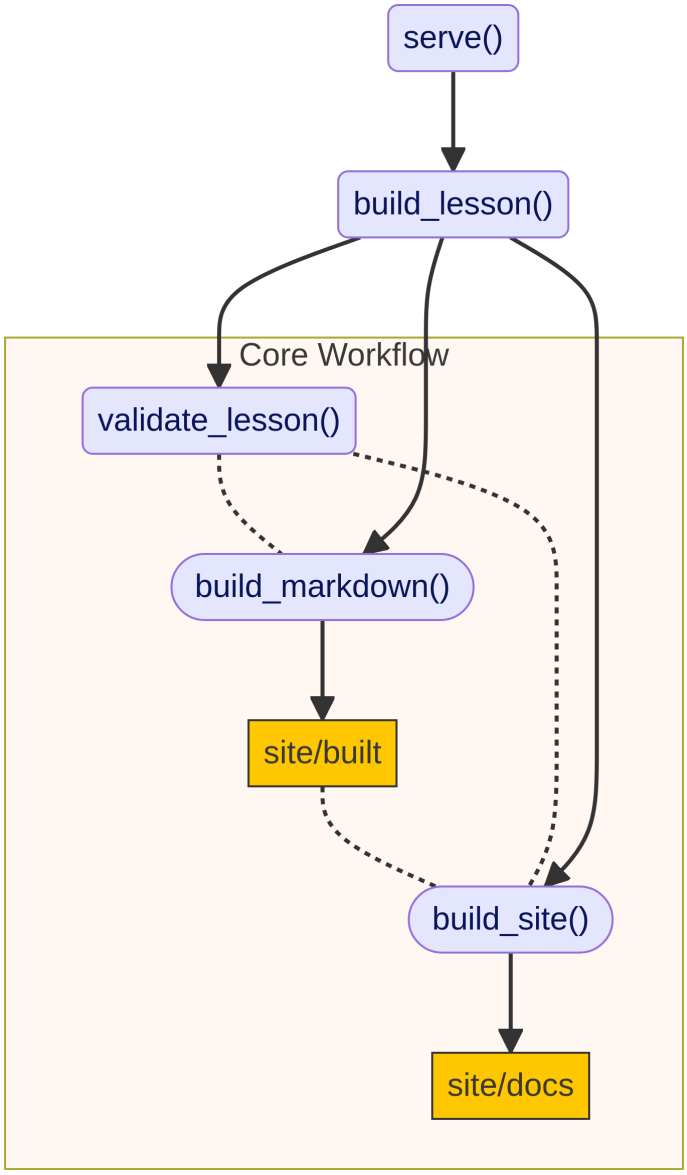
1.1 Building Lessons

In a broad sense, this is what happens when you run `sandpaper::serve()` or `sandpaper::build_lesson()`. The interaction between the three Workbench packages, the lesson content, and the author can be summarised like this where the author makes an edit:

> **Summary Content**
>
> This content is a general picture of what happens between the packages. For a more in-depth discussion and more detailed diagrams, please visit the Flow Diagrams page.


Author
Lesson
{sandpaper}
sandpaper::serve()
1
website preview
2
monitor for changes
make an edit
3
READ changed file(s)
4
validate
5
provision global menu elements
report accessibility
WRITE markdown
lo
WRITE website
website preview
8
Author
Lesson
{sandpaper}

In terms of folder structure, the workflow runs the two-step workflow to first render markdown files into `site/built` and then uses those files to render the HTML, CSS, and JavaScript into `site/built`. These workflows are detailed in The Workflows Chapter.

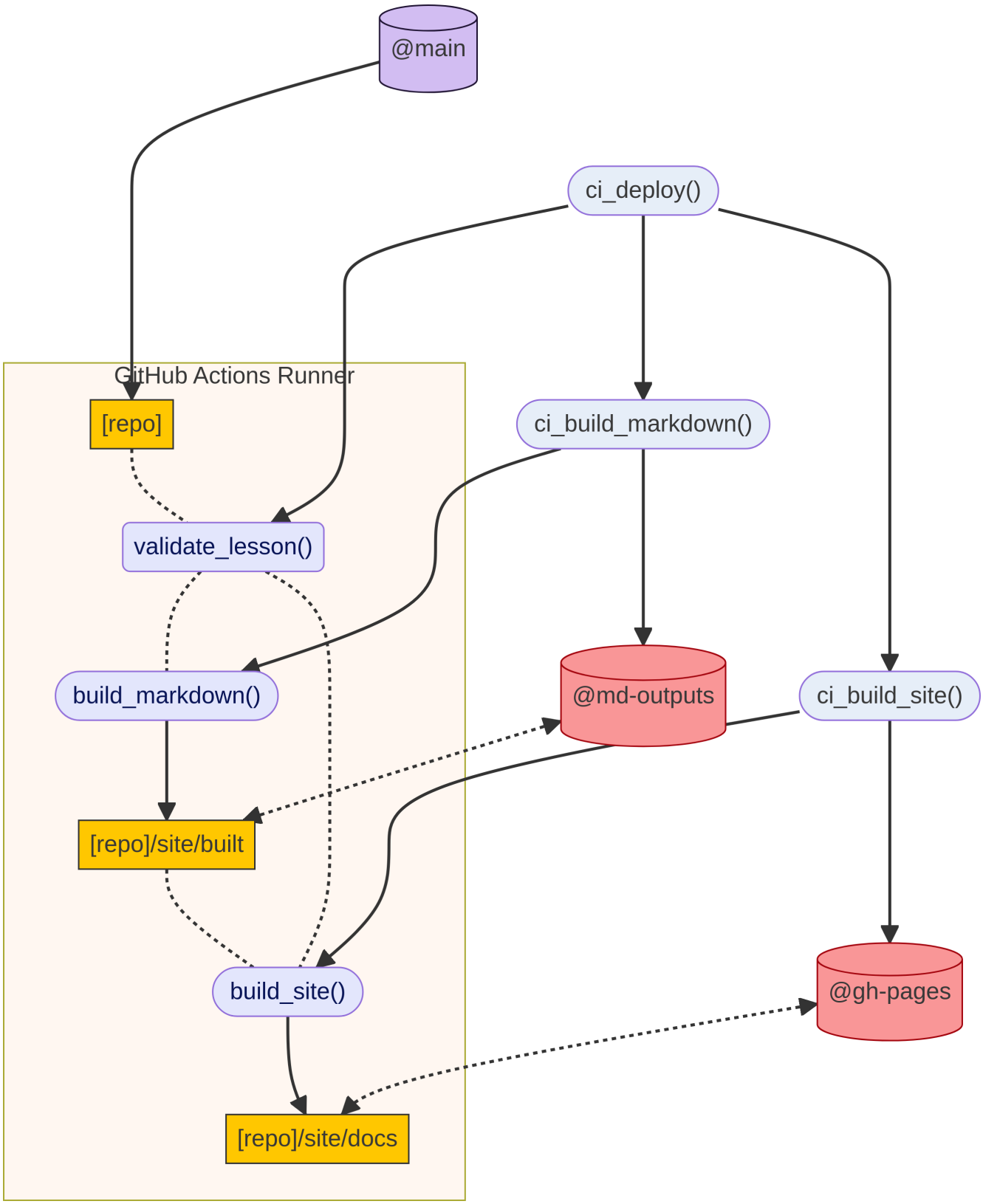


> **Resource folder names**
>
> The names of the folders inside `site/` are considered internal resources and they can change at any time. The reason why the folder for the final website output is called `site/docs/` is because we use the {pkgdown} package to provision the website without needing to bundle the templates inside of {sandpaper}, but we never got around to explicitly changing the name of that folder.

The `site/docs` folder contains the full website that can be safely used offline. This is the core of the workflow and is used both locally and in a remote setting. The only difference with the remote setting is that we use a few Git tricks to provision the markdown cache without needing to store it in the default branch.

1.2 Building Lessons on GitHub

In the remote workflow, **we still use the same workflow as above**, except now we use `ci_deploy()` to link the branches and folders using worktrees, which you can think of as Git branches assigned to separate folders.


@main
ci_deploy()
GitHub Actions Runner
[repo]
ci_build_markdown()
validate_lesson()
build_markdown()
@md-outputs
ci_build_site()
[repo]/site/built
build_site()
@gh-pages
[repo]/site/docs

1.3 Development

Development of The Workbench is overseen by Zhian N. Kamvar. New features are added incrementally as pull requests. Pushes to the main branch are *rare* and discouraged. New features must have tests associated (with the exception of {varnish}).

If you are interested, we have documentation for the release process available.

1.4 Documentation

Reference documentation for individual functions for each package is written alongside the function using {roxygen2}.

This documentation is generated by `devtools::document()`

1.5 Testing

Tests for each package live in `tests/testthat/` and follow a `test-[file-name].R` naming convention. These are controlled by the {testthat} package and run by `devtools::test()`.

You can find more information about testing the core packages in Testing The Workbench

1.6 Continous Integration

The continous integration for each package tests on Ubuntu, MacOS, and Windows systems with the last five versions of R (same as the RStudio convention).

More information about the Continous Integration can be found in the Continuous Integration section of the testing section.

Coming up:

- Testing Pull Requests (Locally and on your fork)
- Resources for R package development
- Adding functionality to {sandpaper}
- Adding functionality to {pegboard}
- Adding styling elements to {varnish}
- Adding functionality to carpentries/actions

2 System Setup

2.1 Software Tools

Development of Workbench components requires the same toolchain for working on lessons:

- R
- pandoc
- Git

It is recommended to have the latest versions of R and pandoc available. You need at least git 2.28 for security purposes.

```
R version
---


R version 4.3.0 (2023-04-21) -- "Already Tomorrow"
Copyright (C) 2023 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under the terms of the
GNU General Public License versions 2 or 3.
For more information about these matters see
https://www.gnu.org/licenses/.



pandoc version
---


pandoc 2.19.2
Compiled with pandoc-types 1.22.2.1, texmath 0.12.5.2, skylighting 0.13,
citeproc 0.8.0.1, ipynb 0.2, hslua 2.2.1
Scripting engine: Lua 5.4
User data directory: /home/runner/.local/share/pandoc
```

```
Copyright (C) 2006-2022 John MacFarlane. Web:  https://pandoc.org
This is free software; see the source for copying conditions. There is no
warranty, not even for merchantability or fitness for a particular purpose.
```

```
git version
---
```

```
git version 2.41.0
```

2.2 R Packages

Once you have these installed, make sure to install ALL of the dependencies for the workbench:

```
install.packages(c("sandpaper", "pegboard", "varnish", "tinkr"),
  dependencies = TRUE,
  repos = c(getOption("repos"), "https://carpentries.r-universe.dev"))
```

Working on Linux?

If you are on Linux, you will run into a couple of fun aspects that you may already be familiar with, especially if you have ever tried to install bioinformatic software:

1. having to also install some extra C libraries (which are akin to R packages, but for C), such as the `xslt` library.
2. having to build all packages from source

You can find detailed instructions in The Sandpaper Setup Guide, but the relevant commands are below.

System Dependencies

Here is the gist **for Ubuntu Users to get system dependencies set up.** Use The Carpentries R-Universe API to get all of the system dependencies. Here's how to do that via CURL:

```
curl https://carpentries.r-universe.dev/stats/sysdeps 2> /dev/null | jq -r '.headers[0]
```

This list can be sent to `apt-get install` to install everything:

```
sudo apt-get install -y \
  $(curl https://carpentries.r-universe.dev/stats/sysdeps 2> /dev/null | jq -r '.headers
  || echo "Not on Ubuntu"
```

Binary Packages

To get binary packages for your system, I will admit that it's *slightly confusing* because they bury the instructions for registering your system to use binaries in the *admin* pages and even then, it's kinda long. The gist is that you need to do two things:

1. set your HTTPUserAgent header to state your R version and platform
2. add the packagemanager CRAN-like repository to R's options:

Here's a script that you can copy and paste into `~/.Rprofile` which will be run every time you start R

```
local({
  # Set the default HTTP user agent to get pre-built binary packages
  RV <- getRversion()
  OS <- paste(RV, R.version["platform"], R.version["arch"], R.version["os"])
  codename <- sub("Codename.\t", "", system2("lsb_release", "-c", stdout = TRUE))
  options(HTTPUserAgent = sprintf("R/%s R (%s)", RV, OS))

  # register the repositories for The Carpentries and CRAN
  options(repos = c(
    carpentries = "https://carpentries.r-universe.dev/",
    CRAN = paste0("https://packagemanager.posit.co/all/__linux__/", codename, "/latest")
  ))
})
```

When you have this set up, you can then install the workbench packages:

```
# Install The Workbench and dependencies
install.packages(c("sandpaper", "varnish", "pegboard", "tinkr"), dep = TRUE)
```

The {sandpaper} package comes with the {usethis} package embedded (though this may change in the future). In addition, you will need the {devtools} for development.

I would also *highly* recommend the {pandoc} package for managing pandoc versions (NOTE: this requires you to have a personal access token set up).

```
install.packages("devtools")
install.packages("pandoc")
```

Once you have devtools, be sure to run `devtools::dev_sitrep()` and `usethis::git_sitrep()` to make sure you have the tools to build The Workbench:

```
devtools::dev_sitrep()
#>   R
#> • version: 4.3.0
#> • path: '/usr/lib/R/'
#>   devtools
#> • version: 2.4.5
#>   dev package
#> • package: <unset>
#> • path: <unset>
#>   All checks passed

usethis::git_sitrep()
#> Git config (global)
#> • Name: 'Zhian N. Kamvar'
#> • Email: 'zkamvar@gmail.com'
#> • Global (user-level) gitignore file: <unset>
#> • Vaccinated: FALSE
#>   See `?git_vaccinate` to learn more
#>   Defaulting to 'https' Git protocol
#> • Default Git protocol: 'https'
#> • Default initial branch name: 'main'
#> GitHub
#> • Default GitHub host: 'https://github.com'
#> • Personal access token for 'https://github.com': '<discovered>'
#> • GitHub user: 'zkamvar'
#> • Token scopes: 'gist, repo, user, workflow'
#> • Email(s): 'zkamvar@gmail.com (primary)', ...
#> Git repo for current project
#>   No active usethis project
```

Created on 2023-05-30 with reprex v2.0.2

2.3 Development Workflow

This development workflow is known as Test Driven Development in which a test is written *before* things work. This way, we can confirm that a bug is fixed once it passes the tests and we have confidence that it will not fail again.

1. open RStudio and switch to the project for the package you are working on
2. checkout a new branch for your feature/bug
3. **load package** via `devtools::load_all()` or ctrl+shift+L (use cmd on macOS) to load the package NAMESPACE
4. **run tests** (either via `devtools::test()` or ctrl+shift+T to run the entire test suite OR to test a single file, use the "run tests" button in a test file or run `testthat::test_local(filter = '[FILE SLUG]')`
5. **modify tests** for new functionality/bug fix
6. **add functionality/bug fix** and move to 3 unless you are ready to push
7. run check with `devtools::check()` or ctrl+shift+E

3 Testing The Workbench

> ⚠ This section is still under construction!
>
> We are still assembling the documentation for this part of the site. If you would like to contribute, please feel free to open an issue.

3.1 Introduction

> The first stage of your testing journey is to become convinced that testing has enough benefits to justify the work. For some of us, this is easy to accept. Others must learn the hard way.
>
> — Wickham and Bryan, Testing Basics, **R Packages** second edition

> If you use software that lacks automated tests, you are the tests.
>
> — Jenny Bryan source tweet (2018-09-22 01:13 UTC)

Every single package that runs code in the lesson infrastructure is tested before it ever reaches any lesson. This is important because we want to give the lesson authors and maintainers as much freedom as they need to write a lesson while maintaining predictability and integrity. We also want to give our community confidence that this system works.

Whenever a new feature or bug fix is added to The Workbench, it is imperative that a test is associated and verified before it gets sent into production.

Tests can be run locally and via continuous integration. This page introduces some of the testing strategies used in The Workbench and the caveats that come with these strategies.

3.2 Unit Testing

The tests under `test/testthat/` are run in alphabetical order using the {testthat} package (see https://r-pkgs.org/testing-basics.html) via `devtools::test()` or `devtools::check()`.

3.2.1 Conditionally Skipped Tests

The tests often need special conditions in order to run and sometimes those conditions are not possible. One of the most common conditions to skip is if the testing happens on CRAN. They are very hawkish about how long test suites can run and it's often difficult to detect the state of a CRAN machine, so it's better to skip long-running tests or those with complex environmental dependencies on CRAN (which does not yet apply to {sandpaper}).

3.3 Continous Integration

All the unit tests are run in continuous integration for every push and pull request that occurs. They also run every week. This provisions the current releases of the R package dependencies along with development versions of critical dependencies such as {renv}.

In continous integration, we run on with the following conditions to make sure it works not only on GitHub, but also on local user machines:

- test coverage (no package structure) with released versions on Ubuntu Linux (though reporting is stalled)
- For each platform (Ubuntu Linux, macOS, and Windows)
  - `R CMD check`, which checks the structure of the package and documentation
  - all run on these versions of R: current, devel, and two previous R versions

Because of occasional provisioning failures on macOS and Windows, we require only that Ubuntu Linux latest version passes check for merging pull requests.

3.4 Lesson Integration Testing

Part I

Building Lessons

4 Flow Diagrams

4.1 Introduction

This section builds on The broad workflow and details the internal process that are invoked with the `sandpaper::build_lesson()` function. If you look at the source for this function, it contains a total of sevens significant lines of code (many more due to documentation and comments).

The pre-flight steps all happen before a single source file is built. These check for pandoc, validate the lesson, and configure global elements. The last two lines are responsible for building the site and combining them with the global variables and templates.

Users will invoke this function in the following ways:

venue	function	purpose
local	`sandpaper::build_lesson()`	render content for offline use
local	`sandpaper::serve()`	dynamically render and preview content
remote	`sandpaper:::ci_deploy()`	render content and deploy to branches

All of these methods will call `sandpaper::validate_lesson()` (which also sets up global metadata and menu variables) and the two-step internal functions `sandpaper:::build_markdown()` and `sandpaper:::build_site()`. Below, I break down and detail the process for each.

4.2 Preflight Checks

Before a lesson can be built, we need to confirm the following:

1. We have access to the tools needed to build a lesson (e.g. pandoc). This is achieved via the `sandpaper::check_pandoc()`
2. We are inside a lesson that can be built with The Carpentries Workbench

4.3 `validate_lesson()`

The lesson validator is a bit of a misnomer. Yes, it does peform lesson validation, which it does so through the methods in the `pegboard::Lesson` R6 class.

In order to use thse methods, it first loads the lesson, via the `sandpaper::this_lesson()` function, which loads *and* caches the `pegboard::Lesson` object. It also caches elements that are mostly duplicated across episodes with small tweaks for each episode:

- metadata in JSON-LD format
- sidebar
- extras menu for learner and instructor views

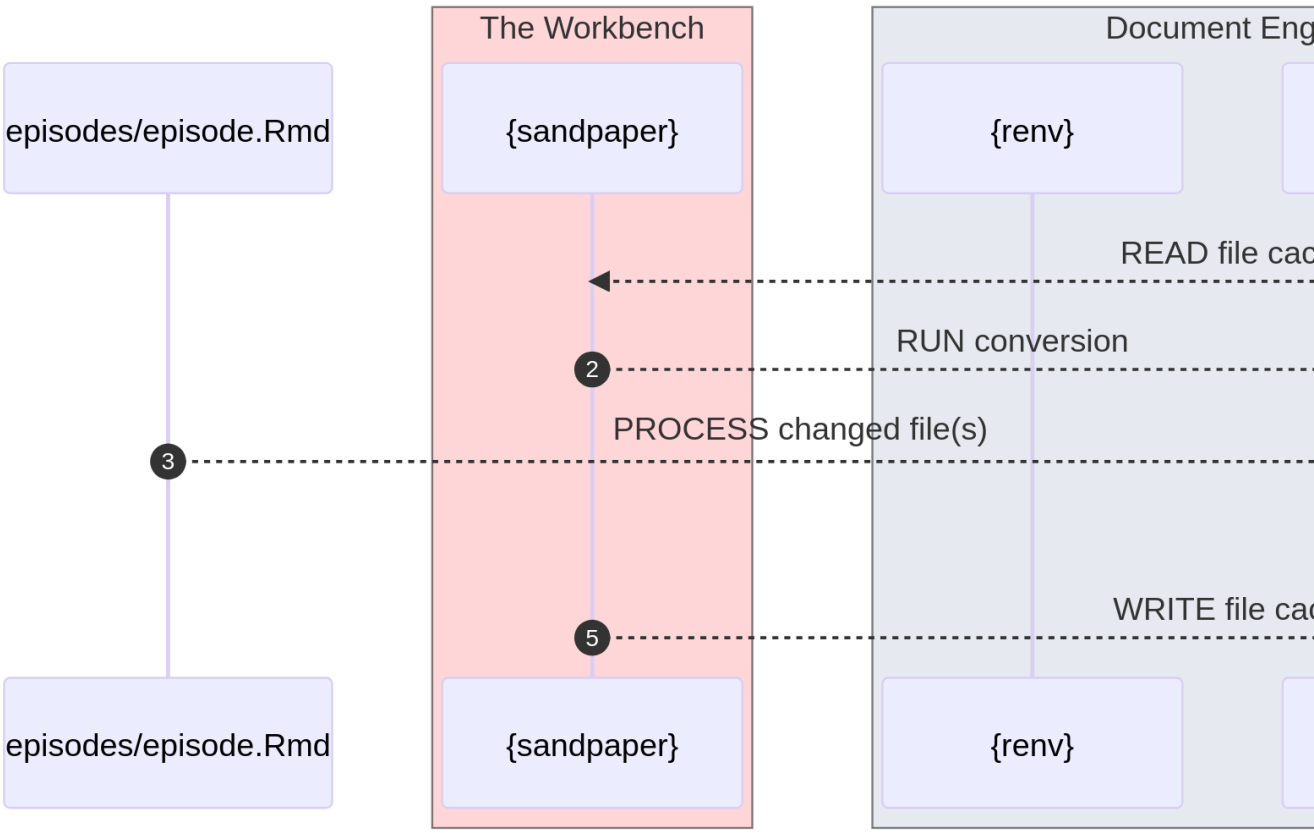
4.4 `build_markdown()`

4.4.1 Generating Markdown

Markdown generation for the lesson is controlled by the internal function `sandpaper:::build_markdown()`.

When a lesson contains R Markdown files, these need to have content rendered to markdownsot hat we can further process them. This content is processed with the {knitr} R package *in a separate R process*. Markdown source content on the other hand is copied to the `site/built` folder.

Because R Markdown files can take some time to render, we use MD5 sums of the episode contents (stored in the `site/built/md5sum.txt` file) to skip any files that have not changed.



Package Cache and Reproducibility

One package that is missing from the above diagram is {renv} and that's partially because it has an indirect effect on the lesson: it provisions the packages needed to build the lesson. When episodes are rendered from R Markdown to Markdown, we attempt to reproduce the build environment as closely as possible by using the {renv} package. If the global package cache from {renv} is available, then the lesson profile is activated before the episode is sent to {knitr} and R will use the packages provided in that profile. This has two distinct advantages:

1. The user does not have to worry about overwriting packages in their own library (i.e. a graduate researcher working on their dissertation does not want to have to rewrite their analyses because of a new version of {sf})
2. The package versions will be the same as the versions on the GitHub version of the site, which means that there will be no false positives of new errors popping up

For details on the package cache, see the Building Lessons With A Package Cache article.

At this step, the markdown has been written and the state of the cache is updated so if we re-run this function, then it will show that no changes have occured. After this step, the internal function `sandpaper:::build_site()` is run where the markdown file that we just created is converted to HTML with pandoc and stored in an R object. This R object is then manipulated and then written to an HTML file with the {varnish} website templates applied.

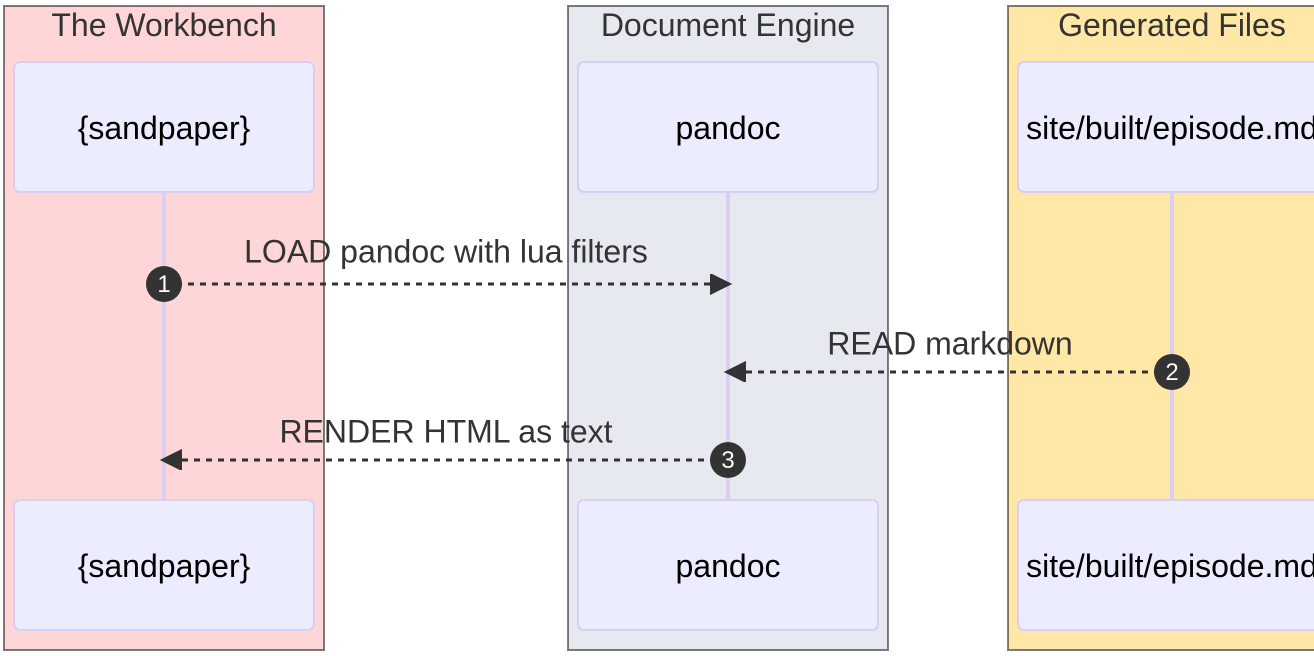
We use this function in the pull request workflows to demonstrate the changes in markdown source files, which is useful when package versions change, causing the output to potentially change.

4.5 `build_site()`

The following sections will discuss the HTML generation (the following section), manipulation (the section after that), and applying the template (the final section) separately because, while these processes are each run via the internal `sandpaper:::build_site()` function, they are functionally separate.

4.5.1 Generating HTML

Each markdown file is processed into HTML via pandoc and returned to R as text. This is done via the internal function `sandpaper:::render_html()`.

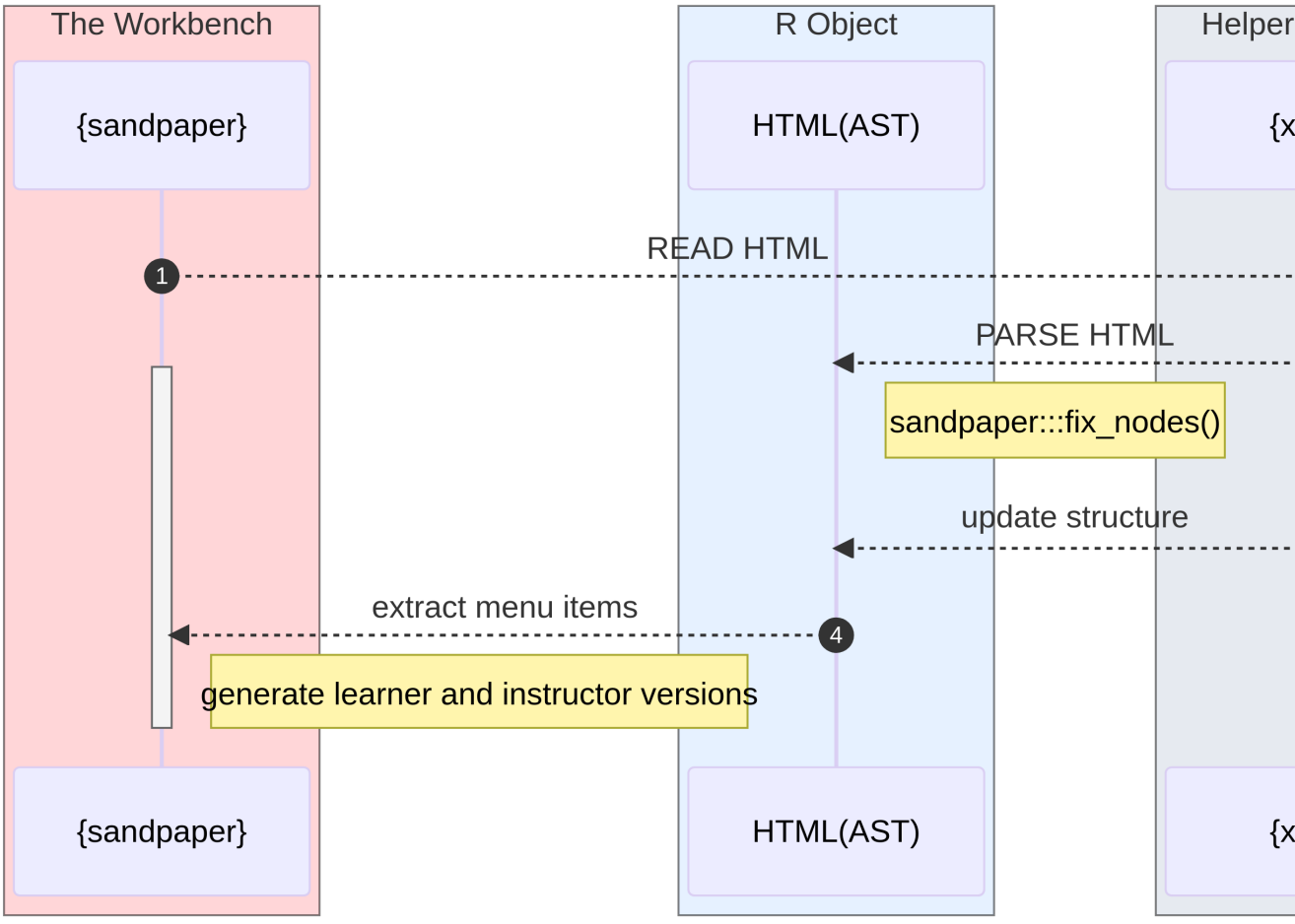


From here, the HTML exists as the internal body content of a website without a header, footer, or any styling. It is nearly ready for insertion into a website template. The next section details the flow we use to tweak the HTML content.

4.5.2 Processing HTML

The HTML needs to be tweaked because the output from pandoc, even with our lua filters, still needs some modification. We tweak the content by first converting the HTML into an Abstract Syntax Tree (AST). This allows us to programmatically manipulate tags in the HTML without resorting to using regular expressions.

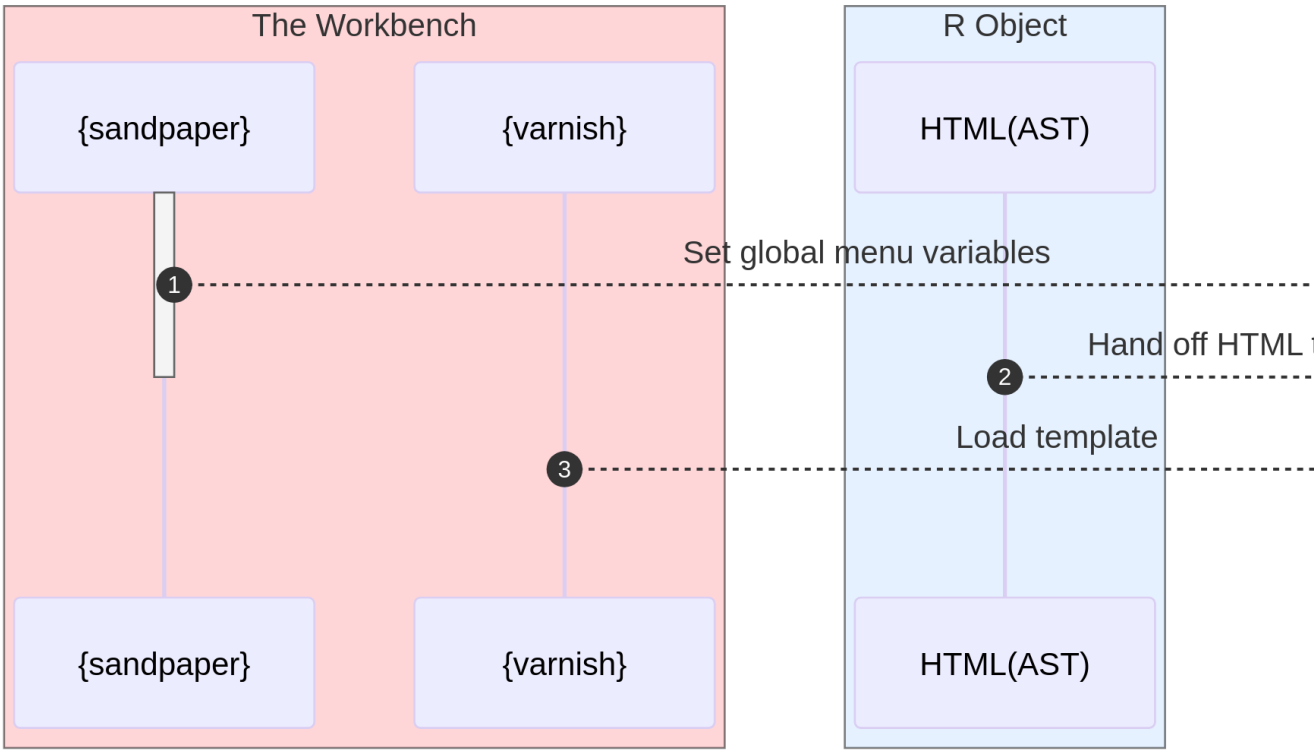
In this part, we update links, images, headings, structure that we could not fix using lua filters. We then use the information from the episode to complete the global menu variable with links to the second level headings in the episode.



> **Working With XML**
>
> Working with XML data is perhaps one of the strangest experiences for an R user because in R, functions will normally return a copy of the data, but when working with an XML document parsed by {xml2}, the data is modified *in place*.
> It allows us to do neat things, but there is a learning curve associated.

4.5.3 Applying Website Template

Now that we have an HTML AST that has been corrected and associated metadata, we are ready to write this to HTML. This process is achieved by passing the AST and metadata to {pkgdown} where it performs a little more manipulation, applies the {varnish} template, and writes it to disk.



Part II

{sandpaper} User Interface

5 The {sandpaper} package

> sandpaper (n.)
>
> Heavy paper coated on one side with sand or other abrasive material and used for smoothing surfaces.

The {sandpaper} package is the user interface for The Carpentries workbench and orchestrates the building and deployment of lessons, thus it helps lesson developers, authors, and maintainers to smooth out their contributions. Because of it's user-facing and modular nature, it is the most complex package in The Workbench. It has both {pegboard} and {varnish} as dependencies and employs the following programming techniques:

- Metaprogramming
- Function Aliasing
- Function Factories
- Subprocessing
- Lua (language, via pandoc)
- Logicless Templating (via mustache)
- XPath
- System calls (to git)

6 Testing {sandpaper}

6.1 Introduction

{sandpaper} is the largest package and the main user and deployment interface for The Workbench. The tests are all designed to work within a lesson context, which means a couple of things to be aware of.

1. **Some tests will take a long time to run.** IO procedures are often one of the most time-consuming steps in computing and {sandpaper} does a lot of it.
2. **Most tests will be dependent on the previous tests in a file.** Because the tests work on a folder on disk, we need to provision different scenarios of lesson state. It is far simpler to do this by side-effect rather than copying, setting up, and tearing down the state of the lesson for each and every test.

6.2 Test Setup

There is nothing that you as the contributor/developer need to do to set up for running these tests beyond what you have already done in your development setup. This section describes conceptually how {testthat} and {sandpaper} setup the testing environment after you run `devtools::test()`.

In short, this is the process:

1. {sandpaper} is loaded
2. Helper test files are loaded
3. The setup script is loaded, which provisions the test lesson
4. Each test file matching`tests/testthat/test-*.R` is run, resetting the test lesson at the top of each file.

6.2.1 Test Helpers

Sandpaper has three test helpers that handle some of the more tedious side-effects of testing.

`helper-hash.R` Expectation `expect_hashed()` that an episode file MD5 sum (expected) matches the MD5 sum (actual) we recorded in the `site/built/md5sum.txt` database.

`helper-processing.R` Provides an output string that demonstrates a screen output of a file being processed by {knitr}. This is used with `expect_output()`.

`helper-snap-transform.R` A function that is passed to the transform parameter of `expect_snapshot()` and will mask the temporary directory path so that the snapshot does not continuously invalidate on every run.

6.2.2 Setup Script

The first script to run is `tests/testthat/setup.R`, where a test lesson and a local git remote is created and stored in a temporary location for the duration of the test suite and a reset function is exposed for the tests.

6.3 Conditionally Skipped Tests

Each link below will open a code search for the different types of skipped tests in {sandpaper}

`skip_on_os` These are often tests that are skipped on Windows, usually for the reason that {renv} and filepaths behave slightly differently on Windows in a continuous integration setting.

`skip_if` Tests that are skipped under various conditions. For example, if Git is not installed on a system, we should not test any of the CI functions because they rely on Git.

`skip("<for reasons>")` This pattern is more rare. It's really useful in a situation where you are refactoring and know that a lot of tests will fail. If you sprinkle in these skips, you can focus on testing the core functionality of the refactor and then address the side-effects. Regarding the skips that remain: during testing, sometimes we encounter a ghost in the machine and we cannot set up the right conditions to run the test properly or the test was created with an earlier model of the package that we haven't been able to shake. In these cases, instead of deleting the test or commenting out code, we add the `skip()` function and write a message of *why* we skipped it so if we need to come back to it later, we can.

6.4 Continuous Integration

Part III

{pegboard} Validation and Parsing

7 The {pegboard} package

nothing to see here yet

Part IV

{varnish} Web Styling

8 The {varnish} package

8.1 Introduction

The {varnish} package is a weird little package in that it does not contain any actual R code. It's purpose is to host HTML templates along with the CSS and JavaScript needed to display the lesson.

We take advantage of the fact that the only thing actually *required* to install an R package is the presence of a `DESCRIPTION` file[1]. These all live inside the `inst/` folder, which is a place that allows files to be installed along with the package.

The {pkgdown} package uses this as a mechanism for package developers to override the default styling, creating customized documentation websites such as the rOpenSci documentation sites: https://docs.ropensci.org/rotemplate/.

This allows people to update the lesson styling however they wish[2] and while we *could* include it in {sandpaper}, it's best kept separate so that people can update {varnish} without needing to update the entire tool suite.

8.2 Design and Implementation Background

The design for the frontend was created by Emily de la Mettrie in 2021 after consultation with Zhian N. Kamvar and François Michonneau using examples from The Unix Shell and parts of Exploring Data Frames for content cues.

The final figma design project[3] was then handed off to a team at Bytes.co, who translated the designs to CSS and JavaScript, subcontracted an a11y testing company to interactively test the prototype for a11y issues.

The prototype we recieved from Bytes.co was a Jekyll template serving HTML files. Zhian created a staging repository called shellac to transform the site from one that was served via

[1] for example, a DESCRIPTION file is a low-rent way of specifying depedencies for a manuscript).

[2] There is a caveat, though. There are some components, such as the sidebar and overview callouts that are built on the {sandpaper} side and then fed into the template.

[3] The final figma design project for The Workbench can be found in this link. To preview what it looks like, click the "Present" button (iconized as a triangle play button) in the top right.

static site generator to one that was standalone. The preview is preserved at https://zkamvar.github.io/shellac/student_carpentries.html.

This site was then stripped of the added into {varnish} in carpentries/varnish#14 between 2022-01-10 and 2022-01-24, when the 1.0.0 release of {varnish} was created and the sandpaper docs website was updated to use the new version of the HTML, CSS, and JavaScript.

Part V

Package Distribution

9 Release Process for Workbench Packages

9.1 Background

The workbench contains three main packages:

- {sandpaper}: user interface and workflow engine
- {pegboard}: parsing and validation engine
- {varnish}: HTML templates, CSS, and JS elements

Each of these packages are available on the Carpentries R-Universe and new versions are checked for hourly. This allows folks to get up-to-date versions of The Workbench packages built for their system without running out of GitHub API query attempts.

In order to maintain quality, packages are only sent to the R-Universe if they have been formally released on GitHub (as specified in the packages.json configuration file). This allows us to incrementally add new experimental features without changing the stable deployments.

9.2 Release Process

When a package is ready for release we use the following checklist:

☐ Update version number in DESCRIPTION

☐ Add NEWS for the changes in this version

☐ Ensure all changes are committed and pushed

☐ add new signed tag with the name " X.Y.Z"

```
# example: create a signed (-s) tag for sandpaper 3.3.3
git tag -s 3.3.3 -m 'sandpaper 3.3.3'
```

☐ create a release on github from the new tag

> **Note**
>
> Zhian likes to create tags via the command line because he has set up his git configuration to use a gpg signature so the tags and the releases are both verified.

The last two items can be achieved in a single step with the github cli with the command `gh release create X.Y.Z` for the version number

```
gh release create 3.3.3
# ? Title (optional) sandpaper 3.3.3
# ? Release notes  [Use arrows to move, type to filter]
#   Write my own
# > Write using generated notes as template
#   Leave blank
```

Selecting "Write using generated notes as a template" opens an editor and populates it with the pull requests that have been accepted since the last release.

Once the relase is created on GitHub, then the package will be available on the R-Universe in about an hour or less.

Part VI

Building Lessons Remotely

Part VII

GitHub Actions

10 Summary

In summary, this book has no content whatsoever.

References

Wickham, Hadley, and Jennifer Bryan. 2023. “R Packages (2e).” https://r-pkgs.org/.