A Reproducible Research Compendium

cf. list of contributors at
https://github.com/rr-mrc-bsu/reproducible-research/graphs/contributors

2019-02-20

Contents

Chapter 1

A 'Living Book' - aims and scope

This book is a little different from your ususal statistics foliant - it is written entirely using **Markdown** and rendered to html, pdf, and epub publishing formats using the R package **bookdown**. Its entire Markdown source code is publicly available on GitHub.com at https://github.com/rr-mrc-bsu/reproducible-research. A pre-build version is hosted as static html website using **GitHub pages** at https://rr-mrc-bsu.github.io/reproducible-research/. This structure allows to easily discuss changes using GitHub issues https://github.com/rr-mrc-bsu/reproducible-research/issues, organize further development using milestones and projects, contribute corrections or even entire chapters by creating pull requests, and to manage editions by GitHub releases. It also means that everybody - and yes, that does include you - can become a contributor by creating pull requests in the GitHub repository. Since the contents are thus evolving over time as long as there are active contributors to the project, the book is 'living'.

The overall purpose of the *Reproducible Research Compendium* is threefold:

1. Provide a platform for discussing aims and objectives as well as best practices for reproducible research with a clear focus on applications in biostatistics.
2. Build-up a lasting compendium for knowledge sharing around various issues and methods for coping with them that may broadly be subsumed under the term 'reproducible research'.
3. The book project itself acts as a learning-by-doing example for its contributors with the goal of anybody participating becoming knowlegable about organizing collaborative open-source [mostly coding] projects.

The complete documentation for **bookdown** can be found at https://bookdown.org/yihui/bookdown/. Note that R is a prerequisite but only for building the book - the contents itself are completely language-agnostic.

Chapter 2

How to contribute

Since this book is a collaborative effort, the most important thing is enabling people to contribute! This chapter is a hands-on tutorial and does not go into the details of the required steps. Each of the techniques will be explained in more detail in the subsequent chapters.

2.1 Before you start

In the following we will assume that you are working on a Linux machine. In fact, using an open source operating system such as Linux is already the first step towards reproducibility since everybody else is free (also as in 'free of charge') to use the exact same operating that you are running. Windows or MacOS on the other hand might not be available to everybody. A great way to get started with Linux is Ubuntu and the easiest way to install it on an existing Windows or MacOS machine might be via a virtual machine. Detailed instructions on how to do so can be found here.

2.2 Getting the book's source code

The book is compiled from a collection of R markdown files which is a special text file format that allows to combine code and text within the same file as well as basic markup[1]. The motivation behind markdown is to separate the content entirely from the layout and stick to an absolute minimum number of markups (e.g. headings, enumeration, hyperlinks) to be able to compile the document to as many different output formats as possible. The actual compilation will be done in 'pandoc' and will be explained later [ref].

A very important pillar of reproducibility is version control, i.e., some mechanism to keep track of changing files over time and enabling roll-backs to previous versions. For more details on why version control is so important in reproducible research and how to implement it, cf. [ref: version control chapter]. For now, we will just focus on how to install and use a specific program, git, to obtain the source code for the book and make changes to it. Assuming that you have a linux system running (we will assume an Ubuntu installation) you will need to install the version control system git [link to chapter]. The easiest way to do this is via the command line package manager 'apt'. Open a terminal window and execute the command

```
sudo apt -y install git
```

This will download and install all required packages from the official repository.

[1] Markups are simple meta information on text such as 'this is a headline' etc. If you are familiar with LaTeX you are already a markup professional since LaTeX supports a huge number of different markup expressions. HTML is also a markup language.

You may now 'clone' the online repository of the book from its GitHub.com website: [https://github.com/rr-mrc-bsu/reproducible-research] (https://github.com/rr-mrc-bsu/reproducible-research). Here 'cloning' does exactly what it says: it downloads an exact copy of the entire source code including its complete history of previous changes to your local computer to work on. Assuming that you have a terminal window opened and the working directory is your home directory '~' you clone the repository by invoking

```
git clone https://github.com/rr-mrc-bsu/reproducible-research.git
```

This will create a new folder 'reproducible-research' in your current working directory and download all necessary files. You should then change the working directory to the new folder via

```
cd reproducible-research
```

2.3 Creating a new 'branch' for your changes

By default your git repository will now be on its 'master' branch. You may verify that via

```
git status
```

Branches are just different variants of the source code that may exist in parallel and one major job of git is making it possible to bring these branches together. The master branch is special in that it is usually considered the current 'best' variant of a project. For most smaller projects, a single master branch might be sufficient but things do get a bit hairy when many people could potentially change this common master branch at the same time. Also, for this book project, each time the master branch in the online repository changes, the entire book is recompiled and published at https://rr-mrc-bsu.github.io/reproducible-research/. Therefore the books contributing guidelines require that no changes are made directly to the master branch. Instead, all work is done on separate feature branches, e.g., on 'my-cool-new-chapter' if you want to add a new chapter. To create this branch you run the following git commands

```
git branch my-cool-new-chapter
git checkout my-cool-new-chapter
```

This creates the new branch and checks it out (activates it). All changes that you now make to files in the directory only affect the version of the book associated with your local branch 'my-cool-new-chapter' (after you commit them, that is).

2.4 Create a new chapter

You may now add a new chapter simply by placing a new numbered .Rmd file in the top level of the book projects directory, e.g. the new file could be called '99-my-cool-new-chapter.Rmd'. Do feel free to browse the other chapters of the book already present to learn more about the R markdown syntax used to write the book. Note that although R markdown relies on R to compile the files, the contents of the file may not contain any R code at all. For more details on R markdown see [link chapter].

Once the new chapter file is created and you added some content you should check whether the altered book still compiles without errors. This is critically important for any piece of source code since even small changes might break the entire thing. Since you will not be able to incorporate you changes in the online version of the book without passing some automated checks, you may just as well check that everything is working locally before attempting to add your changes online.

To compile the book you will need R and some packages. The easiest way to install everything is again via the icommand line

```
sudo apt-get install r-base r-base-dev
Rscript -e 'install.packages("bookdown")'
```

You may then attempt to build the new book with your new chapter by invoking the build script

```
bash _build.sh
```

which essentially runs the R command

```
bookdown::render_book("index.Rmd")
```

to build the book and cleans up afterward. You should now see a new folder '_book/'and be able to open'_book/index.html' in your browser. This just opens the newly build version of the book locally. Make sure that everything is as you want it to be. Next you will want to 'commit' your changes to your local 'my-cool-new-chapter' branch. Commiting means that you store your changes in the git repository thus creating a snapshot in time that you may always return to irrespective of any further changes to the repository.

The repository should be configured in such a way as to ignore the _book/ folder that you created since this is just output. You can therefore simply add all changes and commit them with a short description of what you did

```
git add -A
git commit -m "added cool new chapter"
```

2.5 Publishing your changes

Changes to the master branch in the online repository are organized as pull requests. This is a feature on GitHub.com that allows you to publicly propose merging a branch back to master. Usually this is straightforward to do since the master branch will change very slowly (cf. [ref merge conflicts]). The pull request will then have to be reviewed by at least one other collaborator to the repository before you are able to actually merge the changes into the master branch - only at that point are they actually integrated in the published book.

To do so, you first have to be a contributor to the project [TODO: explain how to do that without being a contributor / how to become one]. Next, you will need to push your local branch to the online repository via

```
git push -u origin my-cool-new-chapter
```

Switch to a browser and open https://github.com/rr-mrc-bsu/reproducible-research. In the top/middle you then need to switch from the '<> Code' tab to the pull requests tab. Create a new pull request by clicking on the button. This opens a panel where you can define your pull request. A pull request always proposes to merge one branch onto another. In our case we want to merge 'my-cool-new-chapter' onto 'master'. That meas that we leave the 'base:' branch master as it stands by default. However, in the rop-down menu for 'compare:' you can now select you new branch. Note that the arrow between the two already indicates that the 'compare:' branch is supposed to be merged onto the 'base:' branch. In the panel below you will then see a git diff, i.e., a listing of all the differences between the two branches (gree: additions, red: deletions). Since you only added new stuff in this example and the master (probably) did not change between you downloading the latest copy of the repository and creating your changes, there are no merge conflicts. Confirm by clicking on 'create pull request'.

This will first create the pull request and the immediately trigger a build script on the continuous integration system Travis [reference to continuous integration]. The continuous integration system will spin up a virtual machine in the cloud, install all required software, download the repository and check whether the build script will still run without errors after merging you pull request. This process will take a few minutes and once it is completed the status of the build will be shown in your pull request. Even if the build script ran perfetly fine on your local machine the Travis build might fail if you introduced new dependencies without altering the Travis build configuration. For normal changes (only editing .Rmd files), the build should work without any errors. The advantage of having a CI system is that anybody reviewing your changes will

immediately know that your pull request did not introduce any breaking changes and that the book can still be compiled on the default minimal build system defined in the travis.yml file.

Once another contributor has reviwed your changes and approved them, your are then free to merge your pull request. Only this last action will actually change the master branch of the repository. This change will again trigger a build on Travis, this time for the newly merged master branch. Since the pull request was already checked, there should not be any further problems. Additionally, any build of the master branch will also execute the _deploy.sh script which will take the compiled book and push the output to the gh-pages branch of the repository. This branch is special in that it only contains the generated output and not the corresponding source code. GitHub.com offers the possibility of hosting static html pages free of charge via GitHub pages and the project is configured such that the contents of the gh-pages branch (feel free to inspect it) are used to populate the GitHub pages homepage of the project. This means that the version of the book displayed at https://rr-mrc-bsu.github.io/reproducible-research/ should automatically corresponds to the version obtained from the last commit to the repositories master branch.

Chapter 3

New Chapter test

[taken from bookdown template!]

Figure 3.1: Hello world !

Chapter 4

Version control

This chapter is dedicated to version contol

Chapter 5

R package development

5.1 Continuous integration

5.2 Unit testing

5.3 Documentation

Chapter 6

Build automation

Large projects can be a pain to manage. Small changes may break your software, or may deem your previously obtained data analysis results useless. Build automation refers to a collection of processes that atempt to automate as many steps as possible. Automation can be done on several levels. Here, we will mainly discuss two ways of build automation: automating your project pipeline using Makefiles and automating software checks using continuous integratino tools.

6.1 Makefiles

Most research projects consist of several different connected components. For example, the end product might be a manuscipt, which depends on intermediate components such as a data analysis script and an R package. In this case, the manuscript depends on the data analysis script, which in turn depends on the R package. Such a hierarchy implies that every time a file changes, all files downstream in the hierarchy should be updated as well. In the previous example, we might adjust a function in the R package, which might change the outcome of the data analysis, and as a result, we'd have to re-run the data analysis script. The outcome of the data analysis might change the manuscript, so we'd have to re-compile that as well. In large projects, this quickly becomes very tedious and difficult to maintain by hand. Luckily there is software available to streamline this process.

Consider the previous example. The hierarchy of files may be represented as: r-package.tar.gz > analysis.R > manuscript.Rnw. A change higher up the hierarchy might change the downstream files. Ideally, we would like to have a command that re-runs/compiles the different files everytime an upstream change is made. In our example, we would like to recompile the manuscript.Rnw file if something in the analysis.R file changes. And similarilly, if the R package changes, the analysis.R file should be re-run. This is exactly what the GNU software Make does.

6.2 Continuous integration

Did you ever wonder what the green/yellow/red 'badges' in some Readme.md files on, e.g., Github.com actually mean? How are they created, what are they for and why should you care?

This section will hopefully shed light on the meaning of some of these badges (those refering to a 'build status') and you will learn how to use these techniques for you own repositories. The key term here is 'continuous integration' (CI) which refers to a concept in software development where all working copies (in git one would refer to branches) of a project are frequently integrated into the mainline (in git terms: the master branch). The rationale being that frequent/continuous integration prevents diverging development

branches. Since the philosophy of git is to create feature branches for small, contained changes of master which are to be merged back as soon as possible CI and git are a natural fit.

In practice, however, frequent changes to master are dangerous. After all, the master branch should maintain the last at least working if not stable version of the project that other feature branches can be started from. It is thus crucial to prevent erroneous changes to be merged into the master branch too often. This means that CI requires some kind of automated quality checks that preemptively check for new bugs/problems before a pull request from a feature branch on master is executed. It is this particular aspect of CI that is most interesting to collaborative work on scientific coding projects - being able to automatically run checks/tests on pull requests proposing changes to the master branch of the project.

[Random thought: we should have an example repository for demonstraiting the different states of PRs etc. instead of just including pictures. Readers could then inspect the 'frozen' repository directly and see the PRs etc.!]

To enable this kind of feature on Github, a cloud-based farm or build servers is required where users can run build scripts in virtual machines and retrieve reports on the build status (0 worked, 1 failed). It is these build-statuses that the green/yellow/red badges report visually (yellow/gray being a pending build)! There are multiple companies offering these services (within reasonable bounds) for free for public repositories and as of 2018 the free academic account for GitHub also enables free builds using TravisCI for private repositories. It must be stressed though that, since everything is running in the cloud, the same constraints as for storing data on GitHub servers apply to downloading or processing data in buildscripts for CI services. [point to Jenkins as on-premis solution]

The obvious setting to use automated builds in is package development. This is by far the most common application and the current tools are definitely geared towards that use case. We will later discuss how to extend the scope to non-package situations. For instance, the repository containing the source code for this book also uses TravisCI for build automation even though it is not an R-package itself.