WDL Workflows

October 10, 2023

Table of Contents

# Overview

This book introduces WDL Workflows on AnVIL. After introducing several concepts, including basic WDL syntax, we present hands-on exercises to run a workflow, write a WDL, localize a file, customize a Docker image, and join the Discourse. No local software installation is required as each exercise leverages web-based resources.



## Skills Level

*Genetics*  
**Novice**: No genetics knowledge needed

*Programming skills*  
**Novice**: No programming experience needed

## Learning Objectives

* Understand when WDL Workflows are the right tool
* Run a Workflow on AnVIL
* Write a WDL using Broad Methods Repository
* Bring your own data to analyze
* Customize your Docker environment
* Join the conversation

## AnVIL Collection

Please check out our full collection of AnVIL and related resources: <https://hutchdatascience.org/AnVIL_Collection/>

# 1 Introduction

This lecture module introduces several basic concepts when working with WDL Workflows. It highlights several considerations when using Docker container technology and provides an overview of the main sections in an example WDL workflow. These slides are adapted from the [WDL 101 Workshop](https://support.terra.bio/hc/en-us/articles/8693717360411). More information can be found in [this presentation](https://zenodo.org/record/7062397#.YyM-vy1h1qu) hosted by Bioinformática UFMG.

**Learning Objectives**

1. Four parts of WDL Workflows on Terra
2. How Docker improves reproducibility
3. Overview of basic WDL syntax
4. Find and configure Workflows

You can view and download the Google Slides [here](https://docs.google.com/presentation/d/1Dioe_Qd3hdJ0UAP4LA6gJsm07Lr1wRpax5FLYr6xU3M).

# 2 Run Workflow

This tutorial demonstrates how to run a WDL Workflow on AnVIL. It uses the fun and accessible [WDL puzzles](https://support.terra.bio/hc/en-us/articles/360056599991) workspace to run a “Hello, World!” style workflow. By showcasing how input parameters are specified and where output files can be found, this exercise helps provide context before writing your own WDL Workflow.

**Learning Objectives**

1. Import WDL by cloning a Workspace
2. Run Workflow using AnVIL
3. Examine Output to appreciate data flow

You can view and download the Google Slides [here](https://docs.google.com/presentation/d/1FbTgCnAZTgVzp-lPwOVqOqU4Kw19nu825dMngfyc-Nw).

# 3 Write WDL

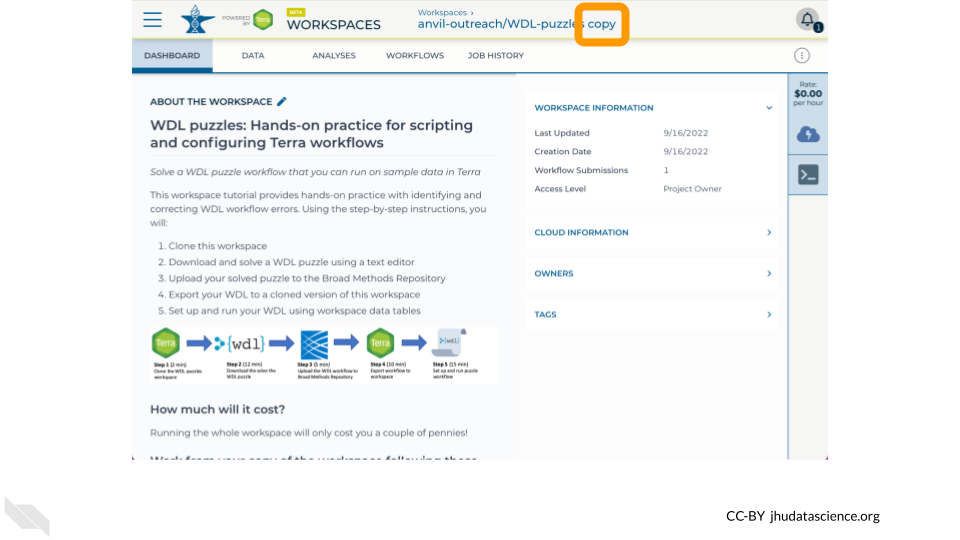
Now that you’ve successfully run a Workflow on AnVIL, this tutorial demonstrates how you can create and edit a WDL using the [Broad Methods Repository](https://portal.firecloud.org/?return=anvil#methods). While this “legacy” Methods repository does not have many of the features present in the open-source [Dockstore](https://anvilproject.org/overview#platform-interoperability) platform, it does offer a convenient web-based editor for demonstration purposes. This material is adapted from the [WDL 101 Workshop](https://support.terra.bio/hc/en-us/articles/8693717360411); you can read about other ways the Broad Methods Repository can be used in [this Terra Support article](https://support.terra.bio/hc/en-us/articles/360031366091).

**Learning Objectives**

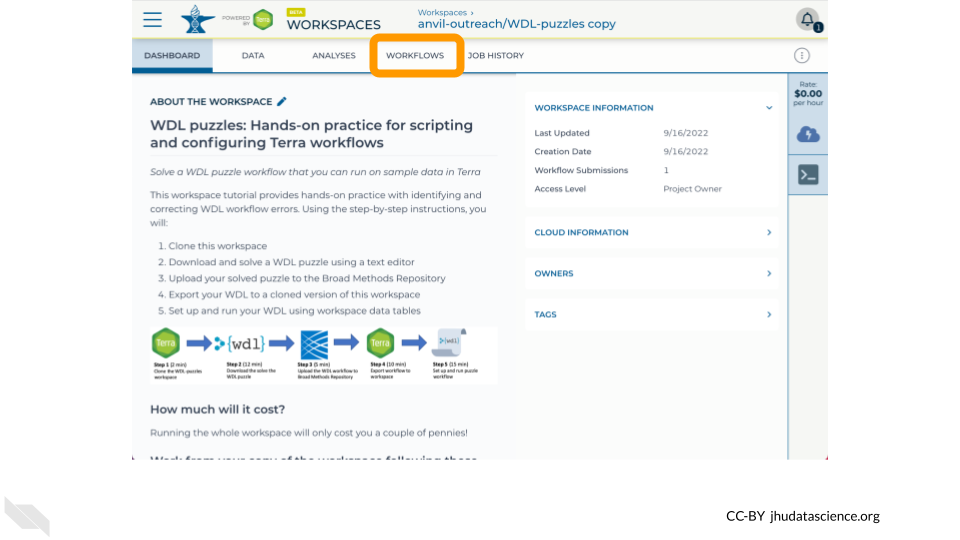
1. Access Broad Methods Repository
2. Write WDL101 Training Example
3. Export to Terra and run

## 3.1 Access Broad Methods Repository

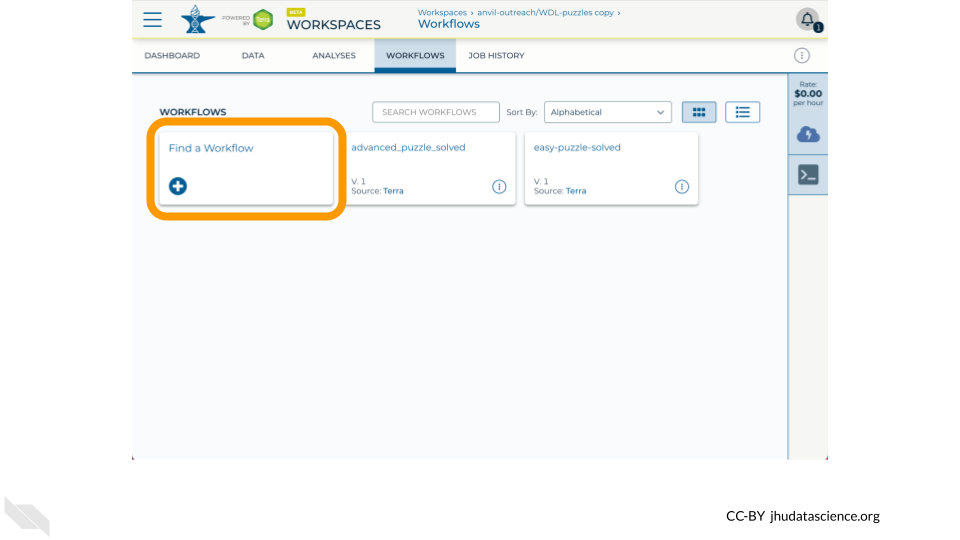
Let’s start by navigating to the WDL-puzzles workspace that we previously cloned. Please double check your workspace name to ensure that this is the copy that you made rather than the original as you will not be able to use the original workspace to create a new WDL or run a workflow.



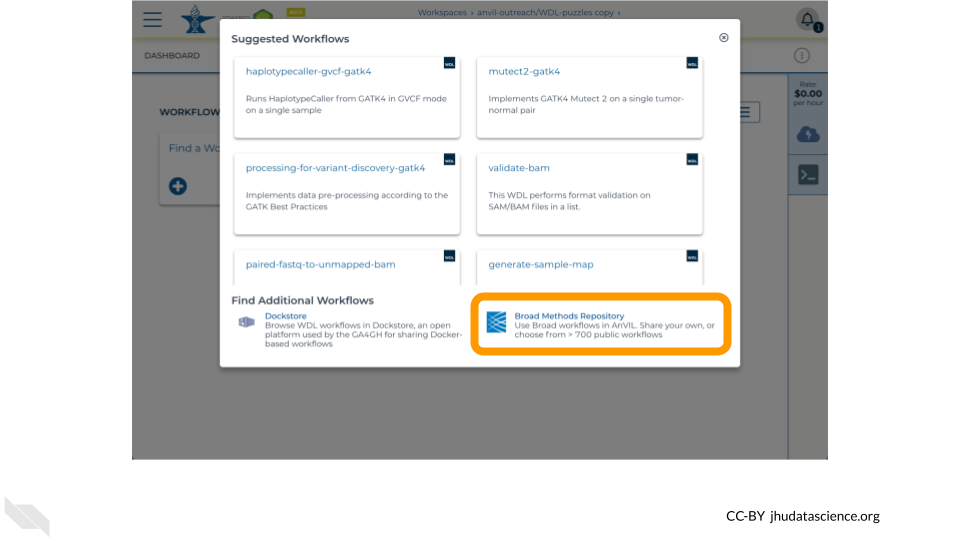
Once you’ve double checked that you are in a workspace that you can modify and compute, click on the Workflows tab.



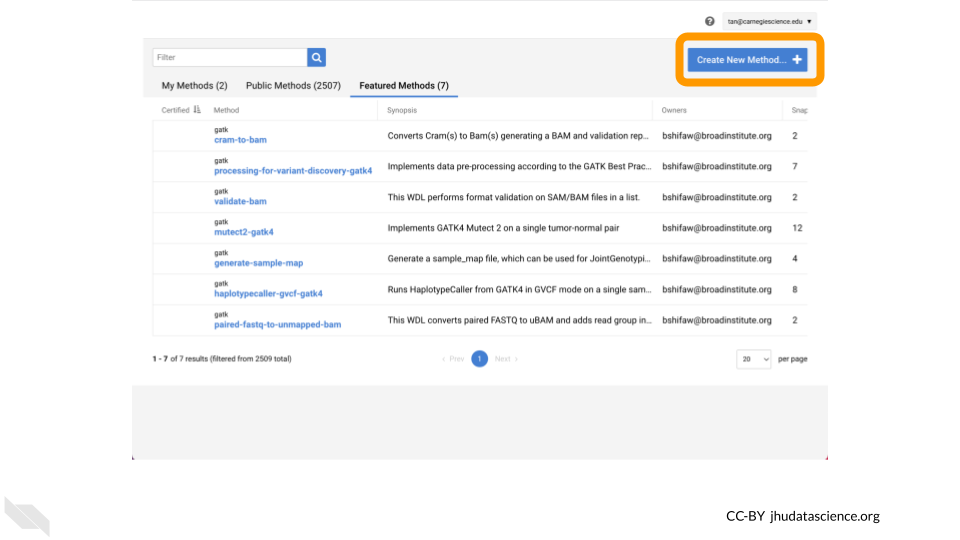
Click on the Find a Workflow card.



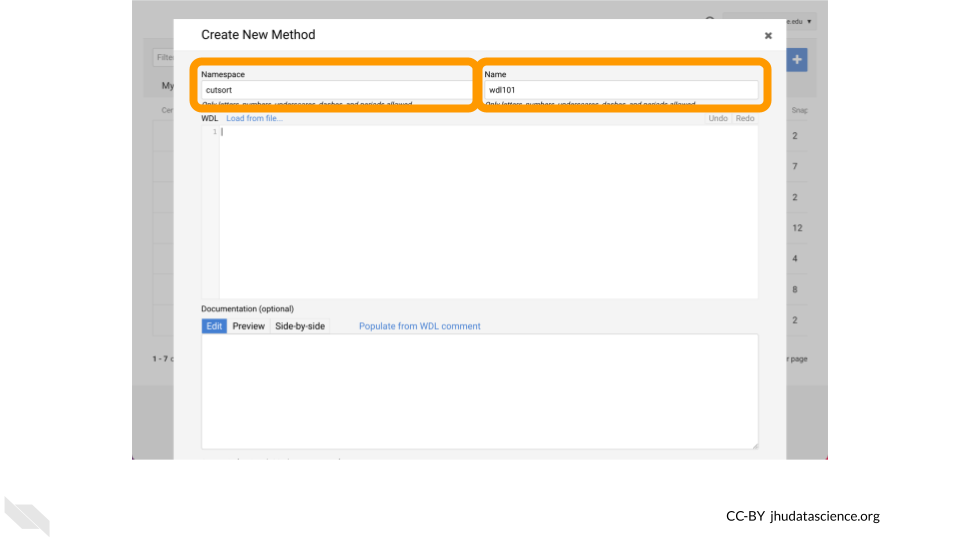
Select the Broad Methods Repository option.



Click Create New Method.



Add a namespace to the first text box to organize your WDLs. Your username (prepended with your lab name) is a reasonable namespace as this must be unique across all of Broad Methods Repository. Afterwards, add a name such as wdl101 to name your WDL.



## 3.2 Write WDL101 Training Example

Let’s now create a basic WDL! This simple “Hello, World!” style workflow will take as input a string, call a single task, and save the output of that task to your workspace bucket. The task that is called will run the [Bash](https://swcarpentry.github.io/shell-novice/01-intro/index.html) echo command to print the input string to stdout.

First note that we are using the [WDL 1.0 spec](https://github.com/openwdl/wdl/tree/main/versions).

version 1.0

Let’s add a workflow HelloInput that calls a single task WriteGreeting.

version 1.0  
workflow HelloInput {  
}  
  
task WriteGreeting{  
}

To create the task, we will define input, command, output, and runtime blocks. Note that the command block is defined as a “[here doc](https://en.wikipedia.org/wiki/Here_document)” and prints the input string to stdout.

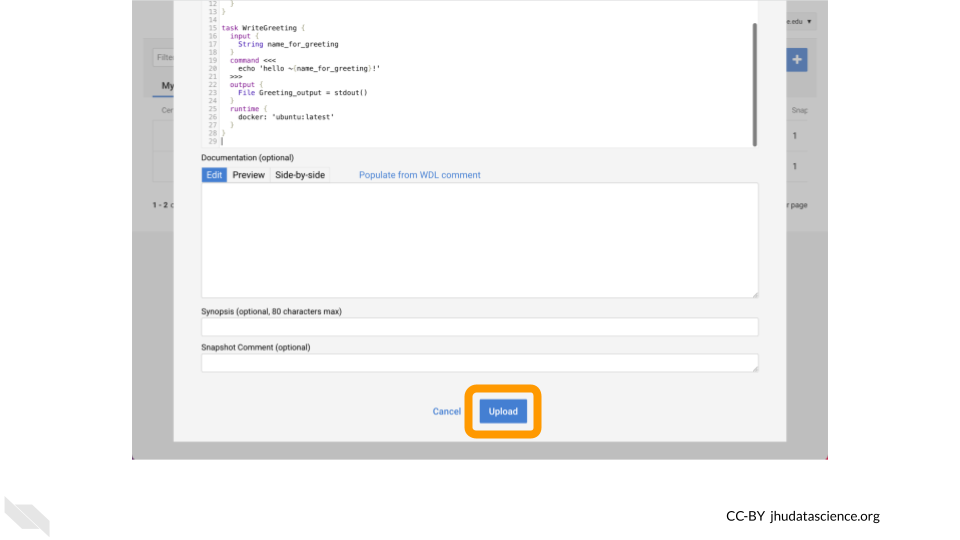
version 1.0  
workflow HelloInput {  
}  
  
task WriteGreeting {  
 input {  
 String name\_for\_greeting  
 }  
 command <<<  
 echo 'hello ~{name\_for\_greeting}!'  
 >>>  
 output {  
 File Greeting\_output = stdout()  
 }  
 runtime {  
 docker: 'ubuntu:latest'  
 }  
}

Putting it all together, we now create the workflow by defining an input string stored in a variable named name\_input, calling the task by passing name\_input to name\_for\_greeting, and storing what is returned by the task in a File labeled final\_output.

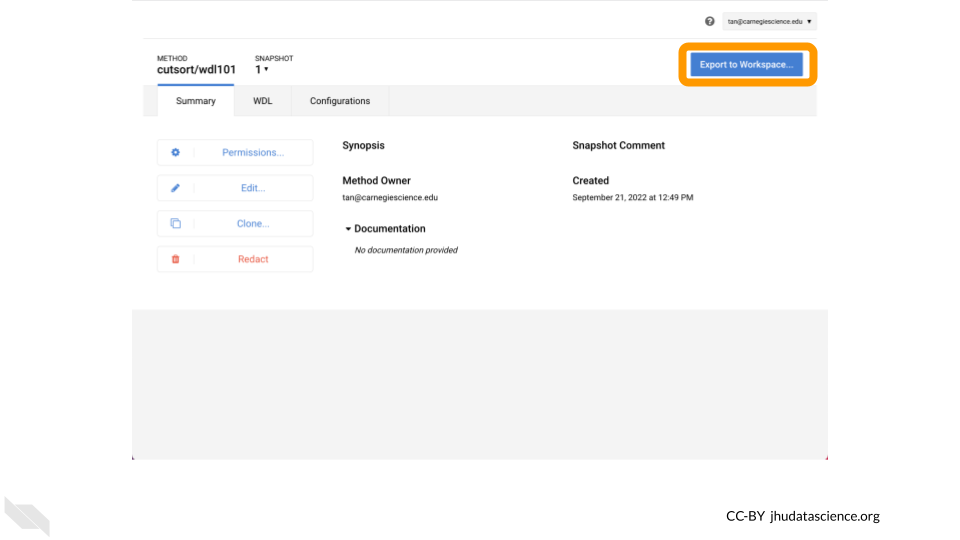
version 1.0  
workflow HelloInput {  
 input {  
 String name\_input  
 }  
 call WriteGreeting {  
 input:   
 name\_for\_greeting = name\_input  
 }  
 output {  
 File final\_output = WriteGreeting.Greeting\_output  
 }  
}  
  
task WriteGreeting {  
 input {  
 String name\_for\_greeting  
 }  
 command <<<  
 echo 'hello ~{name\_for\_greeting}!'  
 >>>  
 output {  
 File Greeting\_output = stdout()  
 }  
 runtime {  
 docker: 'ubuntu:latest'  
 }  
}

## 3.3 Export to AnVIL and run

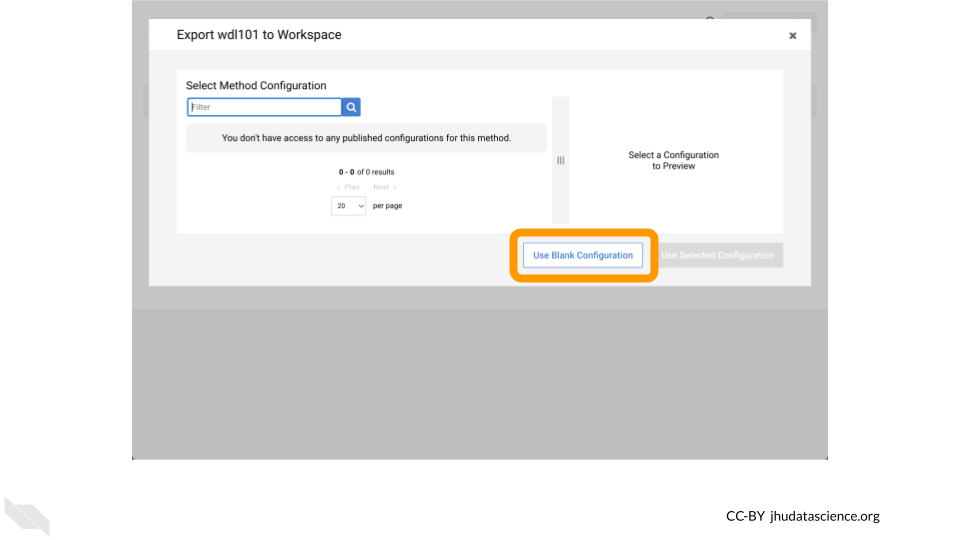
Once your WDL is complete, click on Upload.



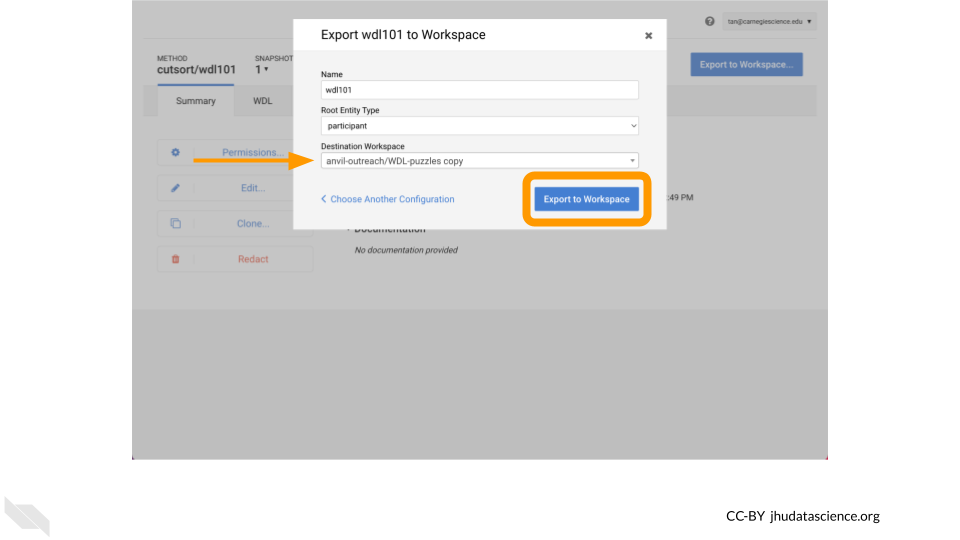
Now click on Export to Workspace.



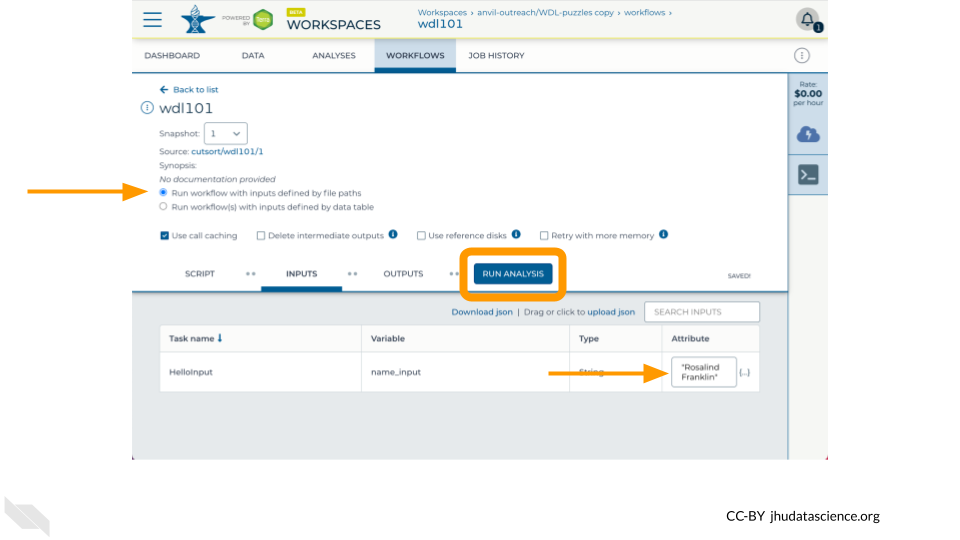
Select Use Blank Configuration.



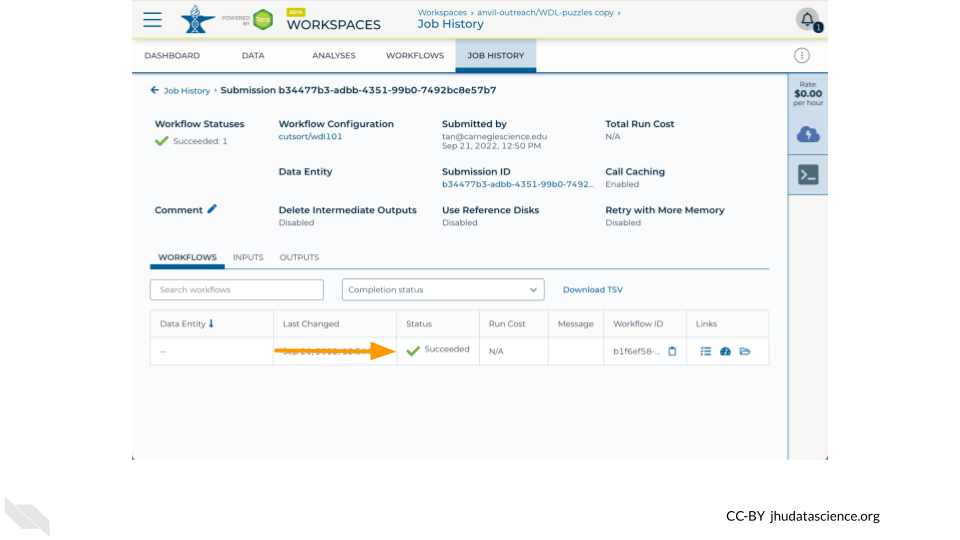
Select a Destination Workspace such as your clone of WDL-puzzles. Afterwards, click Export to Workspace.



Lastly, configure your Workflow as your did previously (e.g. inputs defined by file paths, name in double quotes), click Save, and then click Run Analysis.



Voila! Here’s what you hopefully see after successfully running your WDL101 Training Example !



# 4 Localize Files

# 5 Customize Docker

Having run a workflow, written a WDL, and localized a file, let’s now build a custom Docker image. This tutorial demonstrates how to build a Docker image that contains [SAMtools](https://pubmed.gov/19505943) so that we can calculate [idxstats](http://www.htslib.org/doc/samtools-idxstats.html) on a small .bam file. We will use the GitHub Action [build-and-push-docker-images](https://github.com/marketplace/actions/build-and-push-docker-images) so that you do not need to have Docker installed locally. This material was modified from the [WDL Bootcamp Workshop](https://support.terra.bio/hc/en-us/articles/18618717942427). More information about making Docker images can be found in [this Terra Support article](https://support.terra.bio/hc/en-us/articles/360024737591).

**Learning Objectives**

1. Identify what containerization is, and why it’s helpful for running workflows
2. Identify the key components of a Docker container
3. Understand how to find and use a pre-existing Docker image
4. Configure GitHub Actions
5. Build and run a Docker image
6. Create idxstats WDL
7. Run idxstats workflow
8. Understand how to specify a container for an AnVIL-powered-by-Terra workflow

## 5.1 Prerequisites

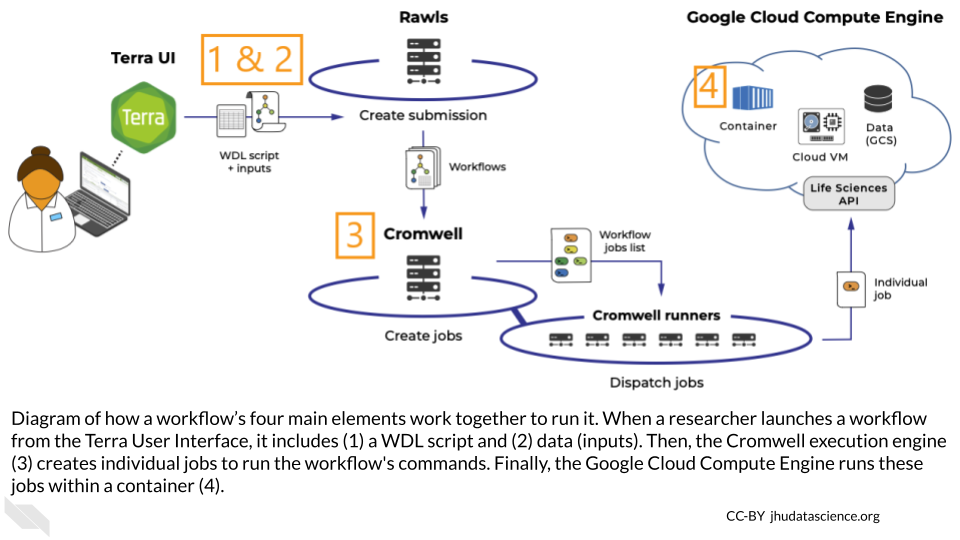
**GitHub**: [Create an account](https://github.com/signup) to build Docker images using GitHub Actions

**Docker Hub**: [Create an account](https://hub.docker.com/signup) to publish Docker images that AnVIL can access

## 5.2 Containerization with Docker

Behind the scenes, a workflow requires four main elements:

1. **Data**
2. A **workflow script** (written in WDL)
3. An **execution engine** to manage the workflow’s jobs
4. A **container** in which to run the workflow



These elements all work together to run the workflow – the container’s role is to control all of the code packages and dependencies used to run the workflow’s WDL script. This is called “**containerization**.”

### 5.2.1 Why is containerization helpful?

Containerization makes it easier to reproduce workflow analyses. For example, if two collaborators are analyzing the same data using different versions of Python, they might get different results. Containerization controls the environments in which you’re running these analyses, saving you from puzzling over incompatible results. Containerization can also ensure that other researchers can replicate your results and apply your tools to their own data. Containerization is even helpful when re-running old code – if your code packages have automatically updated since you last ran the code, using a container may save your code from breaking.

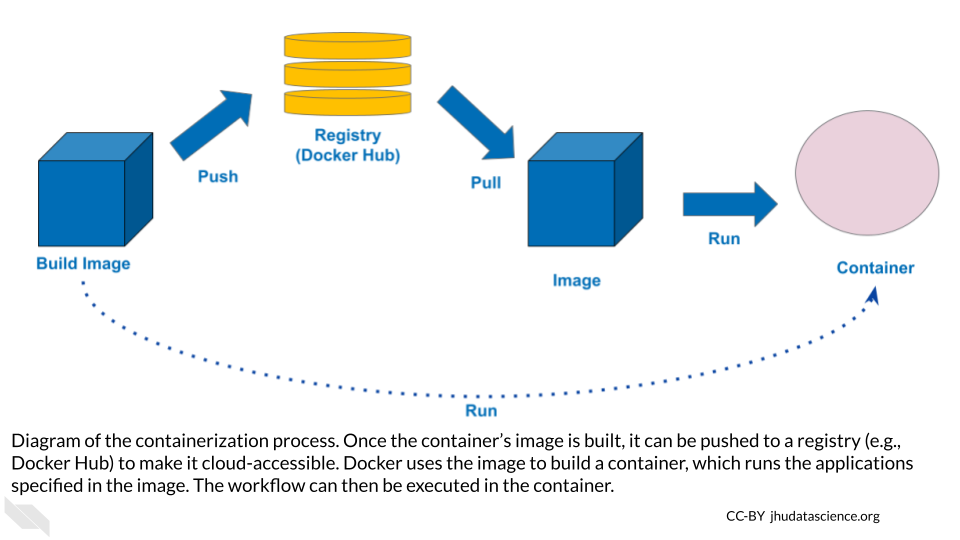
### 5.2.2 Docker containers

You can manage your containers with several services. We’ll focus on **Docker**.

Docker containers have two main components:

1. A **Docker file** that defines the container’s dependencies, environment variables, file system, and applications.
2. A **Docker image** that builds and runs a container, which contains everything defined in the Docker file.

In addition, a **registry** (e.g., DockerHub) is used to share Docker images with others, and to make your Docker image accessible from the Cloud (e.g., in Terra).



### 5.2.3 Using Docker containers in your workflow

A workflow’s container constrains the code that you can write in the WDL script; for example, a WDL script with Python commands must be run in a container that includes Python. So, how can you ensure that you’re using the right container on Terra?

To direct Terra toward the correct container, specify the container’s image in the WDL script. We already did this in the exercise from Chapter 3, by including a “docker” variable in the “runtime” section of the task definition:

runtime {  
 docker: 'ubuntu:latest'  
 }

In this example, we simply specified the most up-to-date Ubuntu image. However, in many cases the best choice is to use a specific image, which won’t change after you’ve written the workflow.

Note that you can use different images in different tasks within your workflow.

## 5.3 Using a Published Docker Image Out of the Box

To set up your workflow’s container, you can build a Docker image from scratch, modify an existing image, or find a published image that you can use out of the box, without any modifications.

This last option is a particularly good option when you only need your container to include a single tool. You can find useful images in the following ways:

### 5.3.1 Google

A good first step is to search for “Docker image” and the name of the software that you want to include in the container on a search engine like Google or Bing.

### 5.3.2 Docker Hub and Quay.io

[Docker Hub](https://hub.docker.com) is an online platform for sharing Docker images. Use the “Explore” menu to filter for suitable images. It’s a good idea to filter for images from trusted sources, including Docker official images and images from verified publishers.

[Quay.io](https://quay.io) is another Docker-sharing platform, with similar functionality.

### 5.3.3 DockerBIO

[DockerBIO](https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6266945) is a Java web application that focuses on Docker images for bioinformatics analyses.This approach requires a bit more setup, but can make it easier to find an image that’s relevant to your work.

### 5.3.4 Next steps

Once you’ve found a published image that suits your needs, clone it to create your own copy. This will ensure that your analysis is reproducible, because the image won’t change unless you edit it.

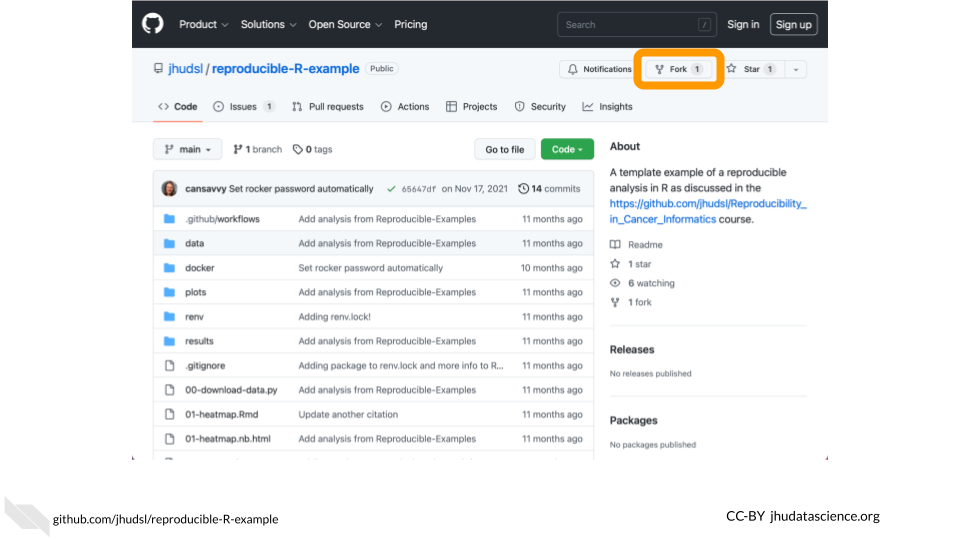
Sections 5.4 and 5.5 show you how to take a step further, and modify a published image for your workflow. To see how to use a Docker image without any modification, skip these sections and start again at section 5.6.

## 5.4 Configure GitHub Actions

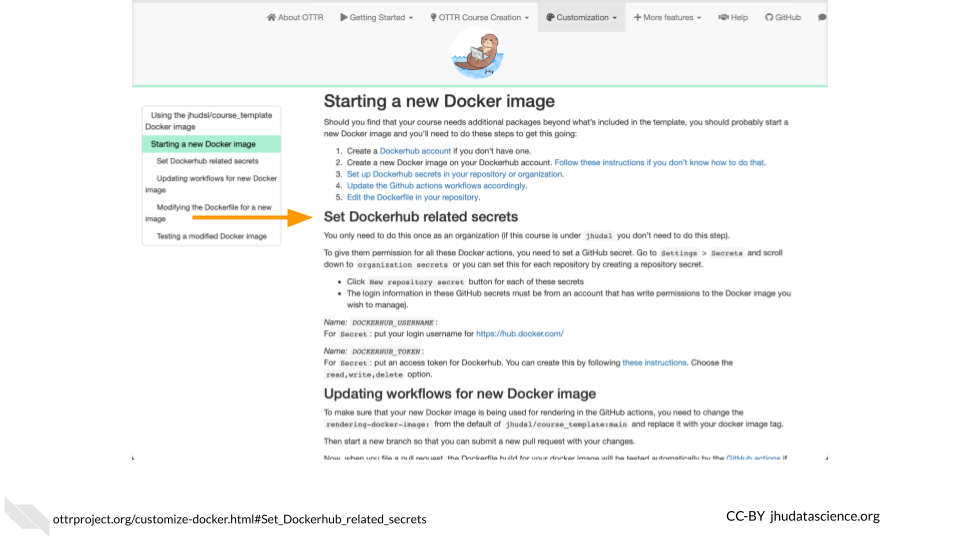
In the next section, we’ll get more in the weeds to show you how to modify a published Docker image to customize it for your workflow.

The GitHub Action [build-and-push-docker-images](https://github.com/marketplace/actions/build-and-push-docker-images) provides a Cloud-based solution to building a Docker image and pushing it to [Docker Hub](https://hub.docker.com). We will start with a repository created for the ITN course [Intro to Reproducibility in Cancer Informatics](https://jhudatascience.org/Reproducibility_in_Cancer_Informatics) that has this GitHub Action configured.

First, fork the [reproducible-R-example](https://github.com/jhudsl/reproducible-R-example) repository.



Your new repository must be configured with the proper credentials (referred to as [Actions secrets](https://docs.github.com/en/actions/security-guides/encrypted-secrets)) to push an image to Docker Hub. Follow the [OTTR Project instructions](https://www.ottrproject.org/customize-docker.html#Set_Dockerhub_related_secrets)) to set DOCKERHUB\_USERNAME and DOCKERHUB\_TOKEN.



Update the .github/workflows/docker-management.yml file with your Docker Hub username and repository. For example, change jhudsl/reproducible-r to my\_username/my\_repository. Note that you need to change this on both line 53 and line 71.



This docker-management.yml file points Docker toward the correct Docker image file and links your GitHub repository with your Docker account.

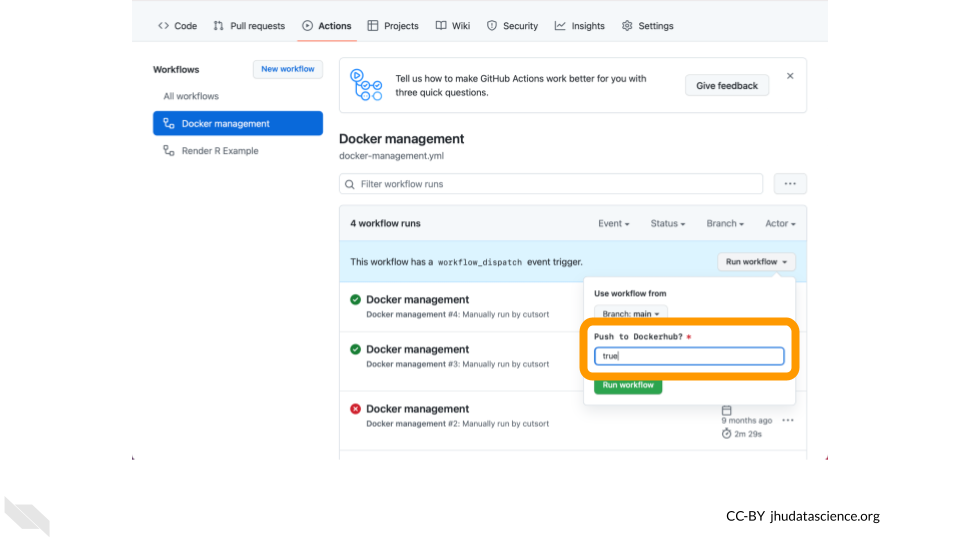
## 5.5 Build Docker image

The docker/Dockerfile file is the Docker image file, which is written in YAML syntax. It typically initializes the container with a base image that provides some basic dependencies, then installs additional software and dependencies as necessary. It may also provide metadata, set up scripts, and define commands to run when the container starts.

For a base image, we’ll start with [condaforge/mambaforge](https://github.com/conda-forge/miniforge-images) which is maintained by condaforge and provides conda on top of ubuntu-20.04. Replace the contents of the docker/Dockerfile file with the following to install [SAMtools via bioconda](https://anaconda.org/bioconda/samtools).

FROM condaforge/mambaforge  
  
RUN conda config \  
 --add channels defaults \  
 --add channels bioconda \  
 --add channels conda-forge  
  
RUN conda install -y samtools

Run the GitHub Action by navigating to the Actions tab, selecting Docker management, and clicking on Run workflow. Note that you must change Push to Dockerhub? to true.



You **must** change Push to Dockerhub? to true for your Docker image to be pushed to Docker Hub.

## 5.6 Use your image with the idxstats WDL

The next step is to modify the WDL script that you wrote in Chapter 4 to run in a container built by this Docker image. Edit your idxstats WDL script to call samtools idxstats on a bam file to specify your customized Docker image.

The result should look something like the version below, except that your version should point to the name of your own Docker image’s repository, rather than cutsort/test:

version 1.0  
workflow samtoolsIdxstats {  
 input {  
 File bamfile  
 }  
 call idxstats {  
 input:   
 bamfile = bamfile  
 }  
 output {  
 File results = idxstats.idxstats  
 }  
}  
  
task idxstats {  
 input {  
 File bamfile  
 }  
 command <<<  
 samtools index ~{bamfile}  
 samtools idxstats ~{bamfile} > idxstats.txt  
 >>>  
 output {  
 File idxstats = "idxstats.txt"  
 }  
 runtime {  
 docker: 'cutsort/test'  
 }  
}

The next step is to add this WDL script to the Broad Methods Repository in order to run the workflow in Terra.

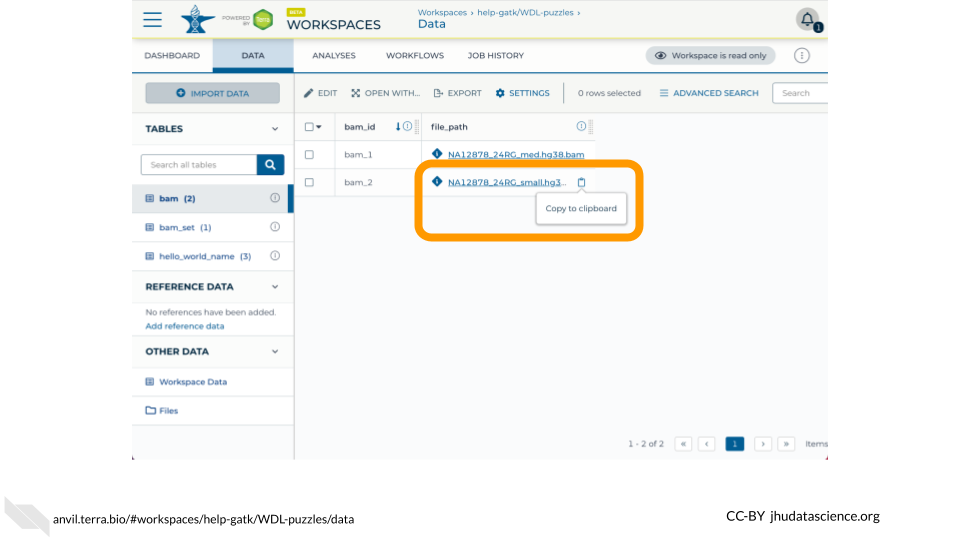
Follow the steps outlined in Chapter 3 to access the Broad Methods Repository and create a new method.

Then, copy or upload your WDL script into the new method and export it to your cloned version of the WDL-puzzles workspace.

## 5.7 Run idxstats workflow

Once you’ve exported the workflow to your AnVIL-powered-by-Terra workspace, open the workflow.

Select “Run workflow(s) with inputs defined by data table”, select the “bam” table, and select one row of the data table for a test run.



Fill in the column that stores the path to the input bam file (this.file\_path).

In the Outputs tab, give the output column a name by typing this.COLUMN\_NAME (ex. this.idxstats\_output).

Then click “run” to run the workflow, and monitor its progress. Once the workflow has finished running, you should see a new column in the “bam” table with a link to a .txt file with the QC metrics output by the workflow.

## 5.8 Best practices for using Docker

The WDL Analysis Research Pipelines (WARP) [GitHub page](https://github.com/broadinstitute/warp-tools/blob/develop/README.md) lists some recommendations for setting up Docker containers. Here are a few highlights:

* **Make sure that your image isn’t too large**, to avoid using unnecessary compute resources. Start with small images (e.g., Alpine) and include as few run steps as possible in the image file.
* Make dockers **publicly accessible**.

## 5.9 Further Reading

If you’re interested in a deeper dive into this chapter’s topics, check out these optional articles:

* To learn more about how to use Docker to create and store images, read 1[Docker/container overview](https://support.terra.bio/hc/en-us/articles/360037340472-Docker-container-overview)
* For information on using Docker to develop images locally, read [how to install docker and test that it works](https://support.terra.bio/hc/en-us/articles/360036000631-How-to-install-Docker-and-test-that-it-works)
* [How to run GATK in a Docker container](https://support.terra.bio/hc/en-us/articles/360036007791-How-to-run-GATK-in-a-Docker-container)
* [Docker Image Publishers’ Tips](https://support.terra.bio/hc/en-us/articles/4409141003547-Docker-Image-Publishers-Tips)

# 6 Join Discourse

# About the Authors

These credits are based on our [course contributors table guidelines](https://github.com/jhudsl/DaSL_Course_Template_Bookdown/wiki/How-to-give-credits).

| Credits | Names |
| --- | --- |
| **Pedagogy** |  |
| Lead Content Instructor(s) | [FirstName LastName](link%20to%20personal%20website) |
| Lecturer(s) (include chapter name/link in parentheses if only for specific chapters) - make new line if more than one chapter involved | Delivered the course in some way - video or audio |
| Content Author(s) (include chapter name/link in parentheses if only for specific chapters) - make new line if more than one chapter involved | If any other authors besides lead instructor |
| Content Contributor(s) (include section name/link in parentheses) - make new line if more than one section involved | Wrote less than a chapter |
| Content Editor(s)/Reviewer(s) | Checked your content |
| Content Director(s) | Helped guide the content direction |
| Content Consultants (include chapter name/link in parentheses or word “General”) - make new line if more than one chapter involved | Gave high level advice on content |
| Acknowledgments | Gave small assistance to content but not to the level of consulting |
| **Production** |  |
| Content Publisher(s) | Helped with publishing platform |
| Content Publishing Reviewer(s) | Reviewed overall content and aesthetics on publishing platform |
| **Technical** |  |
| Course Publishing Engineer(s) | Helped with the code for the technical aspects related to the specific course generation |
| Template Publishing Engineers | [Candace Savonen](https://www.cansavvy.com/), [Carrie Wright](https://carriewright11.github.io/) |
| Publishing Maintenance Engineer | [Candace Savonen](https://www.cansavvy.com/) |
| Technical Publishing Stylists | [Carrie Wright](https://carriewright11.github.io/), [Candace Savonen](https://www.cansavvy.com/) |
| Package Developers ([Leanbuild](https://github.com/jhudsl/leanbuild)) | [John Muschelli](https://johnmuschelli.com/), [Candace Savonen](https://www.cansavvy.com/), [Carrie Wright](https://carriewright11.github.io/) |
| **Art and Design** |  |
| Illustrator(s) | Created graphics for the course |
| Figure Artist(s) | Created figures/plots for course |
| Videographer(s) | Filmed videos |
| Videography Editor(s) | Edited film |
| Audiographer(s) | Recorded audio |
| Audiography Editor(s) | Edited audio recordings |
| **Funding** |  |
| Funder(s) | Institution/individual who funded course including grant number |
| Funding Staff | Staff members who help with funding |

## ─ Session info ───────────────────────────────────────────────────────────────  
## setting value   
## version R version 4.0.2 (2020-06-22)  
## os Ubuntu 20.04.5 LTS   
## system x86\_64, linux-gnu   
## ui X11   
## language (EN)   
## collate en\_US.UTF-8   
## ctype en\_US.UTF-8   
## tz Etc/UTC   
## date 2023-10-10   
##   
## ─ Packages ───────────────────────────────────────────────────────────────────  
## package \* version date lib source   
## assertthat 0.2.1 2019-03-21 [1] RSPM (R 4.0.5)   
## bookdown 0.24 2023-03-28 [1] Github (rstudio/bookdown@88bc4ea)   
## cachem 1.0.7 2023-02-24 [1] CRAN (R 4.0.2)   
## callr 3.5.0 2020-10-08 [1] RSPM (R 4.0.2)   
## cli 3.6.1 2023-03-23 [1] CRAN (R 4.0.2)   
## crayon 1.3.4 2017-09-16 [1] RSPM (R 4.0.0)   
## desc 1.2.0 2018-05-01 [1] RSPM (R 4.0.3)   
## devtools 2.3.2 2020-09-18 [1] RSPM (R 4.0.3)   
## digest 0.6.25 2020-02-23 [1] RSPM (R 4.0.0)   
## ellipsis 0.3.1 2020-05-15 [1] RSPM (R 4.0.3)   
## evaluate 0.20 2023-01-17 [1] CRAN (R 4.0.2)   
## fastmap 1.1.1 2023-02-24 [1] CRAN (R 4.0.2)   
## fs 1.5.0 2020-07-31 [1] RSPM (R 4.0.3)   
## glue 1.4.2 2020-08-27 [1] RSPM (R 4.0.5)   
## htmltools 0.5.5 2023-03-23 [1] CRAN (R 4.0.2)   
## knitr 1.33 2023-03-28 [1] Github (yihui/knitr@a1052d1)   
## magrittr 2.0.3 2022-03-30 [1] CRAN (R 4.0.2)   
## memoise 2.0.1 2021-11-26 [1] CRAN (R 4.0.2)   
## pkgbuild 1.1.0 2020-07-13 [1] RSPM (R 4.0.2)   
## pkgload 1.1.0 2020-05-29 [1] RSPM (R 4.0.3)   
## prettyunits 1.1.1 2020-01-24 [1] RSPM (R 4.0.3)   
## processx 3.4.4 2020-09-03 [1] RSPM (R 4.0.2)   
## ps 1.4.0 2020-10-07 [1] RSPM (R 4.0.2)   
## R6 2.4.1 2019-11-12 [1] RSPM (R 4.0.0)   
## remotes 2.2.0 2020-07-21 [1] RSPM (R 4.0.3)   
## rlang 1.1.0 2023-03-14 [1] CRAN (R 4.0.2)   
## rmarkdown 2.10 2023-03-28 [1] Github (rstudio/rmarkdown@02d3c25)  
## rprojroot 2.0.3 2022-04-02 [1] CRAN (R 4.0.2)   
## sessioninfo 1.1.1 2018-11-05 [1] RSPM (R 4.0.3)   
## stringi 1.5.3 2020-09-09 [1] RSPM (R 4.0.3)   
## stringr 1.4.0 2019-02-10 [1] RSPM (R 4.0.3)   
## testthat 3.0.1 2023-03-28 [1] Github (R-lib/testthat@e99155a)   
## usethis 1.6.3 2020-09-17 [1] RSPM (R 4.0.2)   
## withr 2.3.0 2020-09-22 [1] RSPM (R 4.0.2)   
## xfun 0.26 2023-03-28 [1] Github (yihui/xfun@74c2a66)   
## yaml 2.2.1 2020-02-01 [1] RSPM (R 4.0.3)   
##   
## [1] /usr/local/lib/R/site-library  
## [2] /usr/local/lib/R/library

# References