Creating a study using Ulysses

Contents


```
library(Ulysses)
```

1 Introduction

Running an OHDSI study contains lots of organizational complexity in terms of organizing code and proper documentation to communicate the code. While examples for constructing OHDSI studies have been presented, for example the SOS challenge, there is no clear workflow towards developing an OHDSI study as a piece of software available in a github repository. The OHDSI community would benefit from a workflow tool that will help standardize the development of network studies and improve its organization. This gap led to the development of a new R package called `Ulysses` (Useful Learning Yielded Structuring and Setting Epidemiology Studies) dedicated towards assisting in the development of an OHDSI study.

Ulysses draws inspiration from the R package `usethis`, which is a workflow tool used for the development of R packages. Similar to OHDSI studies, there are several administrative steps and procedures required to develop a stable and transparent R package. `usethis` helps R programmers navigate R package development by supplying functions that automate simple tasks or other useful steps needed to meet this goal. Ulysses can provide a similar solution for OHDSI studies, improving the organization, communication and development of OHDSI studies by supplying simple functions to guide developers towards a study that is transparent, robust and reproducible. The OHDSI community would benefit from a tool that enforces standards and organization in OHDSI studies because it makes it easier for study nodes to execute network studies from a recognizable structure and provide guidance to new researchers seeking to build an OHDSI study if they follow a common workflow. In this software demo, we will showcase an example of how Ulysses can be used to start a new OHDSI study and help initiate necessary tasks for organizing and communicating the study to the OHDSI data network.

In this vignette we walk-through how to build an OHDSI study using the `Ulysses` package. We review:

- Initializing a study in R
- Adding repo documentation

- Handling database credentials
- Adding R scripts
- Providing documentation about the study

2 Initializing an OHDSI study in R

Once you have downloaded the `Ulysses` package, you can begin creating an OHDSI study. To do this, you can run code as shown in the block below:

```
Ulysses::newOhdsiStudy(
  path = here::here("my_ohdsi_study"),
  author = "Ulysses S. Grant",
  type = "Characterization",
  directory = "[my_directory]",
  open = TRUE
)
```

This function will print some information to your console about start-up tasks and open an R project in a new session, for details on Rstudio projects see link.

In the new R session, we are directed to a clean R studio session. In the files pane you will see a directory structure as depicted in the image below. For more details about the directory structure refer to the *Introduction to Ulysses Directory* vignette in this package. Congratulations! You have started an OHDSI study!

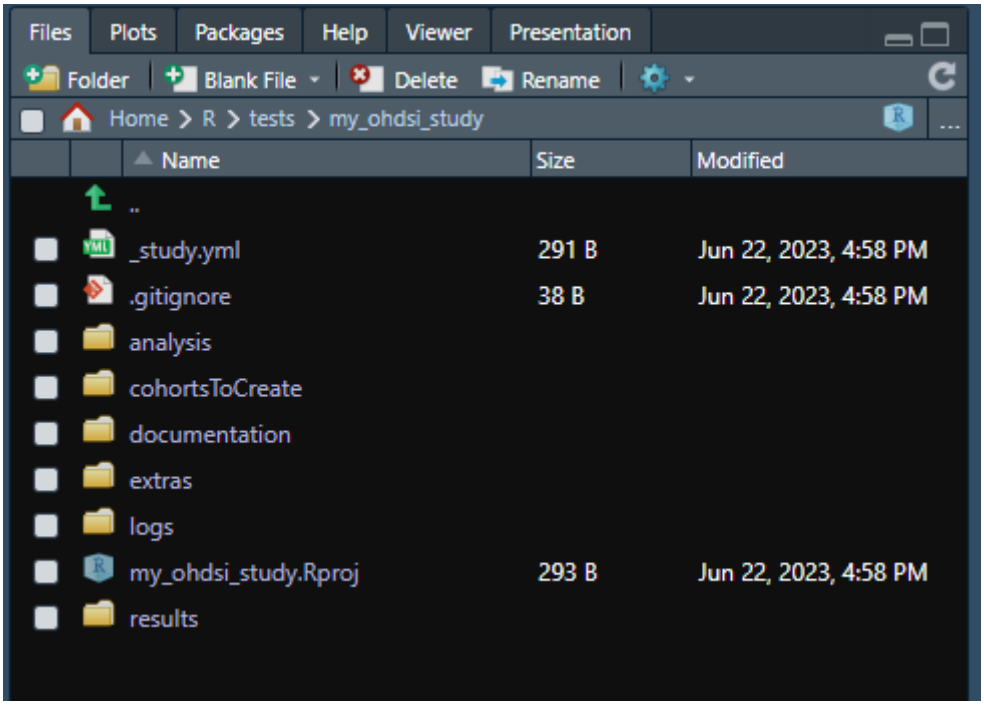

Figure 1: Ulysses Style OHDSI Study Directory

3 Adding Repository Documentation

3.1 The *README* file

With your new OHDSI study created, you need to begin adding documentation for the repository. The first file we usually create is the repository README. For those unfamiliar with code development, the README file is a standard file used to introduce a describe the the project. Think of the README as your cover page. It is the first file users see when they navigate to the github page. All OHDSI projects require a README file, thus `Ulysses` provides a function (`makeReadMe`) that initializes it.

```
Ulysses::makeReadMe()
```

The readme file is initailized using information found in the `_study.yml` file. A typical OHDSI study readme has a section of meta information as seen below:

- **Study Lead**: Ulysses S. Grant
- **Analytics Use Case**: Characterization
- **Study Start Date**: 04-27-1822
- **Study End Date**: 07-23-1885
- **Study Tags**: None
- **Protocol**: Unavailable
- **Publications**: Unavailable
- **Results Explorer**: Unavailable

Following the Meta section, the user will need to provide a description of the study and information on the databases used in the study. Finally, the README provides links to important study information including the protocol, how to run and contributions files. Users can add as much as they please to the README file, `Ulysses` offers suggested guidance for starting it.

Another feature offered by `Ulysses` is support for svg badges. These simple badges appear at the top of the README and provide useful information about the study. Badges seen in OHDSI studies include a study status badge and badges versioning the CDM and OMOP vocabulary. `Ulysses` will provide more support and documentation for badges in the future.

3.2 The *NEWS* file

Another common file in software repositories is the NEWS file. The purpose of the NEWS file is to track changes to the software over time. Of course, version control software such as github keeps a record of the iterations of the study as it is developed, however it is helpful to have a "plain english" file that explains what has happened as the study has developed over time. NEWS files are often maintained via a semantic versioning system. OHDSI studies are not entirely software, however it is important to maintain order over the development of the technical pieces of the study. `Ulysses` offers a simple function that initiates this file, `makeNews`.

```
Ulysses::makeNews()
```

4 Handling database credentials

An important aspect of running an OHDSI study is handling credentials to connect to the Database Management System (DBMS) that hosts the OMOP data. Security of these credentials is very important. `Ulysses` offers support on handling these credentials. Credentials needed to run a study are as follows:

- **dbms**: the name of the dbms used to host the OMOP data
- **user**: your user name needed to access the data in the dbms
- **password**: your password needed to access the data in the dbms
- **connectionString**: this string sets the connection to the database. It will typically look something like: `jdbc:<dbms>://<server url>:<dbms port>/<database>`. This string will vary from site to site.

Contact your system administrator to get the information you need for this credential

- **cdmDatabaseSchema**: a name that defines where the cdm tables sit for a particular database in the dbms. `Database` defines the name of the database where the omop data sits and the `Schema` defines where the cdm is within that database. In databases like sql server or snowflake this variable is often separated by a "." i.e. '.
- **vocabDatabaseSchema**: a name that defines where the vocabulary tables sit for a particular database in the dbms. The vocabDatabaseSchema is typically the same as the cdmDatabaseSchema, although same sites split the vocabulary into a different area.
- **workDatabaseSchema**: a name that defines where the work tables sit for a particular database in the dbms. The work section (referred to synonymously with scratch or write) is an area where the user is given read and write access in the database. The cdmDatabaseSchema is typically only a read-only schema, so users do not corrupt information. The workDatabaseSchema is a dedicated area where researchers can build cohort tables or other intermediary tables needed for studies. Contact your database administrator to ensure you have a dedicated workDatabaseSchema. Note this should be separate from the database schema dedicated to ATLAS.

Often times sites require additional credentials such as database roles or other parameters used to define tempEmulationSchemas, for example. You can add these parameters manually in the `Ulysses` workflow, shown later. Be sure to contact your database administrator to have all these credentials handy before running a study.

Now that you have collected your credentials we need to store them somewhere secure to be accessed continuely within the study. `Ulysses` does this using a *config.yml* file. Yaml is a simple, readable data-serialization language used to provide settings or configurations for applications. For those familiar with R, the *config.yml* file works similarly to an `.Renviron` file. But wait, there is more to the *config.yml* file! The credentials are further protected via `keyring` API. `keyring` stores your credentials behind a password. We use `keyring` and `config` together to ensure that users do not accidentally expose passwords when working on a study. `Ulysses` provides a function that initializes the *config.yml* file:

```
Ulysses::makeConfig(block = "example", database = "synpuf_110k")
```

When a user creates a new `config.yml` file it is added to `.gitignore` to ensure it is not committed to a github repository. The config file shows each credential connected to the block name.

```
# Config File for my_ohdsi_study

default:
  projectName: my_ohdsi_study

# Config block for example

example:
  databaseName: example
  cohortTable: my_ohdsi_study_example
  dbms: !expr keyring::key_get('example_dbms', keyring = 'my_ohdsi_study')
  user: !expr keyring::key_get('example_user', keyring = 'my_ohdsi_study')
  password: !expr keyring::key_get('example_password', keyring = 'my_ohdsi_study')
  connectionString: !expr keyring::key_get('example_connectionString', keyring = 'my_ohdsi_study')
  cdmDatabaseSchema: !expr keyring::key_get('example_cdmDatabaseSchema', keyring = 'my_ohdsi_study')
  vocabDatabaseSchema: !expr keyring::key_get('example_vocabDatabaseSchema', keyring = 'my_ohdsi_study'
  workDatabaseSchema: !expr keyring::key_get('example_workDatabaseSchema', keyring = 'my_ohdsi_study')
```

If we wanted to add another credential in this block we can add new line beneath `workDatabaseSchema`. Be sure to keep the indentation and add an extra space after. To set up the keyrings, `Ulysses` also supplies a helper script to set this up. Follow the script to set up the keyring credentials so that the *config.yml* can be properly used.

```
Ulysses::makeKeyringSetup(configBlock = "example", database = "synpuf_110k")
```

5 Adding R scripts

One way we can add analytical scripts to `Ulysses` is by using the line below

```
Ulysses::makeAnalysisScript(scriptName = "buildCohorts")
```

This function will create a new R script written to the folder `analysis/studyTasks`. The R script will have a prefix of a number indicating the order in which the script should be executed. The first script for example will follow the syntax **01__**. `Ulysses` will automatically open this file to begin editing.

Each analytical task is formatted with the following sections:

A) File Info -> includes information about the file
B) Dependencies -> a list of libraries, options and source files required to run script
C) Connection -> set up connection to dbms
D) Variables -> set variables (parameters) needed for the script
E) Script -> the script to run
F) Session Info -> a summary of the session info and clean up

To complement the analysis scripts, we can also create internal files. These are R files that contain functions or other code that needs to be sourced in the analysis. These files are saved in the `analysis/private` folder. Making an internal function can be done by the following:

```
Ulysses::makeInternals(internalsName = "buildCohorts")
```

The R package `Strategus` helps execute OHDSI analytics. Analytical modules are executed via `Strategus`. This paradigm to OHDSI studies can also be handled by `Ulysses`. More support for using `Strategus` with `Ulysses` will be explored in future releases.

6 Adding Documenation

An important part of a study repository is communicating its contents to study nodes. A study node in a network study must understand the scientific components of the study, how to run the study locally and how to contribute towards the study. `Ulysses` provides functions to help develop documentation to communicate the study to participants in the OHDSI network.

6.1 Analysis Specification

A study requires a protocol or a study analysis plan (SAP), specifying the scientific components of the study. These documents specify the study design, definition of the target, exposure and outcome cohorts and how the data analysis is conducted. This documentation is important because it provides guidance and rationale for the study. Many study nodes also require submission of documentation to an institutional review board to justify execution of the study. `Ulysses` provides templates for an OHDSI protocol, PASS protocols and a suggested format for a SAP. Below are examples of building these templates:

```
Ulysses::makeOhdsiProtocol()
Ulysses::makePassProtocol()
Ulysses::makeStudySAP()
```

6.2 How to Run

Next, a repository needs to communicate how to run the study. This document informs study nodes on technical requirements needed to run the study, provides instructions on installations, and the process of

running the study. OHDSI studies are designed in several different ways, so it is important that the developer explains how this study should be run. Below is an example of how to create the `howToRun.md` file:

```
Ulysses::makeHowToRun(org = "Ohdsi", repo = "my_ohdsi_study")
```

6.3 Contributing

Finally, the repository needs to communicate how study nodes can contribute to the study. This include information on posting results, asking for help and code of conduct. Contribution guidelines are standard documentation of software and would be helpful additions to OHDSI studies. To create a template contribution guideline follow the example below:

```
Ulysses::makeContributionGuidelines()
```