CSHS Workshop: R for hydrologists - building packages

CWRA 2022

Kevin Shook Paul Whitfield Daniel Moore

Canadian Society for Hydrological Sciences (CSHS)

June 4, 2022

CSHS SCSH
Canadian Society for Hydrological Sciences
Société Canadienne des Sciences Hydrologiques
Canadian Water Resources Association
Association Canadienne des Ressources Hydriques

Why create a package?

- The best way to distribute **R** code
- Makes your code reproducible
  - makes code reusable
  - improves code quality
  - takes care of dependencies
  - self-documenting
  - should work for anyone, on any computer

Building a package

- All components are text files
  - You could build them manually
- **DON'T!**
- Use the package **devtools**
  - makes it *much* easier
- Also, install packages **roxygen2, rmarkdown**
- Need LaTex installed to create manuals
- Also, make sure to have **git** installed on your system

Mandatory package components

- 2 Files
  - DESCRIPTION
  - NAMESPACE
- 2 directories are mandatory
  - /R – contains code .R files
  - /man – contains documentation .Rd files
- may have other directories

DESCRIPTION

- Contains package description
- Has to have a specific format
- Has to indicate the packages required by your package

DESCRIPTION example from CSHShydRology

```
Type: Package
Package: CSHShydRology
Title: Canadian Hydrological Analyses
Version: 1.2.1
Date: 2022-04-17
Authors:...
Author:...
Maintainer: Kevin Shook <kevin.shook@usask.ca>
Description: A collection of user submitted functions to aid in the analysi
License: AGPL-3
URL: https://github.com/CSHS-hydRology/CSHShydRology
Depends:
    R (>= 4.0.0)
Imports:
   ...
VignetteBuilder: knitr
```

NAMESPACE

- Contains detailed information about imports and exports of each function
- Do NOT create or edit this file
  - **roxygen2** will automatically create and maintain it

R directory

- Contains the **R** code
- Code must be written as functions
- Each function must be in a separate file
  - file name is same as function name
  - file extension must be **.R**

man directory

- Contains the documentation files
- Creates the help system for the package
- Also creates the manual
- Each .R file has a .Rd file in **man**
  - Don't create these files manually

roxygen2

- Used by **devtools**, installed by it
- Automatically creates the .Rd files
  - uses comments at the beginning of each **.R** file

Example

- All lines begin with #'
- First line contains a 1 line description of the file
- Should not end with a period!
- Example:

Tags

Formatting

- Documentation can include formatting codes

Example

Package function

- You should create a function that has the name of your package
- Example: WISKIr-package.R
- Contains information to create NAMESPACE

Example

Other folders

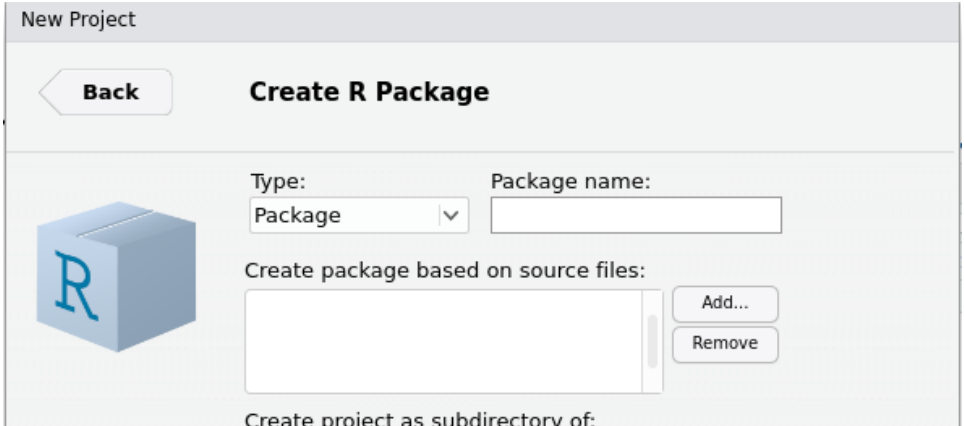
- You may see these folders in packages:
  - /data, data files used by the package
  - /vignettes, documentation written in Markdown
  - /inst, contains the file CITATION showing how to cite the package
  - /src, source code written in C, C++ or Fortran

Workflow

1. Create the package
2. Add code
3. Build package
4. Check package
5. Create package file

1. Creating the package

- Create a new project in a new directory
  - **File|New Project**
  - select **R Package**
  - then give your package a name and a location
- Make sure to use git!

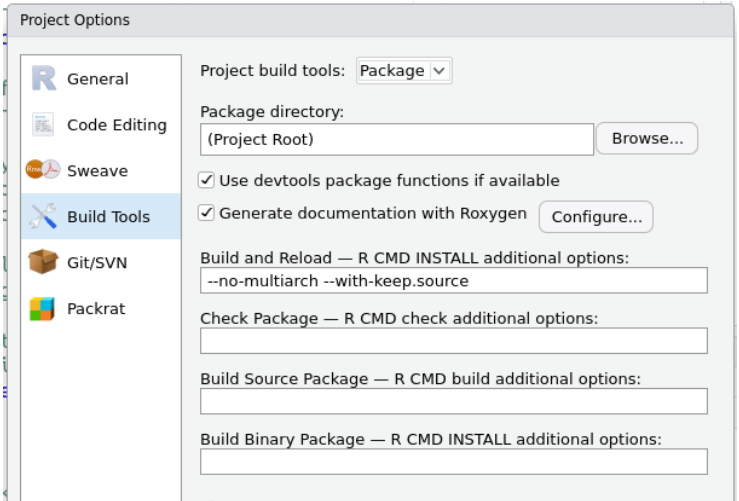


New package

- R studio will create all of the files and directories
  - /R
  - /man
  - DESCRIPTION
  - NAMESPACE
- Also creates a sample file **hello.R** in /R
- Adds folders and files for git

Setting up roxygen

- Not enabled by default
- Set it up using **Tools|Project Options**



2. Adding R code

- Put your R code files in /R
- Must be functions
  - one function per file
- Add the roxygen skeleton to your code for each file
  - **Code|Insert Roxygen Skeleton**
- Fill in skeleton

Example

Converting your R code

- You will ned to make some changes to your code
- Don't use the **library()** function to load packages
  - package importation handled by `NAMESPACE`
- Specify the name of the package in every function (outside of your package and Base R) call
  - syntax is **package::function**

3. Building package

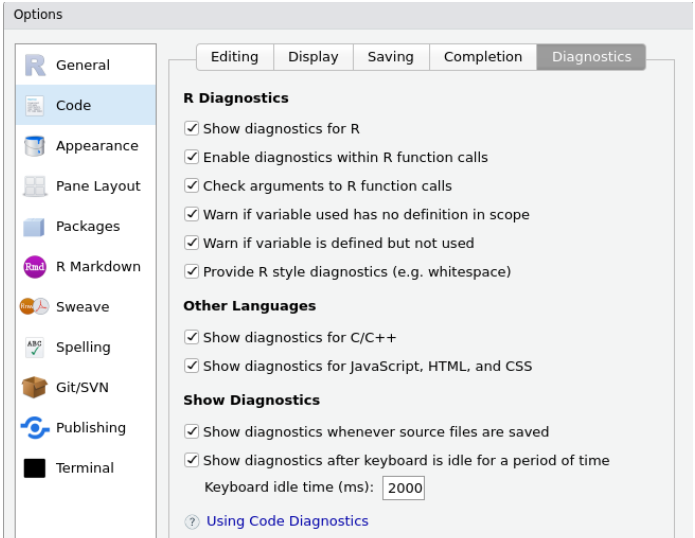
- Use command **Build | Clean and Rebuild**
- Expect to get error messages!
- Fix until package builds
- If the package builds, it will be added to your list of packages

4. Checking the package

- Just because a package can build, doesn't mean that it is good!
- Use command **Build|Check**
  - does a detailed check of entire package
  - tries to run your examples
  - *very* picky
  - you will get many, many errors and warnings

Writing better code

- Turn on Code Diagnostics using **Tools|Global Options**



5. Creating the package file

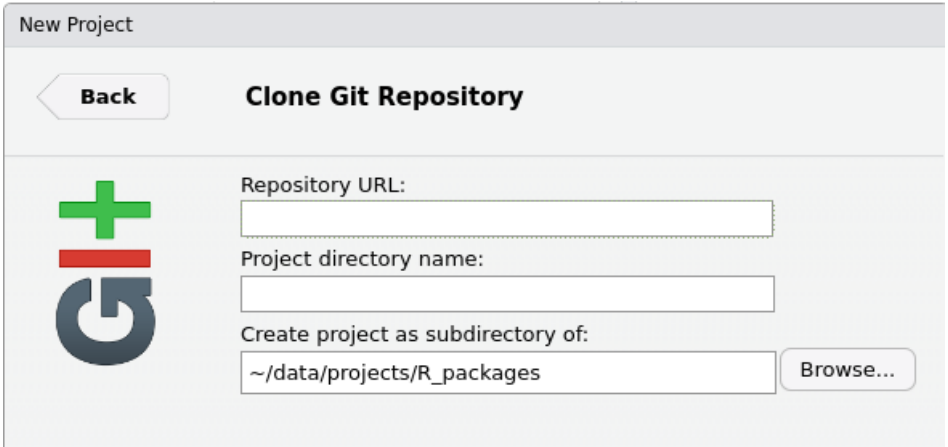
- 2 options:
  - **Build** | **Build Source Package** - contains source code (all languages)
  - **Build** | **Build Binary Package** - contains compiled Fortran, C, C++ code
- Reason is that Windows computers usually don't have compilers
- If just using **R** code, make it a source package
- If you are using Fortran, C, C++, create both types

Building the manual .pdf

- When the package is built, should also create the .pdf
- Must have LaTex installed
- For some reason, this doesn't work for me
  - have to do it manually
  - type in this command in the **R** console:

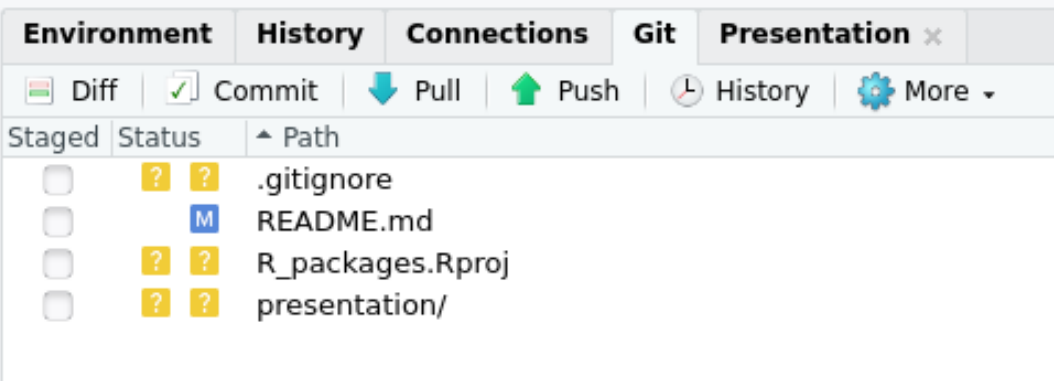
Copying an existing repository

- You can create a package by cloning an existing git repository
- **File|New Project** and select **Version Control**
- Works with GitHub - just enter the url
- Try this one: https://github.com/CentreForHydrology/WISKIr



Using git

- Every time you change your code, commit the changes to your git repository
- If you make a mistake, you can go back to an older version



Misc.

- Make sure you specify a licence for your code
  https://choosealicense.com/
- Don't forget to update the version number in DESCRIPTION
  - version $< 1.0.0$ – not ready for outside use
  - use major.minor.patch numbers
    https://semver.org/
- Remember to update the date in DESCRIPTION

Unit tests

- New feature, part of **devtools**
  - tests the results of functions
  - compares function outputs to known values
  - allows automated testing of functions