



# R PACKAGE BASICS



# R Packages

- The most complete, systematic way of sharing R code (with strict rules!)
- Bundles of code, documentation, data, and tests
- Installed with `install.packages()`, loaded with `library()`, and provide help with `?`.
- Allow for transparent, reproducible workflows
- Require a lot of work up front, and continuing work to maintain

# R Package Structure



NAMESPACE



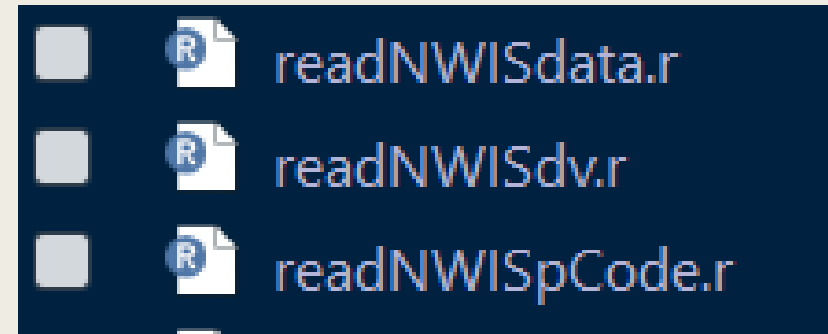
README.md



DESCRIPTION

# R/: R code

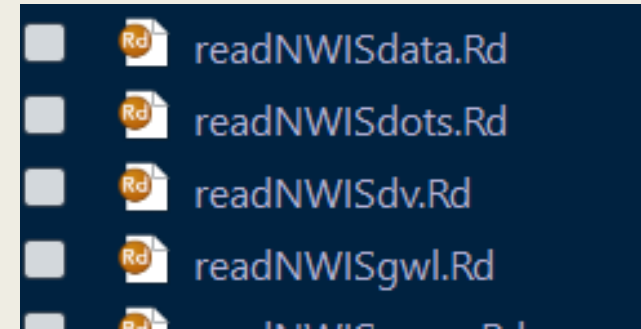
- The R folder is where all R code (in the form of functions) is stored.
- Related functions are organized into scripts with descriptive file names. Each script can contain one or more functions.
- When the package is built and loaded with `library()`, the scripts in the R folder are run, which creates the functions.



```
readNWISdata <- function(..., asDate  
  tz <- match.arg(tz, OlsonNames())  
  valuesList <- readNWISdots(...)  
  service <- valuesList$service  
  values <- sapply(valuesList$values
```

# man/: code documentation

- The *man* (short for manual) folder is where the documentation for the functions is stored.
- These files produce the help pages you see when you use ? or help()
- Each function documentation
  - *Says what the function does*
  - *Explains what the arguments are*
  - *Explains what the function returns*
  - *Provides additional details if needed*
- The roxygen2 package makes writing function documentation easy. **Don't write your documentation manually.**



```
% Generated by roxygen2: do not edit by hand
% Please edit documentation in R/readNWISdata.R
\name{readNWISdata}
\alias{readNWISdata}
\title{General Data Import from NWIS}
\usage{
readNWISdata(..., asDateTime = TRUE, convertType
}
\arguments{
\item{\code{dots}}{see \url{https://waterservices.usgs
list of options. A
list of arguments can also be supplied. One impo
Possible values are "iv"
(for instantaneous) "iv recent" (for instantane
```

# DESCRIPTION

- The DESCRIPTION file gives information about the package:
  - The author(s) and maintainers of the package
  - The version number
  - A description of what the package does
  - Package dependencies (other packages it needs to function)



DESCRIPTION

```
1 Package: dataRetrieval
2 Type: Package
3 Title: Retrieval Function
4 Version: 2.7.5.9002
5 Authors@R: c(
6   person("Laura", "De
7   email = "ldecicco@u
8   comment=c(ORCID="00
9   person("Robert", "H
10  email = "rhirsch@us
11  comment=c(ORCID="00
12  person("David", "Lon
13  person("Jordan", "R
14  email = "jread@usgs
15
```

# NAMESPACE

- The NAMESPACE file describes what's exported from your package and what's imported
- Exported functions are the functions in your package that are available to the user.
- Imported functions are functions from other packages that your package uses.
- Don't write your NAMESPACE file manually. The roxygen2 package does this for you.



```
1 | # Generated by roxygen2: do not edit by hand
2
3 export(addwaterYear)
4 export(calcwaterYear)
5 export(checkWQdates)
6 export(constructNWISURL)
7 export(constructUseURL)
8 export(constructWQPURL)
9 export(countyCd)
10 export(countyCdLookup)
11 export(getWebServiceData)
12 export(importNGWMN)
13 export(importPDR1)
```

# README.md

- The README file is an informal markdown file that gives you a chance to tell users or potential users what you want them to know about the package such as:
  - Installation instructions
  - Recommended workflow
- The README file is especially useful for packages on GitHub. The README file is rendered on the [main page of the repository](#).



dataRetrieval

build passing coverage 89% USGS Research CRAN 2.7.5 downloads 1930/month downloads 70K

Retrieval functions for USGS and EPA hydrologic and water quality data.

For complete tutorial information, see:

<https://usgs-r.github.io/dataRetrieval>

<https://owi.usgs.gov/R/dataRetrieval.html>

### Sample Workflow

#### USGS

```
library(dataRetrieval)
# Choptank River near Greensboro, MD
siteNumber <- "01491000"
ChoptankInfo <- readNWISsite(siteNumber)
parameterCd <- "00600"

#Raw daily data:
rawDailyData <- readNWISdv(siteNumber,parameterCd,
  "1980-01-01","2010-01-01")

# Sample data Nitrate:
parameterCd <- "00618"
qwData <- readNWISqw(siteNumber,parameterCd,
  "1980-01-01","2010-01-01")

pCode <- readNWISpCode(parameterCd)
```



# vignettes/: long form documentation

- Vignettes are long form documentation that mixes text and code in a markdown document.
- They explain specific aspects of a package and explain how the package is to be used.
- You can browse vignettes for a specific package by using, for example: `browseVignettes("dplyr")`



## Introduction to dplyr

When working with data you must:

- Figure out what you want to do.
- Describe those tasks in the form of a computer program.
- Execute the program.

The dplyr package makes these steps fast and easy:

- By constraining your options, it helps you think about your data manipulation challenges.
- It provides simple “verbs”, functions that correspond to the most common data manipulation tasks, to help you translate your thoughts into code.
- It uses efficient backends, so you spend less time waiting for the computer.

This document introduces you to dplyr’s basic set of tools, and shows you how to apply them to data frames. dplyr also supports databases via the dbplyr package, once you’ve installed, read `vignette("dbplyr")` to learn more.

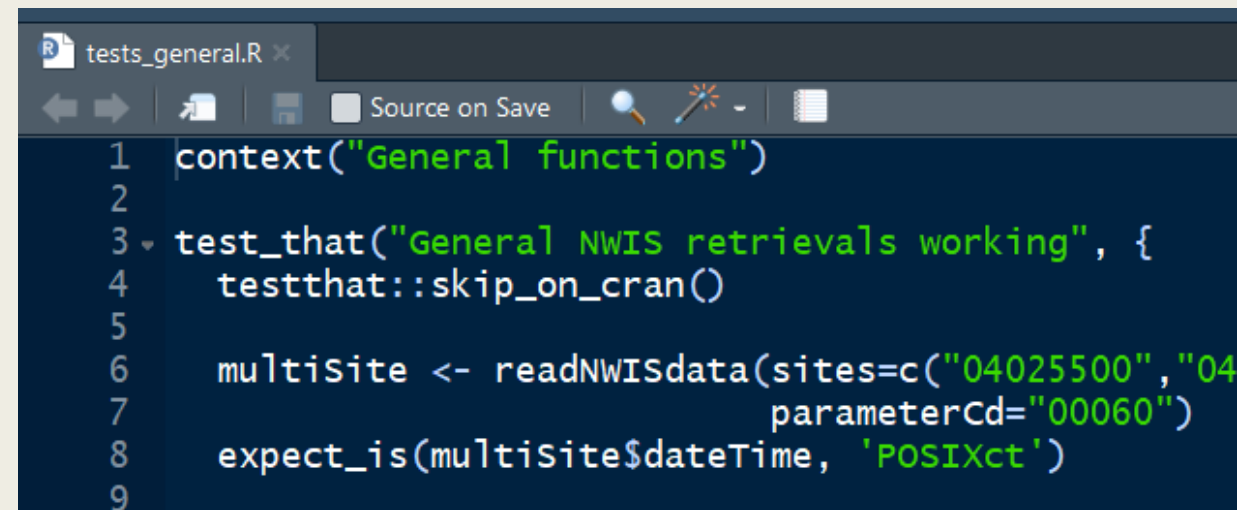
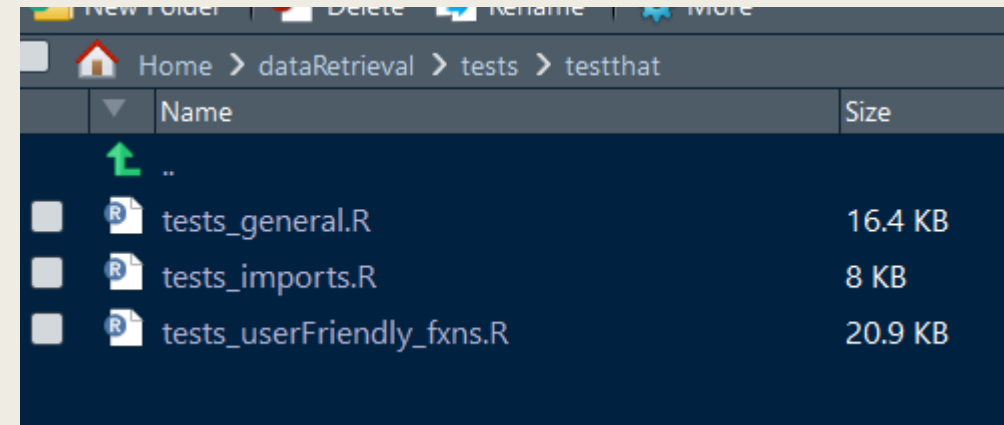
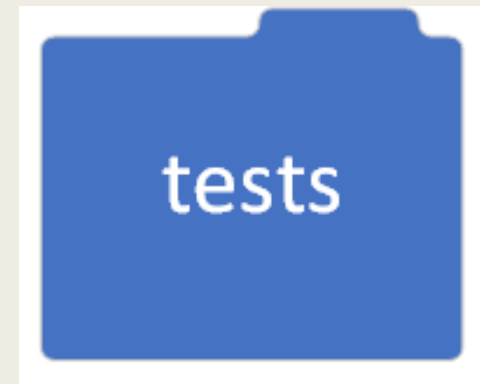
## Data: nycflights13

To explore the basic data manipulation verbs of dplyr, we’ll use `nycflights13::flights`. This dataset contains all 336776 flights that departed from New York City in 2013. The data comes from the US [Bureau of Transportation Statistics](#), and is documented in `?nycflights13`

```
library(nycflights13)
dim(flights)
#> [1] 336776    19
flights
#> # A tibble: 336,776 x 19
```

# tests/: automated tests

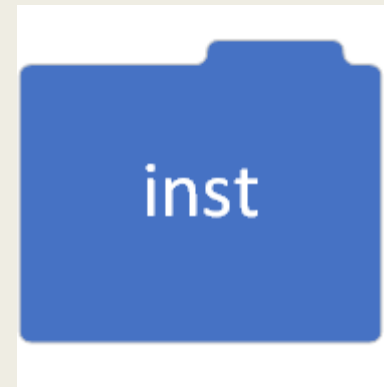
- The test folder contains tests that automatically run each time your code is built.
- Tests help make sure the package works the way you think does, and make sure you don't break the package while you're working on it.
- Write tests using the testthat package in R.
- It's a good idea to write a test for any new functions you write.

A screenshot of an R script editor showing the contents of a file named 'tests\_general.R'. The script uses the testthat package to write a test for the 'readNWISdata' function.

```
1 context("General functions")
2
3 test_that("General NWIS retrievals working", {
4   testthat::skip_on_cran()
5
6   multisite <- readNWISdata(sites=c("04025500", "04
7                               parameterCd="00060")
8   expect_is(multisite$dateTime, 'POSIXct')
9 }
```

# inst/: additional files

- The inst folder is used for any files you need your user to have access to.
  - Sample data where the file format is important (xml, csv)
  - Shiny web applications
- There are other places to store sample data if the format is not important such as example data or lookup tables



# A package workflow

- Write a new function, including documentation with roxygen2.
- Write tests for that function to verify to yourself and others that it works the way you want it to (and doesn't break in the future).
- Build and load the package.
- Commit or merge your code on GitHub (if applicable).
- Look out for bug reports, issues, and feature requests.
- As the package maintainer, clearly communicate expectations and lines of communication to users. (How will they report bugs? What level of support can they expect?)

# Tools and references for package development

- [R Packages, by Hadley Wickham](#)
- [USGS Package Development](#)
- Packages
  - [devtools](#): *everything you need for creating package directories and building and testing your package*
  - [roxygen2](#): *package documentation made easy. Allows the user to write R functions and documentation all in one file*
  - [testthat](#): *a framework for writing tests in your R package*