Package 'this.path'

January 20, 2022

Version 0.5.1

License MIT + file LICENSE

Title Get Executing Script's Path, from 'RStudio', 'Rgui', 'Rscript' (Shells Including Windows Command-Line / / Unix Terminal), and 'source'

Description Determine the full path of the executing script. Works when running a line or selection from a script in 'RStudio' and 'Rgui', when using 'source', 'sys.source', 'debugSource' in 'RStudio', and 'testthat::source_file', and when running R from a shell.

Author Andrew Simmons

Maintainer Andrew Simmons <akwsimmo@gmail.com>

Suggests utils, essentials, microbenchmark

Enhances testthat

URL https://github.com/ArcadeAntics/this.path

BugReports https://github.com/ArcadeAntics/this.path/issues

Encoding UTF-8

NeedsCompilation no

R topics documented:

`this.path-package` *Get Executing Script's Path, from 'RStudio', 'Rgui', 'Rscript' (Shells Including Windows Command-Line // Unix Terminal), and 'source'*

Description

Determine the full path of the executing script. Works when running a line or selection from a script in 'RStudio' and 'Rgui', when using 'source', 'sys.source', 'debugSource' in 'RStudio', and 'testthat::source_file', and when running R from a shell.

Details

The three most important functions from this package are `this.path`, `this.dir`, and `here`.

`this.path()` returns the `normalized` path of the executing script.

`this.dir()` is a shorter way of writing `dirname(this.path())`, returning the `normalized` path of the directory in which the executing script is located.

`here()` constructs file paths relative to the executing script's directory.

Note

This package started from a stack overflow posting, found at:

https://stackoverflow.com/questions/1815606/determine-path-of-the-executing-script

If you like this package, please consider upvoting my answer so that more people will see it! If you have an issue with this package, please use `utils::bug.report(package = "this.path")` to report your issue.

Author(s)

Andrew Simmons

Maintainer: Andrew Simmons <akwsimmo@gmail.com>

See Also

The main functions from **this.path**:

`this.path`, `this.dir`, `here`

`this.path` and `this.dir` variants:

`this.path2`, `this.dir2`, `this.dir3`

Check `this.path()` is functioning correctly:

`check.path`, `check.dir`

`source`, `sys.source`, `debugSource`, `testthat::source_file`

`R.from.shell`

check.path — *Check* `this.path()` *is Functioning Correctly*

Description

Add `check.path("path/to/file")` to the beginning of your script to initialize `this.path()`, and also to check that `this.path()` is returning the path you expect.

Usage

```
check.path(path)
check.dir (path)
```

Arguments

`path`	character string; the path you expect `this.path()` or `this.dir()` to return. `path` can be as deep as necessary (just the basename, the last directory name and the basename, the last two directory names and the basename, ...), but using an absolute path is not intended (recommended against). `this.path` makes R scripts portable, but using an absolute path in `check.path` or `check.dir` makes an R script non-portable, so it defeats the whole point.

Value

If the expected path / / directory matches `this.path` / / `this.dir`, then `TRUE` returned invisibly.

Otherwise, an error is raised.

Examples

```
# this.path::check.path("EOAdjusted/code/provrun.R")
```

here — *Construct Path to File, Beginning with 'this.dir()'*

Description

Construct the path to a file from components in a platform-independent way, starting with `this.dir()`.

Usage

```
here(..., .. = 0L)
ici(..., .. = 0L)
```

Arguments

`...`	further arguments passed to `file.path()`.
`..`	the number of directories to go back.

Details

The path to a file begins with a base. The base is `..` number of directories back from the executing script's directory (`this.dir()`). The argument is named `..` because `".."` refers to the parent directory in Windows, Unix, and URL paths alike.

Value

A character vector of the arguments concatenated term-by-term, beginning with the executing script's directory.

Examples

```
this.path:::write.code(file = FILE <- tempfile(), {


    this.path::here()
    this.path::here(.. = 1)
    this.path::here(.. = 2)


    # use 'here' to read input from a file located nearby
    this.path::here(.. = 1, "input", "file1.csv")


    # or maybe to run another script
    this.path::here("script2.R")


})


source(FILE, echo = TRUE, verbose = FALSE)
```

`R.from.shell` *Using R From a Shell*

Description

How to use R from a shell (including the Windows command-line / / Unix terminal).

Details

For the purpose of running R scripts, there are four ways to do it. Suppose our R script has filename '`script1.R`', we could write any of:
R -f script1.R
R --file=script1.R
R CMD BATCH script1.R
Rscript script1.R

The first two are different ways of writing equivalent statements. The third statement is the first statement plus options '`--restore`' '`--save`' (plus option '`--no-readline`' under Unix-alikes), and it also saves the `stdout` and `stderr` in a file of your choosing. The fourth statement is the second statement plus options '`--no-echo`' '`--no-restore`'. You can try

R --help
R CMD BATCH --help
Rscript --help

for a help message that describes what these options mean. In general, `Rscript` is the one you want to use. It should be noted that `Rscript` has some exclusive `environment variables` (not used by the other executables) that will make its behaviour different from `R`.

For the purpose of making packages, `R CMD` is what you'll need to use. Most commonly, you'll use:
R CMD build
R CMD INSTALL
R CMD check

`R CMD build` will turn an R package (specified by a directory) into tarball. This allows for easy sharing of R packages with other people. `R CMD INSTALL` will install an R package (specified by a directory or tarball), and is used by `utils::install.packages`. `R CMD check` will check an R package (specified by a tarball) for possible errors in code, documentation, tests, and much more.

If, when you execute one of the previous commands, you see the following error message: "'R' is not recognized as an internal or external command, operable program or batch file.", see section **Ease of Use on Windows**.

Ease of Use on Windows

Under Unix-alikes, it is easy to invoke an R session from a shell by typing the name of the R executable you wish to run. On Windows, you should see that typing the name of the R executable you wish to run does not run that application, but instead signals an error. Instead, you will have to type the full path of the directory where your R executables are located (see section **Where are my R executable files located?**), followed by the name of the R executable you wish to run.

This is not very convenient to type everytime something needs to be run from a shell, plus it has another issue of being computer dependent. The solution is to add the path of the directory where your R executables are located to the `Path` environment variable. The `Path` environment variable is a list of directories where executable programs are located. When you type the name of an executable program you wish to run, Windows looks for that program through each directory in the `Path` environment variable. When you add the full path of the directory where your R executables are located to your `Path` environment variable, you should be able to run any of those executable programs by their basenames ('R', 'Rcmd', 'Rscript', and 'Rterm') instead of their full paths.

To add a new path to your `Path` environment variable:

1. Open the **Control Panel**
2. Open category **User Accounts**
3. Open category **User Accounts** (again)
4. Open **Change my environment variables**
5. Click the variable `Path`
6. Click the button **Edit...**
7. Click the button **New**
8. Type (or paste) the full path of the directory where your R executables are located, and press **OK**

This will modify your environment variable `Path`, not the systems. If another user wishes to run R from a shell, they will have to add the directory to their `Path` environment variable as well.

If you wish to modify the system environment variable `Path` (you will need admin permissions):

1. Open the **Control Panel**

2. Open category **System and Security**
3. Open category **System**
4. Open **Advanced system settings**
5. Click the button **Environment Variables...**
6. Modify `Path` same as before, just select `Path` in **System variables** instead of **User variables**

To check that this worked correctly, open a shell and execute the following commands:
R --help
R --version

You should see that the first prints the usage message for the R executable while the second prints information about the version of R currently being run. If you have multiple versions of R installed, make sure this is the version of R you wish to run.

Where are my R executable files located?

In an R session, you can find the location of your R executable files with the following command:
cat(sQuote(normalizePath(R.home("bin"))), "\n")

On Windows, for me, this is:

'C:\Program Files\R\R-4.1.2\bin\x64'

On Linux, for me, this is:

'`/usr/lib/R/bin`'

`this.path` *Determine Executing Script's Filename*

Description

`this.path()` returns the `normalized` path of the executing script.

`this.dir()` is a shorter way of writing `dirname(this.path())`, returning the `normalized` path of the directory in which the executing script is located.

Usage

```
this.path(verbose = getOption("verbose"))
this.dir(...)
```

Arguments

`verbose`	`TRUE` or `FALSE`; should the method in which the path of the executing script was determined be printed?
`...`	arguments passed to `this.path`.

Details

There are three ways in which R code is typically run; in 'RStudio' or 'Rgui' by running the current line or selection with the **Run** button (or appropriate keyboard shortcut), through a source call (a call to function `source`, `sys.source`, `debugSource` in 'RStudio', or `testthat::source_file`), and from a shell (including the Windows command-line / / Unix terminal).

To retrieve the executing script's filename, first an attempt is made to find a source call. The calls are searched in reverse order so as to grab the most recent source call in the case of nested source calls. If a source call was found, the argument *file* (*fileName* in the case of `debugSource`, *path* in the case of `testthat::source_file`) is returned from the function's evaluation environment. If you have your own source-like function that you'd like to be recognized by `this.path`, please contact Andrew Simmons <akwsimmo@gmail.com> so it can be implemented.

If no source call is found up the calling stack, then an attempt is made to figure out how R is currently being used.

If R is being run from a shell, the shell arguments are searched for '`-f`' '`FILE`' or '`--file=FILE`' (the two methods of taking input from '`FILE`'). If exactly one of either type of argument is supplied, the text '`FILE`' is returned. It is an error to use `this.path` when none or multiple arguments of either type are supplied.

If R is being run from a shell under Unix-alikes with '`-g`' '`Tk`' or '`--gui=Tk`', `this.path()` will signal an error. This is because 'Tk' does not make use of its '`-f`' '`FILE`', '`--file=FILE`' argument.

If R is being run from 'RStudio', the active document's filename (the document in which the cursor is active) is returned (at the time of evaluation). If the active document is the R console, the source document's filename (the document open in the current tab) is returned (at the time of evaluation). Please note that the source document will *NEVER* be a document open in another window (with the **Show in new window** button). It is important to not leave the current tab (either by closing or switching tabs) while any calls to `this.path` have yet to be evaluated in the run selection. It is an error for no documents to be open or for a document to not exist (not saved anywhere).

If R is being run from 'Rgui', the source document's filename (the document most recently interacted with besides the R Console) is returned (at the time of evaluation). Please note that minimized documents will be *IGNORED*. It is important to not leave the current document (either by closing the document or interacting with another document) while any calls to `this.path` have yet to be evaluated in the run selection. It is an error for no documents to be open or for a document to not exist (not saved anywhere).

If R is being run from 'AQUA', the executing script's path cannot be determined. Unlike 'RStudio' and 'Rgui', there is currently no way to request the path of an open document. Until such a time that there is a method for requesting the path of an open document, consider using 'RStudio'.

If R is being run in another manner, it is an error to use `this.path`.

If your GUI of choice is not implemented with `this.path`, please contact Andrew Simmons <akwsimmo@gmail.com> so it can be implemented.

Value

character string; the executing script's filename.

Note

The first time `this.path` is called within a script, it will `normalize` the script's path, check that the script exists (throwing an error if it does not), and save it in the appropriate environment. When `this.path` is called subsequent times within the same script, it returns the saved path. This will be faster than the first time, will not check for file existence, and will be independent of the working directory.

As a side effect, this means that a script can delete itself using `file.remove` or `unlink` but still know its own path for the remainder of the script.

Within a script that contains calls to both `this.path` and `setwd`, `this.path` *MUST* be used *AT LEAST* once before the first call to `setwd`. This isn't always necessary; for instance if you ran a script using its absolute path as opposed to its relative path, changing the working directory has no effect. However, it is still advised against.

The following is *NOT* an example of bad practice:

```
setwd(this.path::this.dir())
```

`setwd` is most certainly written before `this.path()`, but `this.path()` will be evaluated first. It is not the written order that is bad practice, but the order of evaluation. Do not change the working directory before calling `this.path` at least once.

See Also

`here`

`this.path-package`

`source`, `sys.source`, `debugSource`, `testthat::source_file`

`R.from.shell`

Examples

```
this.path:::write.code(file = FILE <- tempfile(), {

    withAutoprint({


        cat(sQuote(this.path::this.path(verbose = TRUE)), "\n\n")


    }, verbose = FALSE)

})


source(FILE, verbose = FALSE)
sys.source(FILE, envir = environment())
if (.Platform$GUI == "RStudio")
    get("debugSource", "tools:rstudio", inherits = FALSE)(FILE)
if (requireNamespace("testthat"))
    testthat::source_file(FILE, chdir = FALSE, wrap = FALSE)


this.path:::.Rscript(c("--default-packages=NULL", "--vanilla", FILE))


# this.path also works when source-ing a URL
# (included tryCatch in case an internet connection is not available)
tryCatch({
   source("https://raw.githubusercontent.com/ArcadeAntics/this.path/main/tests/this.path_w_URLs.R")
}, condition = base::message)
```

`this.path-defunct` *Defunct Functions in Package* **this.path**

Description

The functions or variables listed here are no longer part of `this.path` as they are no longer needed or have been moved to a new package.

Usage

```
ArgumentParser(...)

## S4 method for signature 'this.path_ArgumentParser'
add.argument(...)

## S4 method for signature 'this.path_ArgumentParser'
add.help(...)

## S4 method for signature 'this.path_ArgumentParser'
add.skip(...)

## S4 method for signature 'this.path_ArgumentParser'
add.version(...)

## S4 method for signature 'this.path_ArgumentParser'
add.subparsers(...)

Args(...)

Commands(...)

dedent(...)

delayedAssign2(...)

file.open(...)

list.files2(...)
dir2(...)

Missing(...)

normalizeAgainst(...)

commandPrompt(...)

commandQuote(...)
shEncode(...)

path.contract(...)
```

```
pseudoglobalenv(...)

python(...)

R(...)
Rcmd(...)
Rscript(...)
Rterm(...)

writeArgs(...)
readArgs(...)

has.ext(...)
scan2(...)
format4scan(...)
setReadWriteArgsMethod(...)

withArgs(...)
```

Arguments

```
...
```

See Also

Defunct

`this.path2` *Determine Executing Script's Filename*

Description

Versions of `this.path` and `this.dir` which return `NULL` or the `working directory` instead of throwing an error when there is no executing script.

Usage

```
this.path2(...)
this.dir2(...)

this.dir3(...)
```

Arguments

`...`	arguments passed to `this.path`.

Details

There are a few reasons why there would be no executing script:

- R is being run from a shell without supplying argument '`-f`' '`FILE`' or '`--file=FILE`'

- R is being run from a shell under Unix-alikes with option '-g' 'Tk' or '--gui=Tk' (GUI 'Tk' does not make use of its '-f' 'FILE' or '--file=FILE' arguments)
- R is being run from 'RStudio' with no documents open
- R is being run from 'Rgui' with no documents open

The following cases will still throw an error:

- unable to `normalize` the executing script's filename
- using `sys.source` or `testthat::source_file` on a file named "clipboard", "clipboard-128", or "stdin"
- active / / source document in 'RStudio' / / 'Rgui' does not exist (not saved anywhere)
- in 'AQUA', running the current line or selection from an open R script (currently unimplemented)

Value

for `this.path2` or `this.dir2`, the executing script's filename or directory, or `NULL` if unavailable.

for `this.dir3`, the executing script's directory, or the working directory if unavailable.

Examples

```
this.path:::.Rscript(c("--default-packages=NULL", "--vanilla", "-e", "this.path::this.path()" ))
this.path:::.Rscript(c("--default-packages=NULL", "--vanilla", "-e", "this.path::this.path2()"))

this.path:::.Rscript(c("--default-packages=NULL", "--vanilla", "-e", "this.path::this.dir()" ))
this.path:::.Rscript(c("--default-packages=NULL", "--vanilla", "-e", "this.path::this.dir2()" ))
this.path:::.Rscript(c("--default-packages=NULL", "--vanilla", "-e", "this.path::this.dir3()" ))
```

Index