

## A Very Short Introduction to *R*-help By Alexis Diamond<sup>1</sup>

*R*-help pages may seem cryptic at first, but they can be an extremely useful tool for learning about commands, functions, and other *R*-objects that you don't understand.

### I. Accessing *R*-help

To learn about a command you don't understand, type a question mark followed by the name of the command. For example, to see the help page describing the *mean* command (which takes the average of a set of numbers), type:

```
> ?mean
```

If you don't know the name of the command you are looking for, you'll want to do a keyword search using *help.search*. For example, to find all commands that relate to taking the median, typing

```
> help.search("median")
```

produces the following list of functions and their corresponding packages<sup>2</sup>:

Help files with alias or concept or title matching 'median' using fuzzy matching:

<code>mad(stats)</code>	Median Absolute Deviation
<code>median(stats)</code>	Median Value
<code>medpolish(stats)</code>	Median Polish of a Matrix
<code>runmed(stats)</code>	Running Medians
<code>smooth(stats)</code>	Tukey's (Running Median) Smoothing
<code>smoothEnds(stats)</code>	End Points Smoothing

Type 'help(FOO, package = PKG)' to inspect entry 'FOO(PKG)TITLE'.

If it was “median value” that you were interested in, you would display the help page by typing:

```
> help("median", package = stats)
```

because *median* is the FOO (the *R*-object of interest), and the package is *stats*.

---

<sup>1</sup> This was written for Harvard's Gov Dept Math Camp '04, but anyone is free to use it for any purpose whatsoever without permission—as long as modifications from my original document are noted as such.

<sup>2</sup> Functions are bundled in packages.

## II. Reading *R*-help pages

The vast majority of *R*-help pages follow the same basic format:

*Title*

*Description*

*Usage*

*Arguments*

*Details*

*Value*

*References*

*See Also*

*Examples*

If you're looking at an *R*-help page, the first thing you probably want to do is confirm what it is that the *R*-object in question actually does. If the *Description* stumps you, don't sweat it: descriptions may be terse and difficult to understand. *Details*, when provided, are often more helpful for understanding the ins and outs of queried objects. Also, don't forget to play with the examples provided at the end of the page. These examples can almost always be copied and pasted directly into your command console for a hands-on learning experience.

The *Usage* section shows you how the *R*-object ought to be used. If the object is a function,<sup>3</sup> the *Arguments* section will include all the different inputs and the *Value* section will describe all the different outputs the function can produce.

For example, consider the “paste” function, which takes vectors as inputs and converts them into character strings that are put together in a way that may not be immediately intuitive.

To view the help page for this function, type:

```
> ?paste
```

and the following should appear:

---

<sup>3</sup> Functions are *R*-objects that take input and generate output.

## Concatenate Strings

## Description:

Concatenate vectors after converting to character.

## Usage:

```
paste(..., sep = " ", collapse = NULL)
```

## Arguments:

`...`: one or more **R** objects, to be coerced to character vectors.

`sep`: a character string to separate the terms.

`collapse`: an optional character string to separate the results.

## Details:

'paste' converts its arguments to character strings, and concatenates them (separating them by the string given by 'sep'). If the arguments are vectors, they are concatenated term-by-term to give a character vector result.

If a value is specified for 'collapse', the values in the result are then concatenated into a single string, with the elements being separated by the value of 'collapse'.

## Value:

A character vector of the concatenated values. This will be of length zero if all the objects are, unless 'collapse' is non-NULL, in which case it is a single empty string.

## References:

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

## See Also:

String manipulation with 'as.character', 'substr', 'nchar', 'strsplit'; further, 'cat' which concatenates and writes to a file, and 'sprintf' for C like string construction.

## Examples:

```
paste(1:12) # same as as.character(1:12)
paste("A", 1:6, sep = "")
paste("Today is", date())
```

Peruse the help page to get a feel for *paste*, and **R**-help more generally. If you are new to **R** and/or programming, the page may be unintelligible and the best way to understand *paste* may be to copy and paste examples (at the bottom of the page) into your **R** console.

A function's arguments are often the key to understanding what it does and how to use it. From the *Arguments* section of the help page, it should be clear that the *paste* function may involve up to three sets of arguments: the **R** objects that are strung together, *sep*, and *collapse*. Default settings for *sep* and *collapse* are given in the *Usage* section, and they only need to be specified if you want something other than the default.

If the purposes of the arguments are not clear from reading their descriptions in the *Arguments* section, try playing with the examples provided.<sup>4</sup> You might begin with the second example from the help page:

```
> paste("A", 1:6, sep = "")
```

Now type:

```
> paste("A", 1:6, sep = "!")
```

and

```
> paste("A", 1:6, -5:1, sep = "!")
```

To see how the *collapse* argument works, , type something like:

```
> paste("A", 1:6, -5:1, sep = "!", collapse = " ")
```

or

```
> paste("A", 1:6, -5:1, sep = "!", collapse = " and then ")
```

Using **R**-help is a skill, and like any other skill it requires practice. Don't be discouraged if the help page is difficult to understand, or if you're having trouble identifying the function you need. Hang in there, and before you know it you'll be using **R**-help quickly and effortlessly as your first point of reference.

---

<sup>4</sup> Some of the commands below involve `1:6` as an argument; if you don't already know what this means, just type it in at the command line prompt:

```
> 1:6
```

Then, if you're in the mood, experiment with other formulations, like

```
> 1:100/100
```