File Input and Output

Introduction to Computational BioStatistics with R

Alexey Fedoseev

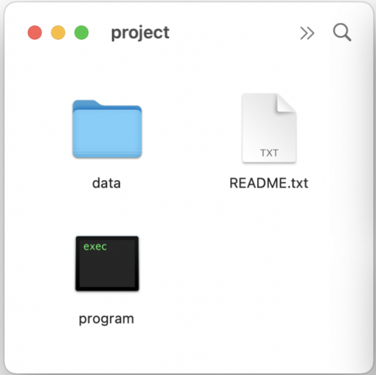
November 23, 2023

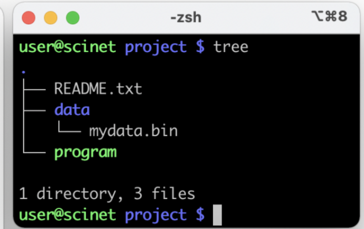
SciNet
ADVANCED RESEARCH COMPUTING at the UNIVERSITY OF TORONTO

Institute of Medical Science
UNIVERSITY OF TORONTO

Basic File Input and Output in R

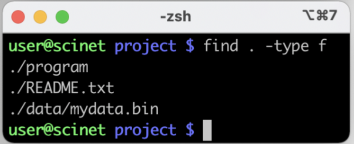
There are number of ways you can view files on your computer:





```
user@scinet project $ tree
.
├── README.txt
├── data
│   └── mydata.bin
└── program

1 directory, 3 files
user@scinet project $
```



```
user@scinet project $ find . -type f
./program
./README.txt
./data/mydata.bin
user@scinet project $
```

Directory management

You can use commands directly in R to view and manipulate your files and directories. For example:

- use `getwd()` to check where you are
- use `dir()` or `list.files()` to view files in the current directory
- use `setwd('scripts')` to change the directory
- use `dir.create('mydir')` to create a directory
- use `dir.exists('mydir')` to check whether the directory exists (returns `TRUE` or `FALSE`)

```
> dir()
[1] "data"  "program"  "README.txt"
> dir.exists("data")
[1] TRUE
> if (dir.exists("data")) {
+     setwd("data")
+ } else {
+     dir.create("data")
+ }
```

File management

File management commands:

- `file.exists("filename")`
- `normalizePath("filename")`
- `dirname("filename")`

```
> dir()
[1] "data"          "program"     "README.txt"
> file.exists("README.txt")
[1] TRUE
> normalizePath("README.txt")
[1] "/Users/alexey/project/README.txt"
```

Writing to a file

- `file(filename, 'w')` opens the file for writing. Use 'r' to read and 'a' to append
- `writeLines` writes lines to the file, depending on how the file was opened
- `write` opens, writes and closes the file in one shot
- `cat` writes to a file, but it doesn't automatically add newlines
- This technique is useful if you need to save some meta-data from an analysis
- If you are writing something in a loop, open a file once before the loops starts, use the loop to write data to the file, close the file after the loop

```
> myfile <- file("output.txt", "w")
> writeLines("Hello", myfile)
> writeLines("World", myfile)
> close(myfile)
>
> write(file = "output.txt", "Hello")
>
> write(file = "output.txt", "World",
+   append = T)
>
> cat("Hello\n", file = "output.txt",
+   append = T)
```

Writing a data frame to a file

You can save a whole data frame to a text file:

- Not generally recommended, especially if it's large
- It's better to save it as a binary file, using one of the techniques we'll cover later
- There is a whole family of functions: `write.csv` and `write.table` in particular
- Using `row.names = F` tells R to not include the row indices in the file, which you generally don't want
- If your data has actual row names though, you will probably want to keep them.

```
> mydata <- trees
> write.csv(mydata, file = "mydata.csv", row.names = F)
```

Using readLines

How to read from a text file:

- `readLines` by default reads the entire contents of the file
- Use the `n = 1` option to read just one line at a time.
- `readLines` will start at the beginning of the file once you reach the end
- You can also use `read.csv`, `read.delim`, `read.table`, etc.

```
> myfile <- file("output.txt", "r")
> file.content <- readLines(myfile)
> str(file.content)
 chr [1:3] "Hello" "World" "Hello2"
> fst.line <- readLines(myfile, n = 1)
> fst.line
[1] "Hello"
> close(myfile)
```

Using file wildcards

- The `glob2rx` function takes a Unix-style wildcard expression and converts it to a regular expression pattern.
- This is useful for finding files.

```
> dir(pattern = glob2rx("*.txt"))
[1] "output.txt" "README.txt"
```

Minimizing IOPs

It is important that you minimize file input and output operations as much as possible.

Common mistake is to open and close file in a loop when it is not necessary. Doing this will slow down your program a lot!

When reading data from a file, make sure to open file once, read/write the data in the file at once or in a loop if required, and finally close the file.

```
> mydata <- "Hello world"
>
> myfile <- file("hiworld.txt", "w")
> cat(mydata, file = myfile)
> close(myfile)
>
> cat("\n", file = "hiworld.txt", append = T)
```

Tips for IOPs

- Disk I/O is almost always the slowest part of a data pipeline
- If manipulating data from files is most of what you do, try to minimize IOPs (Input/Output Operations per second)

Tips for making things better: - Load everything into memory once

- Reuse data, don't keep re-reading it from your files
- If you must keep writing temporary things to disk, use ramdisk (memory as disk)
- Keep your results in memory, and then write your results in one shot when you are finished
- Use binary files

What’s in a file?

Files come in different formats. For our purposes there are two basic types:

Text:

- On its face this seems attractive: you can just read it
- But this is not as trivial as it may sound
- A bit pattern must be assigned to each letter or symbol (encoding)

Binary:

- This corresponds to saving data the way the machine keeps the data in memory: fast and efficient
- Good binary formats include information about the data within the file, e.g.: HDF5, NetCDF.

Text format

An introduction to text:

- ASCII Encoding: 7 bits = 1 character
- 128 possible, but only 95 printable characters
- Uses 8-bit bytes: storage efficiency 82% at best
- ASCII representation of floating point numbers:
  - Needs about 18 bytes vs 8 bytes in binary: *inefficient*
  - Representation must be computed: *slow*
  - *Non-exact* representation.

ASCII

Integers	Characters
32	(space)
33-47	!“#$%&’()*+,-./
48-57	0-9
58-64	:;<=>?
65-90	A-Z
91-96	[\]^_
97-122	a-z
123-126	{|}~

Text Encodings

There are a variety of text encoding available:

- ASCII: 7-bit encoding. For English.
- Latin-1: 8-bit encoding. For western European Languages mostly
- UTF-8: Variable-width encoding that can represent every character in the Unicode character set.
- Unicode: standard containing more than 110,000 characters.

R can deal with these encodings:

- Use the `Encoding` function to set the encoding of a string
- Use the `iconv` function to convert between encodings.
- Use the unicode escape character `\U` to indicate to R that the following is a Unicode character.

Text Encodings

```
> x <- "fa\xe7ile"
> Encoding(x)
[1] "unknown"
> Encoding(x) <- "latin1"
> x
[1] "façile"
> xx <- iconv(x, "latin1", "UTF-8")
> Encoding(xx)
[1] "UTF-8"
> x
[1] "façile"
> a <- "\U00B5"
> a
[1] "µ"
```

Binary format

Binary is a different way of storing information.

- The data are output to storage in the same format in which they are stored in memory
- Fast and space-efficient, especially numbers

Writing 128M doubles

	SciNet file system	ramdisk
ASCII	173 s	174 s
binary	6 s	1 s

- Not human readable

Why you should not use raw binary data

Data which is dumped to disk without any added formatting is called ‘raw’. Such a dump of the memory is very fast, but you lose the information describing the data. For example:

- Suppose you dump a 2D array of 100x100 floating point numbers
- This gives you a file of 40,000 bytes
- If you give this to someone else, how will he know what it is? It could be almost anything:
    - a 2D array of 100x100 numbers
    - a 1D array of 10,000 floating point numbers
    - a string of 40,000 characters
    - etc.

Obviously we need some metadata to go with the actual information we are trying to save.

Binary Formats

You could invent your own binary format, but it's better to take an existing standard: this saves you potential bugs, the burden of documentation and/or maintaining an I/O library.

- Rdata: An R-specific format. Cannot be read by other languages
- RDS: Another R-specific format. Stores a single R object only
- HDF5: Another standard, self-describing format. Almost a file system in a file. Several bio-informatics packages use this as a back-end

Bioinformatics, genetics, and other bio-type fields all have their own formats.

The Rdata type

The simplest way to save and retrieve data:

- You can save variables using the `save` function
- To load saved data, use `load`
- Note that your loaded data will overwrite any existing variables of the same name

```
> var1 <- 10; var2 <- "hello"
> save(var1, var2, file = "mydata.Rdata")
```

Now exit and re-launch the R prompt:

```
> load("mydata.Rdata")
> print(var1); print(var2)
[1] 10
[1] "hello"
```

RDS files

RDS is similar to the Rdata file format, with some exceptions:

- only a single object can be saved
- the object is serialized during the saving
- when loaded, the object is directly assigned to a variable, instead of relying on the variable name that it had previously
- `load()` can overwrite objects, silently; `readRDS()` cannot.

```
> a <- 1:10
> saveRDS(a, file = "mydata.RDS")
> b <- readRDS("mydata.RDS")
> b
[1] 1 2 3 4 5 6 7 8 9 10
```

On the use of meta-data

What is meta-data? Simply put: data-about-the-data

- The best binary formats have the meta-data baked right into the data file
- This way the meta-data and the data are never separated; the meta-data is always available.
- Always, always, include the meta-data with the data itself
- If you don't keep your meta-data in the same file as the data, at least keep it in the same directory
- Why? You need to know where did it come from. Under what conditions? Can you trust it?

On the use of meta-data

What do I include in my meta-data (data about the data)?

- Include your name, as the author of the data.
- Include the date and time the data was created or collected.
- Include the name of the code, and the version number of the code, which was used to create it.
- Include where it was created, what operating system.
- Include the values of key variables that were used to create the data, if your functions have optional values.
- Include anything and everything that might help you, in six months, to understand the what/where/why/how of the data.
- Include any other information that will allow you to TRUST the data.

If you're not sure, include it!

Final Tips

Some tips for optimizing your IOPS:

- Don't create millions of files: it's unworkable and slows down directories. If must have lots of directories, bundle them into tarballs.
- Stick to letters, numbers, underscores and periods in file names (no spaces!)
- Minimize IOPS: write/read big chunks at a time; try to reuse data or load more into memory.
- If your data is not text, do not save it as text.
- Always always save your meta-data with your data.