R - Packages

R packages are a collection of R functions, complied code and sample data. They are stored under a directory called **"library"** in the R environment. By default, R installs a set of packages during installation. More packages are added later, when they are needed for some specific purpose. When we start the R console, only the default packages are available by default. Other packages which are already installed have to be loaded explicitly to be used by the R program that is going to use them.

All the packages available in R language are listed at R Packages.

Below is a list of commands to be used to check, verify and use the R packages.

Check Available R Packages

Example: Get library locations containing R packages

```
> .libPaths()
[1] "C:/Users/hp/AppData/Local/R/win-library/4.4"
[2] "C:/Program Files/R/R-4.4.0/library"
```

Example: Get the list of all the packages installed:

```
> library()
```

When we execute the above code, it produces the following result. It may vary depending on the local settings of your pc. In our case, the results will be:

```
Packages in library ‘C:/Users/hp/AppData/Local/R/win-library/4.4’:

abind                   Combine Multidimensional Arrays
AnnotationDbi           Manipulation of SQLite-based annotations
                        in Bioconductor
askpass                 Password Entry Utilities for R, Git, and
                        SSH
base64enc               Tools for base64 encoding
BH                      Boost C++ Header Files
Biobase                 Biobase: Base functions for Bioconductor
BiocGenerics            S4 generic functions used in Bioconductor
BiocIO                  Standard Input and Output for
```

Get all packages currently loaded in the R environment

```
> search()
 [1] ".GlobalEnv"        "tools:rstudio"     "package:stats
"
 [4] "package:graphics"  "package:grDevices" "package:utils
"
 [7] "package:datasets"  "package:methods"   "Autoloads"
[10] "package:base"
```

When we execute the above code, it produces the above result. It may vary depending on the local settings of your pc.

Install a New Package

There are two ways to add new R packages. One is installing directly from the CRAN (Comprehensive R Archive Network) directory and another is downloading the package to your local system and installing it manually.

Install directly from CRAN

The following command gets the packages directly from CRAN webpage and installs the package in the R environment. You may be prompted to choose a nearest mirror. Choose the one appropriate to your location.

install.packages("Package Name")

```
> # Install the package named "XML".
> install.packages("XML")

WARNING: Rtools is required to build R packages but is not curren
tly installed. Please download and install the appropriate versio
n of Rtools before proceeding:

https://cran.rstudio.com/bin/windows/Rtools/
Installing package into ‘C:/Users/hp/AppData/Local/R/win-library/
4.4’
(as ‘lib’ is unspecified)

  There is a binary version available but the source version
  is later:
        binary    source needs_compilation
XML 3.99-0.16.1 3.99-0.17              TRUE

  Binaries will be installed
trying URL 'https://cran.rstudio.com/bin/windows/contrib/4.4/XML_
3.99-0.16.1.zip'
Content type 'application/zip' length 3103340 bytes (3.0 MB)
```

```
downloaded 3.0 MB

package ‘XML’ successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\Public\Documents\iSkysoft\CreatorTemp\RtmpWeHl94\
downloaded_packages
```

Install package manually

Go to the link R Packages to download the package needed. Save the package as a **.zip** file in a suitable location in the local system.

Now you can run the following command to install this package in the R environment.

install.packages(file_name_with_path, repos = NULL, type = "source")

Example:

```
> # Install the package named "XML"
>
> install.packages("E:/XML_3.98-1.3.zip", repos = NULL, typ
e = "source")
```

Load Package or Library

Before a package can be used in the code, it must be loaded to the current R environment. You also need to load a package that is already installed previously but not available in the current environment. A package is loaded using the following command:

```
> # Loading library called "xlsx"
>
> library("xlsx")
```

Viewing the contents of loaded library

In R, you can view the contents of a library (also known as a package) using the ls() function or by exploring the package's documentation. Here's how you can do it:

```
> ls("package:xlsx")
```

```
 [1] "addAutoFilter"           "addDataFrame"
 [3] "addHyperlink"            "addMergedRegion"
 [5] "addPicture"              "Alignment"
 [7] "autoSizeColumn"          "Border"
 [9] "BORDER_STYLES_"          "CB.setBorder"
[11] "CB.setColData"           "CB.setFill"
[13] "CB.setFont"              "CB.setMatrixData"
[15] "CB.setRowData"           "CELL_STYLES_"
[17] "CellBlock"               "CellProtection"
[19] "CellStyle"               "createCell"
[21] "createCellComment"       "createFreezePane"
[23] "createRange"             "createRow"
[25] "createSheet"             "createSplitPane"
[27] "createWorkbook"          "DataFormat"
[29] "Fill"                    "FILL_STYLES_"
[31] "Font"                    "forceFormulaRefresh"
[33] "forcePivotTableRefresh" "get_java_tmp_dir"
[35] "getCellComment"          "getCells"
[37] "getCellStyle"            "getCellValue"
[39] "getRanges"               "getRows"
[41] "getSheets"               "HALIGN_STYLES_"
[43] "INDEXED_COLORS_"         "is.Alignment"
[45] "is.Border"               "is.CellBlock"
[47] "is.CellProtection"       "is.CellStyle"
[49] "is.DataFormat"           "is.Fill"
[51] "is.Font"                 "loadWorkbook"
[53] "printSetup"              "read.xlsx"
[55] "read.xlsx2"              "readColumns"
[57] "readRange"               "readRows"
[59] "removeCellComment"       "removeMergedRegion"
[61] "removeRow"               "removeSheet"
[63] "saveWorkbook"            "set_java_tmp_dir"
[65] "setCellStyle"            "setCellValue"
[67] "setColumnWidth"          "setPrintArea"
[69] "setRowHeight"            "setZoom"
[71] "VALIGN_STYLES_"          "write.xlsx"
[73] "write.xlsx2"
```

You can also explore the documentation of a package to see its contents by Using **help()** or **?** to Explore Documentation in the help tab (down right window of RStudio).

```
> # Explore the documentation of package in help tab
>
> help(package = "xlsx")
>
> # Using ? to get help (this is used to get help of everyt
hing)
> ? "xlsx"
```

Detaching a Package

you can **detach** a package from the search path, which effectively removes its functions and datasets from your current R session. Use the **detach()** function to remove a package from the search path:

```
> # Viewing current loaded packages
>
> search()
 [1] ".GlobalEnv"        "package:xlsx"      "tools:rstudio
"
 [4] "package:stats"     "package:graphics"  "package:grDev
ices"
 [7] "package:utils"     "package:datasets"  "package:metho
ds"
[10] "Autoloads"         "package:base"
>
> # Detaching “xlsx” package
>
> detach("package:xlsx", unload = TRUE)
>
> # Viewing loaded packages after detaching “xlsx” package
>
> search()
 [1] ".GlobalEnv"        "tools:rstudio"     "package:stats
"
 [4] "package:graphics"  "package:grDevices" "package:utils
"
 [7] "package:datasets"  "package:methods"   "Autoloads"
[10] "package:base"
```

- "package:dplyr": Specifies the package to detach.
- unload = TRUE: Ensures the package is unloaded from memory (optional but recommended).

What we need to work with Bioinformatics in R?

- Install R and RStudio.
- Setup Bioconductor (**BiocManager** package).
- Install key Bioinformatics packages (e.g., Biostrings, ShortRead, *etc*..)

What are Bioconductor?

Bioconductor is an open-source R-based platform for bioinformatics and computational biology. It provides **specialized tools** for analyzing high-throughput genomic, proteomic, and other omics data. Key features include:

- **Packages** for DNA-seq, RNA-seq, ChIP-seq, single-cell analysis, and more.
- **Efficient data structures** (e.g., GRanges, SummarizedExperiment).
- **Reproducible workflows** with built-in documentation.
- **Integration** with CRAN and GitHub.

How to install BiocManager package?

```
> #Installing BiocManager "("The Bioconductor package"
> # if condition used to check if it was installed, if not it wil
l install it
>
> if (!require("BiocManager")) install.packages("BiocManager")
```

How to install specific Bioconductor packages?

Bioconductor provides **specialized bioinformatics packages** for genomics, proteomics, and other omics data analysis. We can install these packages using the following installation function:

```
> BiocManager::install("ShortRead")
```

What are FASTA and FASTAQ files?

FASTA and **FASTQ** are standard file formats for storing biological sequence data (DNA, RNA, or protein). Here's a concise comparison:

1. FASTA Format

- **Purpose**: Stores nucleotide or protein sequences (*without* quality scores).
- **Structure**:
  - **Header line**: Starts with > (e.g., >gene123).
  - **Sequence data**: Lines of bases (A,T,C,G for DNA; A,U,C,G for RNA)

Example:

```
>sequence1
ATGCGATCGATCGATCGATCG
>sequence2
CGATCGATCGATCGATCGATA
```

2. FASTQ Format

- **Purpose**: Stores sequences *with quality scores* (common in high-throughput sequencing, e.g., Illumina).
- **Structure**:
  - **Header line**: Starts with **@** (e.g., @read123).
  - **Sequence line**: Bases (e.g., ATGCG...).
  - **Quality header**: + (optional repeat of header).
  - **Quality scores**: ASCII characters encoding accuracy per base (e.g., ! = low quality, ~ = high quality).

Example:

```
@read1
ATGCGATCG
+
!!!!!~~~
```

Key Differences

Feature	**FASTA**	**FASTQ**
Quality	✗ No quality scores	✓ Includes quality scores
Header	>	@ and +
Use Case	Reference genomes, proteins	Raw sequencing reads

Example Use Cases

- **FASTA**: Reference genomes (e.g., human GRCh38), protein databases (UniProt).
- **FASTQ**: Output from DNA sequencers (e.g., Illumina, Nanopore).

Note: FASTQ files are larger due to quality data. Tools like gzip compress them efficiently.

How to work with FASTA and FASTAQ files in R?

To work with **FASTA** and **FASTQ** files in R, you'll need specialized packages, primarily from **Bioconductor**, as well as some CRAN packages. Below is a structured guide to the essential tools and their functionalities:

1. Core Bioconductor Packages

A. Biostrings

- **Purpose**: Handles biological sequences (DNA, RNA, proteins) and supports FASTA/FASTQ I/O.
- **Key Functions**:
  - readDNAStringSet(), readAAStringSet(): Load sequences into R.
  - writeXStringSet(): Export sequences to FASTA/FASTQ files.

  - Supports quality scores for FASTQ files (though ignored by default in readDNAStringSet).

- **Installation**:

```
> BiocManager::install("Biostrings")
'getOption("repos")' replaces Bioconductor standard re
positories, see
'help("repositories", package = "BiocManager")' for de
tails.
Replacement repositories:
    CRAN: https://cran.rstudio.com/
Bioconductor version 3.20 (BiocManager 1.30.25), R 4.4
.0 (2024-04-24
  ucrt)
Installing package(s) 'Biostrings'
trying URL 'https://bioconductor.org/packages/3.20/bio
c/bin/windows/contrib/4.4/Biostrings_2.74.1.zip'
Content type 'application/zip' length 13732150 bytes (
13.1 MB)
downloaded 13.1 MB

package ‘Biostrings’ successfully unpacked and MD5 sum
s checked
```

B. ShortRead

- **Purpose**: Specialized for high-throughput sequencing data (e.g., Illumina FASTQ).
- **Key Functions**:
  - readFastq(): Imports FASTQ files into a ShortReadQ object (stores sequences, IDs, and quality scores).
  - quality(): Extracts Phred quality scores.
- **Installation**:

```
> BiocManager::install("ShortRead")
```

2. Additional Useful Packages

A. insect (CRAN)

- **Purpose**: Lightweight FASTA/FASTQ parsing.
- **Key Functions**:
  - readFASTA(), readFASTQ(): Returns sequences as DNAbin objects or character strings.
- **Installation**:

```
> install.packages("insect")
```

B. qrqc (Bioconductor)

- **Purpose**: Quality control for FASTQ files.
- **Key Functions**:
  - readSeqFile(): Summarizes nucleotide distributions, qualities, and sequence lengths.
- **Installation**:

```
> BiocManager::install("qrqc")
```

3. Workflow Examples

Loading a FASTA File

```
> sequences <- readDNAStringSet("2_16Sd.fasta")  # Returns
a DNAStringSet object
> sequences # Showing the sequences

DNAStringSet object of length 2:
    width seq                                              names
[1]  1262 TAAAATTCGAGGTTCGGCCT...TTAATTTTCCGGGAATTGGC 2_16S
d
[2]  1170 CTAAAACTGAGAGGTTTCGG...TTGAACGGTGGGGAAACCTT 3_16S
d
```

Loading a FASTQ File

```
> library(ShortRead)
> fastq_data <- readFastq("4_16Sd.fastq")  # Returns ShortR
eadQ object
```

```
> qual_scores <- quality(fastq_data)          # Extract qu
ality scores
> qual_scores
class: FastqQuality
quality:
BStringSet object of length 1:
    width seq
[1]   980 3+*..>074B><FW21[U3'O[FP[E?T><:...575-5-=:0D/N=UN
I?N?KDD5:80---7
```

Converting FASTQ to FASTA

```
> library(ShortRead)
> fastq <- readFastq("4_16Sd.fastq")
> writeFasta(fastq, "output.fasta")  # Requires explicit ou
tput filenames :cite[5]
```

4. Performance Considerations

- For **large files**, Biostrings and ShortRead are memory-efficient.
- insect offers binary (DNAbin) formats for reduced memory usage.
- Avoid seqinr::read.fasta() for large files—it's slower than Biostrings

5. Summary of Recommended Packages

Task	**Package**	**Key Function**	**Notes**
FASTA I/O	Biostrings	readDNAStringSet()	Handles compressed files.
FASTQ I/O	ShortRead	readFastq()	Retains quality scores.
Lightweight parsing	insect	readFASTQ()	Good for small files.

Task	Package	Key Function	Notes
QC & Preprocessing	Rfastp/qrqc	rfastp()	Trimming, filtering.