

# Package ‘BiocPkgTools’

January 16, 2025

**Type** Package

**Title** Collection of simple tools for learning about Bioconductor Packages

**Version** 1.25.2

**Date** 2024-11-01

**Description** Bioconductor has a rich ecosystem of metadata around packages, usage, and build status. This package is a simple collection of functions to access that metadata from R. The goal is to expose metadata for data mining and value-added functionality such as package searching, text mining, and analytics on packages.

**Depends** htmlwidgets

**Imports** BiocFileCache, BiocManager, biocViews, tibble, magrittr, methods, rlang, stringr, stats, rvest, dplyr, xml2, readr, httr, htmltools, DT, tools, utils, igraph, jsonlite, gh, RBGL, graph, rorcid

**VignetteBuilder** knitr

**Suggests** BiocStyle, knitr, rmarkdown, testthat, tm, lubridate, networkD3, visNetwork, clipr, blastula, kableExtra, DiagrammeR, SummarizedExperiment

**License** MIT + file LICENSE

**BugReports** <https://github.com/seandavi/BiocPkgTools/issues/new>

**URL** <https://github.com/seandavi/BiocPkgTools>

**Encoding** UTF-8

**RoxygenNote** 7.3.2

**Roxygen** list(markdown = TRUE)

**SystemRequirements** mailsend-go

**biocViews** Software, Infrastructure

**git\_url** <https://git.bioconductor.org/packages/BiocPkgTools>

**git\_branch** devel

**git\_last\_commit** 19ff678

**git\_last\_commit\_date** 2024-11-01

**Repository** Bioconductor 3.21

**Date/Publication** 2025-01-16

**Author** Shian Su [aut, ctb],  
 Lori Shepherd [ctb],  
 Marcel Ramos [aut, ctb] (ORCID:  
<https://orcid.org/0000-0002-3242-0582>),  
 Felix G.M. Ernst [ctb],  
 Jennifer Wokaty [ctb],  
 Charlotte Soneson [ctb],  
 Martin Morgan [ctb],  
 Vince Carey [ctb],  
 Sean Davis [aut, cre]

**Maintainer** Sean Davis <seandavi@gmail.com>

## Contents

.getDepGain . . . . .	3
.get_cre_orcid . . . . .	4
.get_orcid_rec . . . . .	4
activitySince . . . . .	5
anacondaDownloadStats . . . . .	6
biocBuildEmail . . . . .	7
biocBuildReport . . . . .	9
biocBuildReportDB . . . . .	10
biocBuildStatusDB . . . . .	10
biocDownloadStats . . . . .	11
biocExplore . . . . .	12
biocMaintained . . . . .	13
biocPkgList . . . . .	14
biocPkgRanges . . . . .	15
BiocPkgTools . . . . .	16
BiocPkgTools-cache . . . . .	17
biocRevDepEmail . . . . .	18
buildPkgDependencyDataFrame . . . . .	19
buildPkgDependencyIgraph . . . . .	20
class-dependencies . . . . .	22
CRANstatus . . . . .	22
dataciteXMLGenerate . . . . .	23
firstInBioc . . . . .	24
generateBiocPkgDOI . . . . .	24
getBiocVignette . . . . .	25
getPackageInfo . . . . .	26
get_bioc_data . . . . .	27
get_cre_orcids . . . . .	27

<code>.getDepGain</code>	3
--------------------------	---

githubDetails . . . . .	28
githubURLParts . . . . .	29
inducedSubgraphByPkgs . . . . .	29
latestPkgStats . . . . .	30
orcid_table . . . . .	31
pkgBiocDeps . . . . .	32
pkgBiocRevDeps . . . . .	33
pkgCombDependencyGain . . . . .	34
pkgDepImports . . . . .	35
pkgDepMetrics . . . . .	36
pkgDownloadRank . . . . .	37
pkgDownloadStats . . . . .	38
problemPage . . . . .	38
repositoryStats . . . . .	39
subgraphByDegree . . . . .	41
templatePath . . . . .	42
<b>Index</b>	<b>43</b>

---

<code>.getDepGain</code>	<i>Calculate the 'dependency gain' from excluding one or more direct dependencies</i>
--------------------------	---

---

**Description**

Calculate the difference between the total number of dependencies of a package and the number of dependencies that would remain if one or more of the direct dependencies were removed.

**Usage**

`.getDepGain(g, pkg, depsToRemove)`

**Arguments**

- `g` Package dependency graph
- `pkg` Character string representing the package of interest
- `depsToRemove` Character vector representing the dependencies to remove

**Value**

The 'dependency gain' that would be achieved by excluding the indicated direct dependencies

**Author(s)**

Charlotte Soneson

---

<code>.get_cre_orcid</code>	<i>get the ORCID id from cre field of Authors@R in packageDescription result</i>
-----------------------------	--

---

**Description**

get the ORCID id from cre field of Authors@R in packageDescription result

**Usage**

```
.get_cre_orcid(pkgname)
```

**Arguments**

<code>pkgname</code>	<code>character(1)</code>
----------------------	---------------------------

---

<code>.get_orcid_rec</code>	<i>process employment data from ORCID</i>
-----------------------------	---

---

**Description**

process employment data from ORCID

**Usage**

```
.get_orcid_rec(orcid, rename = TRUE)
```

**Arguments**

<code>orcid</code>	<code>character(1)</code>
<code>rename</code>	logical(1) if TRUE use short names

---

activitySince

*What are the issues, pulls, commits created since a date?*


---

## Description

This function uses the `gh` package to get a list of either issues, pull requests, or GitHub commits since the specified date for a particular GitHub repository. The repository must have both the username / organization and the name, e.g., "Bioconductor/S4Vectors".

## Usage

```
activitySince(
  gh_repo,
  activity = c("issues", "pulls", "commits"),
  status = c("closed", "open", "all"),
  Date,
  issue_metadata = c("created_at", "number", "title"),
  token = NULL
)
```

## Arguments

<code>gh_repo</code>	character(1) The GitHub repository location including the username / organization and the repository name, e.g., "Bioconductor/S4Vectors"
<code>activity</code>	character(1) The type of repository activity to pull from the GitHub API. It can be one of "issues" (default), "pulls", or "commits".
<code>status</code>	character(1) One of 'closed', 'open', or 'all' corresponding to the issue state desired from the GitHub API (Default: "closed"). This argument is ignored for the "commits" activity report.
<code>Date</code>	character(1) The date cutoff from which to analyze closed issues in the YYYY-MM-DD or YYYY-MM-DDTHH:MM:SSZ format (ISO 8601).
<code>issue_metadata</code>	character() The metadata labels to extract from the <code>gh : gh</code> response. See <code>?gh : gh</code> for more details. Defaults to 'created_at', 'number', and 'title'. This argument is ignored for the "commits" activity report.
<code>token</code>	character(1) For big requests, e.g., commit history, you may be prompted to use a GitHub Personal Access Token. Enter the token as plain text.

## Details

The tibble returned by the commits activity report contains five columns:

- 'committer\_date'
- 'commit' - hash
- 'parents' - hash of parent for merge commits
- 'author'

- 'message'

For information on other columns, refer to the GitHub API under repository issues or pulls (e.g., `/repos/:repo/issues`).

### Value

A tibble with three columns corresponding to issue metadata (i.e., "created\_at", "number", "title")

### Examples

```
if (interactive()) {

  activitySince("Bioconductor/S4Vectors", "issues", "closed", "2021-05-01")
  activitySince("Bioconductor/S4Vectors", "issues", "open", "2022-05-01")
  activitySince("Bioconductor/S4Vectors", "commits", Date = "2022-05-01")

}
```

---

`anacondaDownloadStats` *Get download statistics for Bioconductor packages distributed via Anaconda.*

---

### Description

Get download statistics for Bioconductor packages distributed via Anaconda.

### Usage

```
anacondaDownloadStats()
```

### Details

Anaconda provide daily download counts for all software packages they distribute. These are summarised into monthly tables of counts and made available from <https://github.com/grimbough/anaconda-download-stats> This function provides a mechanism to download these monthly counts for Bioconductor packages distributed through Anaconda.

### Value

A `data.frame` of download statistics for all Bioconductor packages distributed by Anaconda, in tidy format. Note: Anaconda do not provide counts for unique IP addresses. This column is listed as NA for all packages to provide continuity with data from Bioconductor.org obtained by [biocDownloadStats](#). The counts are updated monthly, so do not expect to see counts for the current month.

### Author(s)

Mike L. Smith

## Examples

```
anacondaDownloadStats()
```

---

biocBuildEmail	Create and copy e-mail package notification template to clipboard
----------------	---

---

## Description

The `biocBuildEmail` function provides a template for notifying maintainers of errors in the Bioconductor Build System (BBS). This convenience function returns the body of the email from a template within the package and provides a copy in the clipboard.

## Usage

```
biocBuildEmail(
  pkg,
  version = c("release", "devel"),
  PS = character(1L),
  dry.run = TRUE,
  to = NULL,
  cc = NULL,
  bcc = NULL,
  emailTemplate = templatePath(),
  core.name = NULL,
  core.email = NULL,
  core.id = NULL,
  textOnly = FALSE,
  resend = FALSE,
  verbose = FALSE,
  credFile = "~/blastula_creds"
)

sentHistory()
```

## Arguments

<code>pkg</code>	<code>character(1)</code> The name of the package in trouble
<code>version</code>	<code>character()</code> A vector indicating which version of Bioconductor the package is failing in (either 'release' or 'devel'; defaults to both)
<code>PS</code>	<code>character(1)</code> Postscript, an additional note to the recipient of the email (i.e., the package maintainer)
<code>dry.run</code>	<code>logical(1)</code> Display the email without sending to the recipient. It only works for HTML email reports and ignored when <code>textOnly=TRUE</code>
<code>to</code>	<code>character()</code> A vector of email addresses serving as primary recipients for the message. For secondary recipients, use the <code>cc</code> and <code>bcc</code> arguments.

<code>cc, bcc</code>	<code>character()</code> A vector of email addresses for sending the message as a carbon copy or blind carbon copy.
<code>emailTemplate</code>	<code>character(1)</code> The path to the email template Rmd file as obtained by <code>templatePath()</code> . A custom template can be provided as file path.
<code>core.name</code>	<code>character(1)</code> The full name of the core team member
<code>core.email</code>	<code>character(1)</code> The Roswell Park email of the core team member
<code>core.id</code>	<code>character(1)</code> The internal identifier for the Roswell employee. This ID usually matches <code>^[A-Z]{2}[0-9]{5}</code> for more recent identifiers.
<code>textOnly</code>	<code>logical(1)</code> Whether to return the text of the email only. This avoids the use of the 'blastula' package and adds the text to the system clipboard if the clipr package is installed (default: FALSE)
<code>resend</code>	<code>logical(1)</code> Whether to force a resend of the email
<code>verbose</code>	<code>logical(1)</code> Whether to output full email information from 'smtp_send' (when <code>dry.run</code> is FALSE and 'blastula' is installed)
<code>credFile</code>	<code>character(1)</code> An optional file generated by the <code>blastula::create_smtp_creds_file</code> function containing email authentication information (default: <code>"~/blastula_creds"</code> ). See ? <code>biocBuildEmail</code> details.

## Details

The `credFile` argument is a convenience for avoiding password entry at every instance an email is sent. If the default file `~/blastula_creds` does not exist, the user will be prompted for authorization information. Currently it is configured to emails for the core-team:

```
blastula::create_smtp_creds_file(
  file = "~/blastula_creds",
  user = "user.email@domain.org",
  host = "smtp.office365.com",
  port = 587,
  use_ssl = TRUE
)
```

## Value

A character string of the email

## sentHistory

Check the history of emails sent

---

biocBuildReport

*Tidy Bioconductor build report results*


---

## Description

The online Bioconductor build reports are great for humans to look at, but they are not easily computable. This function scrapes HTML and text files available from the build report online pages to generate a tidy data frame version of the build report.

## Usage

```
biocBuildReport(
  version = BiocManager::version(),
  pkgType = c("software", "data-experiment", "data-annotation", "workflows"),
  stage.timings = FALSE
)
```

## Arguments

version	character(1) the character version number as used to access the online build report. For example, "3.14". The default is the "current version" as given by <code>BiocManager::version()</code> . Note that this is a character vector of length one and not a number.
pkgType	character(1) The type of packages for which to get build status information for. Valid values are: <ul style="list-style-type: none"> <li>• software: Software packages</li> <li>• data-experiment: Experiment data packages</li> <li>• data-annotation: Annotation data packages</li> <li>• workflows: Workflow packages</li> </ul>
stage.timings	logical(1) Whether to include the start, end, and elapsed time for each build, check, install stage from each building in the result (default: FALSE)

## Value

A `tbl_df` object with columns `pkg`, `version`, `author`, `commit`, `date`, `node`, `stage`, and `result`.

## Examples

```
# Set the stage--what version of Bioc am I using?
BiocManager::version()

latest_build <- biocBuildReport()
head(latest_build)
```

---

biocBuildReportDB	<i>Parse the Build Report tarball for a Bioconductor release</i>
-------------------	--

---

## Description

This function parses the Build Report tarball for a Bioconductor release. By default it will pull all the `report.tgz` files for each Bioconductor package type. The Bioconductor Build System (BBS) Build Report tarball contains build status information for all packages in a Bioconductor release. This function is mainly used by `biocBuildReport()`.

## Usage

```
biocBuildReportDB(
  version = BiocManager::version(),
  pkgType = c("software", "data-experiment", "data-annotation", "workflows"),
  stage.timings = FALSE
)
```

## Arguments

version	character(1) The numeric version of Bioconductor to use, e.g., "3.19". Keywords "release" and "devel" are also accepted.
pkgType	character(1) The type of packages for which to get build status information for. Valid values are: <ul style="list-style-type: none"> <li>• software: Software packages</li> <li>• data-experiment: Experiment data packages</li> <li>• data-annotation: Annotation data packages</li> <li>• workflows: Workflow packages</li> </ul>
stage.timings	logical(1) Whether to include the start, end, and elapsed time for each build, check, install stage from each building in the result (default: FALSE)

---

biocBuildStatusDB	<i>Download and parse the build status information for Bioconductor packages</i>
-------------------	--

---

## Description

This function downloads and parses the build status information for Bioconductor packages. The build status information is available for the current release and the previous release. Other versions may be available.

**Usage**

```
biocBuildStatusDB(
  version = BiocManager::version(),
  pkgType = c("software", "data-experiment", "data-annotation", "workflows")
)
```

**Arguments**

version	character(1) The numeric version of Bioconductor to use, e.g., "3.19". Keywords "release" and "devel" are also accepted.
pkgType	character(1) The type of packages for which to get build status information for. Valid values are: <ul style="list-style-type: none"> <li>• software: Software packages</li> <li>• data-experiment: Experiment data packages</li> <li>• data-annotation: Annotation data packages</li> <li>• workflows: Workflow packages</li> </ul>

**Value**

A data.frame with the following columns:

- pkg: The name of the package
- node: The builder on which the package was built
- stage: The stage of the build, e.g., 'install', 'buildsrc', 'checksrc', etc.
- result: The status of the build, e.g., 'OK', 'ERROR', 'WARNINGS', etc.

---

biocDownloadStats	<i>Get Bioconductor download statistics</i>
-------------------	---

---

**Description**

Get Bioconductor download statistics

**Usage**

```
biocDownloadStats(
  pkgType = c("software", "data-experiment", "workflows", "data-annotation")
)
```

**Arguments**

pkgType	character() All, some, or one of 'software', 'data-experiment', 'workflows', or 'data-annotation' (defaults to all types)
---------	---

**Details**

Note that Bioconductor package download stats are not version-specific.

**Value**

A tibble of download statistics for all Bioconductor packages

**Examples**

```
biocDownloadStats()
```

---

biocExplore

*Explore Bioconductor packages interactively*

---

**Description**

Explore Bioconductor packages through an interactive bubble plot. Click on bubbles to bring up additional information about the package. Size and proximity to center of a bubble is based on the downloads the package has in the past month.

**Usage**

```
biocExplore(top = 500L, ...)
```

**Arguments**

top	maximum number of packages displayed in any biocView
...	parameters passed to <code>htmlwidgets::createWidget()</code>

**Value**

A bubble plot of Bioconductor packages

---

biocMaintained

*Bioconductor Maintained Packages*


---

## Description

List all the packages associated with a maintainer. By default, it will return all packages associated with the `maintainer@bioconductor.org` email. `hasBiocMaint` returns a logical vector corresponding to the input character vector of packages indicating whether any package is maintained by the Bioconductor core team.

## Usage

```
biocMaintained(
  main = "maintainer@bioconductor\\.org",
  version = BiocManager::version(),
  pkgType = c("software", "data-experiment", "workflows", "data-annotation")
)

hasBiocMaint(
  pkg,
  version = BiocManager::version(),
  main = "maintainer@bioconductor\\.org",
  repo = c("BioCsoft", "BioCexp", "BioCworkflows", "BioCann")
)
```

## Arguments

<code>main</code>	character(1) The regex for searching through the Maintainer column as obtained from <code>biocPkgList()</code> .
<code>version</code>	character(1) the character version number as used to access the online build report. For example, "3.14". The default is the "current version" as given by <code>BiocManager::version()</code> . Note that this is a character vector of length one and not a number.
<code>pkgType</code>	character(1) The type of packages for which to get build status information for. Valid values are: <ul style="list-style-type: none"> <li>• <code>software</code>: Software packages</li> <li>• <code>data-experiment</code>: Experiment data packages</li> <li>• <code>data-annotation</code>: Annotation data packages</li> <li>• <code>workflows</code>: Workflow packages</li> </ul>
<code>pkg</code>	character(1) A vector of package names (case sensitive).
<code>repo</code>	character() A vector of Bioconductor repositories to search through. By default, it will search through all Bioconductor repositories.

**Value**

For biocMaintained: a tibble of packages associated with the maintainer.

For hasBiocMaint: a logical vector indicating whether the package is maintained by Bioconductor.

**Examples**

```
biocMaintained()

## maintained by Hervé and not maintainer at bioconductor dot org
hasBiocMaint("BiocGenerics")
```

---

biocPkgList

---

*Get full Bioconductor software package listing, with details*


---

**Description**

The BiocViews-generated VIEWS file is available for Bioconductor release and devel repositories. It contains quite a bit more information from the package DESCRIPTION files than the PACKAGES file. In particular, it contains biocViews annotations and URLs for vignettes and developer URLs.

**Usage**

```
biocPkgList(
  version = BiocManager::version(),
  repo = c("BioCsoft", "BioCexp", "BioCworkflows", "BioCann", "CRAN"),
  addBiocViewParents = TRUE
)
```

**Arguments**

version	The requested Bioconductor version. Will default to use the BiocManager defaults (i.e., version()).
repo	character(1) The requested Bioconductor repository. The default is to pull from the "BioCsoft" repository. Possible repositories include "BioCsoft", "BioCexp", "BioCworkflows", "BioCann", and "CRAN". Note that not all repos are available for all versions, particularly older versions.
addBiocViewParents	logical(1) whether to add all biocViews parents to biocViews annotations.

**Details**

Since packages are annotated with the most specific views, the default functionality here is to add parent terms for all views for each package. For example, in the bioCsoft repository, all packages will have at least "Software" added to their biocViews. If one wants to stick to only the most specific terms, set addBiocViewParents to FALSE.

**Value**

An object of class `tbl_df`.

**Examples**

```
bpkgl <- biocPkgList(repo = "BioCsoft")
bpkgl
unlist(bpkgl[1,'Depends'], use.names = FALSE)

# Get a list of all packages that
# import "GEOquery"
library(dplyr)
bpkgl |>
  filter(Package == 'GEOquery') |>
  pull('importsMe') |>
  unlist()
```

---

<code>biocPkgRanges</code>	<i>Grab build report results from BUILD_STATUS_DB for a particular package range</i>
----------------------------	--

---

**Description**

Grab build report results from BUILD\_STATUS\_DB for a particular package range

**Usage**

```
biocPkgRanges(
  start,
  end,
  condition = c("ERROR", "WARNINGS"),
  phase = "buildsrc",
  version = c("devel", "release")
)
```

**Arguments**

<code>start</code>	character(1) alphabetically first package name in range
<code>end</code>	character(1) alphabetically last package name in range
<code>condition</code>	character(1) condition string, typically 'ERROR' or 'WARNING'
<code>phase</code>	character(1) string for phase of event: 'install', 'checksrc', or 'buildsrc' (default)
<code>version</code>	character(1) string indication Bioconductor version, either 'devel' (default) or 'release'

**Author(s)**

Vincent J. Carey

## Examples

```
## Not run:
biocPkgRanges(
  start = "a4", end = "CMA",
  condition = "ERROR", version = "devel"
)

## End(Not run)
```

---

BiocPkgTools

*BiocPkgTools: Examine and analyze Bioconductor package metadata*

---

## Description

Bioconductor has a rich ecosystem of metadata around packages, usage, and build status. This package is a simple collection of functions to access that metadata from R. The goal is to expose metadata for data mining and value-added functionality such as package searching, text mining, and analytics on packages.

## For developers

The [biocBuildReport](#) function returns a computable form of the Bioconductor Build Report.

## For users

The [biocDownloadStats](#) function gets Bioconductor download stats, allowing users to quickly find commonly used packages. The [biocPkgList](#) is useful for getting a complete listing of all Bioconductor packages.

## Infrastructure

Bioconductor packages all have Digital Object Identifiers (DOIs). This package contains basic infrastructure for creating, updating, and de-referencing DOIs.

## Author(s)

**Maintainer:** Sean Davis <seandavi@gmail.com>

Authors:

- Shian Su <su.s@wehi.edu.au> [contributor]
- Marcel Ramos <marcel.amos@roswellpark.org> ([ORCID](#)) [contributor]

Other contributors:

- Lori Shepherd <Lori.Shepherd@roswellpark.org> [contributor]
- Felix G.M. Ernst <felix.gm.ernst@outlook.com> [contributor]
- Jennifer Wokaty <jennifer.wokaty@gmail.com> [contributor]

- Charlotte Soneson <charlottesoneson@gmail.com> [contributor]
- Martin Morgan <martin.morgan@roswellpark.org> [contributor]
- Vince Carey <stvjc@channing.harvard.edu> [contributor]

### See Also

Useful links:

- <https://github.com/seandavi/BiocPkgTools>
- Report bugs at <https://github.com/seandavi/BiocPkgTools/issues/new>

---

BiocPkgTools-cache	<i>Manage cache for BiocPkgTools</i>
--------------------	--------------------------------------

---

### Description

Managing user data is important to allow use of email functions such as `biocBuildEmail` and made easy with `BiocFileCache`.

### Usage

```
setCache(
  directory = tools::R_user_dir("BiocPkgTools", "cache"),
  verbose = TRUE,
  ask = interactive()
)

pkgToolsCache(...)
```

### Arguments

<code>directory</code>	The file location where the cache is located. Once set future downloads will go to this folder.
<code>verbose</code>	Whether to print descriptive messages
<code>ask</code>	logical (default TRUE when interactive session) Confirm the file location of the cache directory
<code>...</code>	For <code>pkgToolsCache</code> , arguments are passed to <code>setCache</code>

### pkgToolsCache

Get the directory location of the cache. It will prompt the user to create a cache if not already created. A specific directory can be used via `setCache`.

### setCache

Specify the directory location of the data cache. By default, it will go to the user's home/.cache/R and "appname" directory as specified by `tools::R_user_dir` (with `package="BiocPkgTools"` and `which="cache"`).

---

biocRevDepEmail	<i>Notify downstream maintainers of changes in upstream packages</i>
-----------------	--

---

## Description

The `biocRevDepEmail` function collects all the emails of the reverse dependencies and sends a notification that upstream package(s) have been deprecated or removed. It uses a template found in `inst/resources` with the `templatePath()` function.

## Usage

```
biocRevDepEmail(
  packages,
  which = c("strong", "most", "all"),
  PS = character(1L),
  version = BiocManager::version(),
  dry.run = TRUE,
  cc = NULL,
  emailTemplate = templatePath("revdepnote"),
  core.name = NULL,
  core.email = NULL,
  core.id = NULL,
  textOnly = FALSE,
  verbose = FALSE,
  credFile = "~/blastula_creds",
  ...,
  pkg
)
```

## Arguments

<code>packages</code>	<code>character()</code> A vector of CRAN and/or Bioconductor packages for whose reverse dependencies are to be checked and notified.
<code>which</code>	a character vector listing the types of dependencies, a subset of <code>c("Depends", "Imports", "LinkingTo", "Suggests", "Enhances")</code> . Character string "all" is shorthand for that vector, character string "most" for the same vector without "Enhances", character string "strong" (default) for the first three elements of that vector.
<code>PS</code>	<code>character(1)</code> Postscript, an additional note to the recipient of the email (i.e., the package maintainer)
<code>version</code>	<code>character()</code> A vector indicating which version of Bioconductor the package is failing in (either 'release' or 'devel'; defaults to both)
<code>dry.run</code>	<code>logical(1)</code> Display the email without sending to the recipient. It only works for HTML email reports and ignored when <code>textOnly=TRUE</code>
<code>cc</code>	<code>character()</code> A vector of email addresses for sending the message as a carbon copy.

emailTemplate	character(1) The path to the email template Rmd file as obtained by <code>templatePath()</code> . A custom template can be provided as file path.
core.name	character(1) The full name of the core team member
core.email	character(1) The Roswell Park email of the core team member
core.id	character(1) The internal identifier for the Roswell employee. This ID usually matches <code>^[A-Z]{2}[0-9]{5}</code> for more recent identifiers.
textOnly	logical(1) Whether to return the text of the email only. This avoids the use of the 'blastula' package and adds the text to the system clipboard if the clipr package is installed (default: FALSE)
verbose	logical(1) Whether to output full email information from 'smtp_send' (when <code>dry.run</code> is FALSE and 'blastula' is installed)
credFile	character(1) An optional file generated by the <code>blastula::create_smtp_creds_file</code> function containing email authentication information (default: <code>"~/blastula_creds"</code> ). See <code>?biocBuildEmail</code> details.
pkg	character(1) DEPRECATED. The name of a single package whose reverse dependencies are to be checked and notified.
...	Additional inputs to internal functions (not used).

### Examples

```
biocRevDepEmail(
  "FindMyFriends", version = "3.13", dry.run = TRUE, textOnly = TRUE
)
```

---

```
buildPkgDependencyDataFrame
```

*Work with Bioconductor package dependencies*

---

### Description

Bioconductor is built using an extensive set of core capabilities and data structures. This leads to package developers depending on other packages for interoperability and functionality. This function extracts package dependency information from [biocPkgList](#) and returns a `tidy.data.frame` that can be used for analysis and to build graph structures of package dependencies.

### Usage

```
buildPkgDependencyDataFrame(dependencies = c("strong", "most", "all"), ...)
```

### Arguments

dependencies	character() a vector listing the types of dependencies, a subset of <code>c("Depends", "Imports", "LinkingTo", "Suggests", "Enhances")</code> . Character string "all" is shorthand for that vector, character string "most" for the same vector without "Enhances", character string "strong" (default) for the first three elements of that vector.
...	parameters passed along to <a href="#">biocPkgList</a>

**Value**

A data.frame (also a tbl\_df) of S3 class "biocDepDF" including columns "Package", "dependency", and "edgetype".

**Note**

This function requires network access.

**See Also**

See [buildPkgDependencyIgraph](#), [biocPkgList](#).

**Examples**

```
# performs a network call, so must be online.
library(BiocPkgTools)
depdf <- buildPkgDependencyDataFrame()
head(depdf)
library(dplyr)
# filter to include only "Imports" type
# dependencies
imports_only <- depdf |> filter(edgetype=='Imports')

# top ten most imported packages
imports_only |> select(dependency) |>
  group_by(dependency) |> tally() |>
  arrange(desc(n))

# The Bioconductor packages with the
# largest number of imports
largest_importers <- imports_only |>
  select(Package) |>
  group_by(Package) |> tally() |>
  arrange(desc(n))

# not sure what these packages do. Join
# to their descriptions
biocPkgList() |> select(Package, Description) |>
  left_join(largest_importers) |> arrange(desc(n)) |>
  head()
```

**Description**

Package dependencies represent a directed graph (though Bioconductor dependencies are not an acyclic graph). This function simply returns an igraph graph from the package dependency data frame from a call to [buildPkgDependencyDataFrame](#) or any tidy data frame with rows of (Package, dependency) pairs. Additional columns are added as igraph edge attributes (see [graph\\_from\\_data\\_frame](#)).

**Usage**

```
buildPkgDependencyIgraph(pkgDepDF)
```

**Arguments**

pkgDepDF            a tidy data frame. See description for details.

**Value**

An igraph directed graph. See the igraph package for details of what can be done.

**See Also**

See [buildPkgDependencyDataFrame](#), [graph\\_from\\_data\\_frame](#), [inducedSubgraphByPkgs](#), [subgraphByDegree](#), [igraph-es-indexing](#), [igraph-vs-indexing](#)

**Examples**

```
library(igraph)

pkg_dep_df = buildPkgDependencyDataFrame()

# at this point, filter or join to manipulate
# dependency data frame as you see fit.

g = buildPkgDependencyIgraph(pkg_dep_df)
g

# Look at nodes and edges
head(V(g)) # vertices
head(E(g)) # edges

# subset graph by attributes

head(sort(degree(g, mode='in'), decreasing=TRUE))
head(sort(degree(g, mode='out'), decreasing=TRUE))
```

---

class-dependencies	<i>Retrieve Class relationships</i>
--------------------	-------------------------------------

---

### Description

As the title says it should do something with class relationships

### Usage

```
buildClassDepGraph(class, includeUnions = FALSE)
buildClassDepData(class, includeUnions = FALSE)
buildClassDepFromPackage(pkg, includeUnions = FALSE)
plotClassDep(class, includeUnions = FALSE)
plotClassDepData(data)
plotClassDepGraph(g)
```

### Arguments

class	a single character value defining a ‘S4’ class name
includeUnions	TRUE or FALSE: Should union definitions included in the result? (default: FALSE)
pkg	a single character value defining a package name
data	a data.frame with compatible columns. See output of buildClassDepData
g	an igraph object with compatible edge attributes. See output of buildClassDepGraph

### Examples

```
library("SummarizedExperiment")
depData <- buildClassDepData("RangedSummarizedExperiment")
depData
g <- buildClassDepGraph("RangedSummarizedExperiment")
plotClassDepGraph(g)
```

---

CRANstatus	<i>Check the CRAN build report page and email a notification</i>
------------	--

---

### Description

The CRANstatus function allows users to check the status of a package and send an email report of any failures.

**Usage**

```
CRANstatus(
  pkg,
  core.name = NULL,
  core.email = NULL,
  core.id = NULL,
  to.mail = "maintainer@bioconductor.org",
  dry.run = TRUE,
  emailTemplate = templatePath("cranreport")
)
```

**Arguments**

pkg	character(1) The name of the package in trouble
core.name	character(1) The full name of the core team member
core.email	character(1) The Roswell Park email of the core team member
core.id	character(1) The internal identifier for the Roswell employee. This ID usually matches <code>^[A-Z]{2}[0-9]{5}</code> for more recent identifiers.
to.mail	The email of the CRAN report recipient
dry.run	logical(1) Display the email without sending to the recipient. It only works for HTML email reports and ignored when <code>textOnly=TRUE</code>
emailTemplate	character(1) The path to the email template Rmd file as obtained by <code>templatePath()</code> . A custom template can be provided as file path.

---

dataciteXMLGenerate     *The Bioconductor datacite.org XML generator*

---

**Description**

This function is used internally to generate XML elements from the datacite.org website for incoming Bioconductor packages.

**Usage**

```
dataciteXMLGenerate(pkg)
```

**Arguments**

pkg	The name of a Bioconductor package
-----	------------------------------------

**Value**

An `xml_document` object from the `xml2` package.

**See Also**

```
?xml2::`xml_document-class`
```

---

firstInBioc	<i>When did a package enter Bioconductor?</i>
-------------	---

---

### Description

This function uses the biocDownloadStats data to *approximate* when a package entered Bioconductor. Note that the download stats go back only to 2009.

### Usage

```
firstInBioc(download_stats)
```

### Arguments

download\_stats a data.frame from [biocDownloadStats](#)

### Examples

```
dls <- biocDownloadStats()
tail(firstInBioc(dls))
```

---

generateBiocPkgDOI	<i>Generate a DOI for a Bioconductor package</i>
--------------------	--

---

### Description

This function makes calls out to the DataCite REST API described here: <https://support.datacite.org/docs/api-create-fois>. The function creates a new DOI for a Bioconductor package (cannot already exist). The target URL for the DOI is the short Bioconductor package URL.

### Usage

```
generateBiocPkgDOI(pkg, authors, pubyear, event = "publish", testing = TRUE)
```

### Arguments

pkg	character(1) package name
authors	character vector of authors (will be "pasted" together)
pubyear	integer(1) publication year
event	Either "hide", "register", or publish". Typically, we use "publish" to make the DOI findable.
testing	logical(1) If true, will use the apitest user with the password apitest. These DOIs will expire. The same apitest:apitest combination can be used to login to the website for doing things using the web interface. If false, the Bioconductor-specific user credentials should be in the correct environment variables

**Details**

The login information for the "real" Bioconductor account should be stored in the environment variables "DATACITE\_USERNAME" and "DATACITE\_PASSWORD"

The GUI is available here: <https://doi.datacite.org/>.

**Value**

The DOI as a character(1) vector.

**Examples**

```
## Not run:
x = generateBiocPkgDOI('RANDOM_TEST_PACKAGE','Sean Davis',1972)

## End(Not run)
```

---

`getBiocVignette`*Download a Bioconductor vignette*

---

**Description**

The actual vignette path is available using [biocPkgList](#).

**Usage**

```
getBiocVignette(
  vignettePath,
  destfile = tempfile(),
  version = BiocManager::version()
)
```

**Arguments**

<code>vignettePath</code>	character(1) the additional path information to get to the vignette
<code>destfile</code>	character(1) the file location to store the vignette
<code>version</code>	character(1) such as "3.7", defaults to user version

**Value**

character(1) The filename of the downloaded vignette

**Examples**

```

x = biocPkgList()
tmp = getBiocVignette(x$vignettes[[1]][1])
tmp

## Not run:
library(pdftools)
y = pdf_text(tmp)
y = paste(y, collapse=" ")
library(tm)
v = VCorpus(VectorSource(y))
library(magrittr)

v <- v %>%
  tm_map(stripWhitespace) %>%
  tm_map(content_transformer(tolower)) %>%
  tm_map(removeWords, stopwords("english")) %>%
  tm_map(stemDocument)
dtm = DocumentTermMatrix(v)
inspect(DocumentTermMatrix(v,
  list(dictionary = as.character(x$Package))))

## End(Not run)

```

---

getPackageInfo

Generate needed information to create DOI from a package directory.

---

**Description**

Generate needed information to create DOI from a package directory.

**Usage**

```
getPackageInfo(dir)
```

**Arguments**

dir                      character(1) Path to package

**Value**

A data.frame

---

get_bioc_data	<i>Get data from Bioconductor</i>
---------------	-----------------------------------

---

**Description**

Get data from Bioconductor

**Usage**

```
get_bioc_data()
```

**Value**

A JSON string containing Bioconductor package details

**Examples**

```
bioc_data <- get_bioc_data()
```

---

get_cre_orcids	<i>get ORCID ids from cre fields of Authors@R in packageDescription results</i>
----------------	---

---

**Description**

get ORCID ids from cre fields of Authors@R in packageDescription results

**Usage**

```
get_cre_orcids(pkgnames)
```

**Arguments**

pkgnames            character() must be installed

**Note**

returns NA if no ORCID provided in Authors@R for package description

**Examples**

```
get_cre_orcids(c("BiocPkgTools", "utils"))
```

githubDetails

*Get package details from GitHub*

---

**Description**

For packages that live on GitHub, we can mine further details. This function returns the GitHub details for the listed packages.

**Usage**

```
githubDetails(pkgs, sleep = 0)
```

**Arguments**

pkgs	a character() vector of username/repo for one or more GitHub repos, such as seandavi/GEOquery.
sleep	numeric() denoting the number of seconds to sleep between GitHub API calls. Since GitHub rate limits its APIs, it might be necessary to either use small chunks of packages iteratively or to supply a non-zero argument here. See the details section for a better solution using GitHub tokens.

**Details**

The `gh` function is used to do the fetching. If the number of packages supplied to this function is large (>40 or so), it is possible to run into problems with API rate limits. The `gh` package uses the environment variable "GITHUB\_PAT" (for personal access token) to authenticate and then provide higher rate limits. If you run into problems with rate limits, set `sleep` to some small positive number to slow queries. Alternatively, create a Personal Access Token on GitHub and register it. See the `gh` package for details.

**Examples**

```
pkglist = biocPkgList()

# example of "pkgs" format.
head(pkglist$URL)

gh_list = githubURLParts(pkglist$URL)
gh_list = gh_list[!is.null(gh_list$user_repo),]

head(gh_list$user_repo)

ghd = githubDetails(gh_list$user_repo[1:5])
lapply(ghd, '[', "stargazers")
```

---

githubURLParts	<i>Extract GitHub user and repo name from GitHub URL</i>
----------------	--

---

**Description**

Extract GitHub user and repo name from GitHub URL

**Usage**

```
githubURLParts(urls)
```

**Arguments**

urls                      character() A vector of URLs

**Value**

A data.frame with four columns:

- url: The original GitHub URL
- user\_repo: The GitHub "username/repo", combined
- user: The GitHub username
- repo: The GitHub repo name

**Examples**

```
# find GitHub URL details for
# Bioconductor packages
bpkg1 = biocPkgList()
urldetails = githubURLParts(bpkg1$URL)
urldetails = urldetails[!is.na(urldetails$url),]
head(urldetails)
```

---

inducedSubgraphByPkgs	<i>Return a minimal subgraph based on package name(s)</i>
-----------------------	---

---

**Description**

Find the subgraph induced by including specific packages. The induced subgraph is the graph that includes the named packages and all edges connecting them. This is useful for a developer, for example, to examine her packages and their intervening dependencies.

**Usage**

```
inducedSubgraphByPkgs(g, pkgs, pkg_color = "red")
```

## Arguments

<code>g</code>	an igraph graph, typically created by <a href="#">buildPkgDependencyIgraph</a>
<code>pkgs</code>	character() vector of packages to include. Package names not included in the graph are ignored.
<code>pkg_color</code>	character(1) giving color of named packages. Other packages in the graph that fall in connecting paths will be colored as the igraph default.

## Examples

```
library(igraph)
g <- buildPkgDependencyIgraph(buildPkgDependencyDataFrame())
## subgraph of only the first 10 packages maintained by Bioconductor
biocmaintained <- head(biocMaintained()[["Package"]], 10L)
g2 <- inducedSubgraphByPkgs(g, pkgs = biocmaintained)
g2
V(g2)

plot(g2)

## subgraph of a package's strong Bioconductor package dependencies
maedeps <- unlist(pkgBiocDeps(
  "MultiAssayExperiment", which = "strong",
  recursive = TRUE, only.bioc = TRUE
), use.names = FALSE)
g3 <- inducedSubgraphByPkgs(g, pkgs = maedeps)
plot(g3)

## same subgraph with networkD3::forceNetwork
library(networkD3)
wt <- cluster_walktrap(g3)
members <- membership(wt)
ndg3 <- igraph_to_networkD3(g3, group = members)
forceNetwork(
  Links = ndg3$links, Nodes = ndg3$nodes, Source = 'source',
  Target = 'target', NodeID = 'name', Group = 'group', zoom = TRUE,
  linkDistance = 200, fontSize = 20, opacity = 0.9, opacityNoHover = 0.9
)
```

---

latestPkgStats

*Summary of the latest package statistics*


---

## Description

The `latestPkgStats` function combines outputs from several functions to generate a table of relevant statistics for a given package.

**Usage**

```
latestPkgStats(
  gh_repo,
  Date,
  pkgType = c("software", "data-experiment", "workflows", "data-annotation")
)
```

**Arguments**

gh_repo	character(1) The GitHub repository location including the username / organization and the repository name, e.g., "Bioconductor/S4Vectors"
Date	character(1) The date cutoff from which to analyze closed issues in the YYYY-MM-DD or YYYY-MM-DDTHH:MM:SSZ format (ISO 8601).
pkgType	character(1) One of 'software', 'data-experiment', 'workflows', or 'data-annotation' (defaults to 'software')

**Examples**

```
if (interactive()) {
  latestPkgStats("Bioconductor/BiocGenerics", "2021-05-05")
}
```

---

orcid_table	<i>get data.frame of employment info from orcid</i>
-------------	---

---

**Description**

get data.frame of employment info from orcid

**Usage**

```
orcid_table(orcid)
```

**Arguments**

orcid	character()
-------	-------------

**Examples**

```
if (interactive()) { # need a token?
  oids <- c("0000-0003-4046-0063", "0000-0003-4046-0063")
  print(orcid_table(oids))
  oids <- c(oids, NA)
  print(orcid_table(oids))
  print(orcid_table(oids[1]))
}
```

---

pkgBiocDeps

Look up a package's Bioconductor dependencies

---

## Description

The function uses the `pkgType` argument to restrict the look up to only the relevant Bioconductor repository. It works for multiple packages of the same type.

## Usage

```
pkgBiocDeps(
  pkg,
  pkgType = c("software", "data-experiment", "workflows", "data-annotation"),
  which = "strong",
  only.bioc = TRUE,
  recursive = FALSE,
  version = BiocManager::version()
)
```

## Arguments

<code>pkg</code>	<code>character(1)</code> The package for which to look up dependencies.
<code>pkgType</code>	<code>character()</code> Any of 'software', 'data-experiment', 'workflows', and / or 'data-annotation' (defaults to all)
<code>which</code>	a character vector listing the types of dependencies, a subset of <code>c("Depends", "Imports", "LinkingTo", "Suggests", "Enhances")</code> . Character string "all" is shorthand for that vector, character string "most" for the same vector without "Enhances", character string "strong" (default) for the first three elements of that vector.
<code>only.bioc</code>	<code>logical(1)</code> Whether to only return Bioconductor dependencies in the list (default TRUE)
<code>recursive</code>	a logical indicating whether (reverse) dependencies of (reverse) dependencies (and so on) should be included, or a character vector like <code>which</code> indicating the type of (reverse) dependencies to be added recursively.
<code>version</code>	(Optional) <code>character(1)</code> or <code>package_version</code> indicating the <i>Bioconductor</i> version (e.g., "3.8") for which repositories are required.

## Examples

```
pkgBiocDeps("MultiAssayExperiment", only.bioc = TRUE)

pkgBiocDeps("MultiAssayExperiment", only.bioc = FALSE)
```

---

pkgBiocRevDeps

---

*Obtain all the reverse dependencies for a Bioconductor package*


---

## Description

The function returns a slightly upgraded list with dependency types as elements and package names in each of those elements, if any. The types of dependencies can be seen in the `which` argument documentation.

## Usage

```
pkgBiocRevDeps(
  pkg,
  pkgType = c("software", "data-experiment", "workflows", "data-annotation"),
  which = "all",
  only.bioc = TRUE,
  version = BiocManager::version()
)

## S3 method for class 'biocrevdeps'
summary(object, ...)
```

## Arguments

<code>pkg</code>	character(1) The package for which to look up dependencies.
<code>pkgType</code>	character() Any of 'software', 'data-experiment', 'workflows', and / or 'data-annotation' (defaults to all)
<code>which</code>	a character vector listing the types of dependencies, a subset of c("Depends", "Imports", "LinkingTo", "Suggests", "Enhances"). Character string "all" is shorthand for that vector, character string "most" for the same vector without "Enhances", character string "strong" (default) for the first three elements of that vector.
<code>only.bioc</code>	logical(1) Whether to only return Bioconductor dependencies in the list (default TRUE)
<code>version</code>	(Optional) character(1) or package_version indicating the <i>Bioconductor</i> version (e.g., "3.8") for which repositories are required.
<code>object</code>	an object for which a summary is desired.
<code>...</code>	additional arguments affecting the summary produced.

## Details

The summary method of the `biocrevdeps` class given by `pkgBiocRevDeps` provides a tally in each dependency field.

**Value**

A biocrevdeps list class object

**Examples**

```
rdeps <- pkgBiocRevDeps("MultiAssayExperiment", which = "all")
rdeps
summary(rdeps)
```

---

`pkgCombDependencyGain` *Calculate dependency gain achieved by excluding combinations of packages*

---

**Description**

Calculate dependency gain achieved by excluding combinations of packages

**Usage**

```
pkgCombDependencyGain(pkg, depdf, maxNbr = 3L)
```

**Arguments**

<code>pkg</code>	character, the name of the package for which we want to estimate the dependency gain
<code>depdf</code>	a tidy data frame with package dependency information obtained through the function <a href="#">buildPkgDependencyDataFrame</a>
<code>maxNbr</code>	numeric, the maximal number of direct dependencies to leave out simultaneously

**Value**

A data frame with three columns: `ExclPackages` (the excluded direct dependencies), `NbrExcl` (the number of excluded direct dependencies), `DepGain` (the dependency gain from excluding these direct dependencies)

**Author(s)**

Charlotte Soneson

**Examples**

```
depdf <- buildPkgDependencyDataFrame(
  dependencies=c("Depends", "Imports"),
  repo=c("BioCsoft", "CRAN")
)
pcd <- pkgCombDependencyGain('GEOquery', depdf, maxNbr = 3L)
head(pcd[order(pcd$DepGain, decreasing = TRUE), ])
```

---

pkgDepImports

*Report package imported functionality*

---

## Description

Function adapted from 'itdepends::dep\_usage\_pkg' at <https://github.com/r-lib/itdepends> to obtain the functionality imported and used by a given package.

## Usage

```
pkgDepImports(pkg)
```

## Arguments

pkg	character() name of the package for which we want to obtain the functionality calls imported from its dependencies and used within the package.
-----	---

## Details

Certain imported elements, such as built-in constants, will not be identified as imported functionality by this function.

## Value

A tidy data frame with two columns:

- pkg: name of the package dependency.
- fun: name of the functionality call imported from the the dependency in the column pkg and used within the analyzed package.

## Author(s)

Robert Castelo

## Examples

```
pkgDepImports('BiocPkgTools')
```

---

pkgDepMetrics

Report package dependency burden

---

## Description

Elaborate a report on the dependency burden of a given package.

## Usage

```
pkgDepMetrics(pkg, depdf)
```

## Arguments

pkg	character() name of the package for which we want to obtain metrics on its dependency burden.
depdf	a tidy data frame with package dependency information obtained through the function <a href="#">buildPkgDependencyDataFrame</a> .

## Value

A tidy data frame with different metrics on the package dependency burden. More concretely, the following columns:

- ImportedAndUsed: number of functionality calls imported and used in the package.
- Exported: number of functionality calls exported by the dependency.
- Usage:  $(\text{ImportedAndUsed} \times 100) / \text{Exported}$ . This value provides an estimate of what fraction of the functionality of the dependency is actually used in the given package.
- DepOverlap: Similarity between the dependency graph structure of the given package and the one of the dependency in the corresponding row, estimated as the **Jaccard index** between the two sets of vertices of the corresponding graphs. Its values goes between 0 and 1, where 0 indicates that no dependency is shared, while 1 indicates that the given package and the corresponding dependency depend on an identical subset of packages.
- DepGainIfExcluded: The 'dependency gain' (decrease in the total number of dependencies) that would be obtained if this package was excluded from the list of direct dependencies.

The reported information is ordered by the Usage column to facilitate the identification of dependencies for which the analyzed package is using a small fraction of their functionality and therefore, it could be easier remove them. To aid in that decision, the column DepOverlap reports the overlap of the dependency graph of each dependency with the one of the analyzed package. Here a value above, e.g., 0.5, could, albeit not necessarily, imply that removing that dependency could substantially lighten the dependency burden of the analyzed package.

An NA value in the ImportedAndUsed column indicates that the function pkgDepMetrics() could not identify what functionality calls in the analyzed package are made to the dependency.

**Author(s)**

Robert Castelo  
Charlotte Soneson

**Examples**

```
depdf <- buildPkgDependencyDataFrame(
  dependencies=c("Depends", "Imports"),
  repo=c("BioCsoft", "CRAN")
)
pkgDepMetrics('BiocPkgTools', depdf)
```

---

pkgDownloadRank

*What is a package's download rank?*

---

**Description**

This function uses available `.packages` to calculate the download rank *percentile* of a given package. It approximates what is observed in the Bioconductor landing page.

**Usage**

```
pkgDownloadRank(
  pkg,
  pkgType = c("software", "data-experiment", "workflows", "data-annotation"),
  version = BiocManager::version()
)
```

**Arguments**

<code>pkg</code>	character(1) The name of a Bioconductor package
<code>pkgType</code>	character(1) One of 'software', 'data-experiment', 'workflows', or 'data-annotation' (defaults to 'software')
<code>version</code>	(Optional) character(1) or <code>package_version</code> indicating the <i>Bioconductor</i> version (e.g., "3.8") for which repositories are required.

**Value**

The package's percentile rank, in terms of download statistics, and proportion in the name

**Examples**

```
## Percentile rank for BiocGenerics (top 1%)
pkgDownloadRank("BiocGenerics", "software")
```

---

pkgDownloadStats	<i>Get Bioconductor download statistics for a package</i>
------------------	---

---

## Description

Get Bioconductor download statistics for a package

## Usage

```
pkgDownloadStats(
  pkg,
  pkgType = c("software", "data-experiment", "workflows", "data-annotation"),
  years = format(Sys.time(), "%Y")
)
```

## Arguments

pkg	character(1) The name of a Bioconductor package
pkgType	character(1) One of 'software', 'data-experiment', 'workflows', or 'data-annotation' (defaults to 'software')
years	numeric(), character() A vector of years from which to obtain download statistics (defaults to current year)

## Value

A tibble of download statistics

## Examples

```
pkgDownloadStats("GenomicRanges")
```

---

problemPage	<i>generate hyperlinked HTML for build reports for Bioc packages</i>
-------------	--

---

## Description

This is a quick way to get an HTML report of packages maintained by a specific developer or which depend directly on a specified package. The function is keyed to filter based on either the maintainer name or by using the 'Depends', 'Suggests' and 'Imports' fields in package descriptions.

**Usage**

```
problemPage(
  authorPattern = "V.*Carey",
  dependsOn,
  ver = "devel",
  includeOK = FALSE
)
```

**Arguments**

authorPattern	character(1) regexp used with grep() to filter author field of package DESCRIPTION for listing
dependsOn	character(1) name of a Bioconductor package. The function will return the status of packages that directly depend on this package. Can only be used when 'authorPattern' is the empty string.
ver	character(1) version tag for Bioconductor
includeOK	logical(1) include entries from the build report that are listed as "OK". Default FALSE will result in only those entries that are in WARNING or ERROR state.

**Value**

DT::datatable call; if assigned to a variable, must evaluate to get the page to appear

**Author(s)**

Vince Carey, Mike L. Smith

**Examples**

```
if (interactive()) {
  problemPage()
  problemPage(dependsOn = "limma")
}
```

---

 repositoryStats

*Bioconductor Binary Repository Statistics*


---

**Description**

Summarize binary packages compatible with the Bioconductor or Terra container in use.

**Usage**

```

repositoryStats(
  version = BiocManager::version(),
  binary_repository = BiocManager::containerRepository(version)
)

## S3 method for class 'repositoryStats'
print(x, ...)

```

**Arguments**

version	(Optional) character(1) or package_version indicating the <i>Bioconductor</i> version (e.g., "3.8") for which repositories are required.
binary_repository	character(1) location of binary repository as given by BiocManager::containerRepository (default)
x	the object returned by repositoryStats().
...	further arguments passed to or from other methods (not used).

**Value**

a list of class repositoryStats with the following fields:

- container: character(1) container label, e.g., bioconductor\_docker, or NA if not evaluated on a supported container
- bioconductor\_version: package\_version the Bioconductor version provided by the user.
- repository\_exists: logical(1) TRUE if a binary repository exists for the container and Bioconductor\_Version version.
- bioconductor\_binary\_repository: character(1) repository location, if available, or NA if the repository does not exist.
- n\_software\_packages: integer(1) number of software packages in the Bioconductor source repository.
- n\_binary\_packages: integer(1) number of binary packages available. When a binary repository exists, this number is likely to be larger than the number of source software packages, because it includes the binary version of the source software packages, as well as the (possibly CRAN) dependencies of the binary packages
- n\_binary\_software\_packages: integer(1) number of binary packages derived from Bioconductor source packages. This number is less than or equal to n\_software\_packages.
- missing\_binaries: integer(1) the number of Bioconductor source software packages that are not present in the binary repository.
- out\_of\_date\_binaries: integer(1) the number of Bioconductor source software packages that are newer than their binary counterpart. A newer source software package might occur when the main Bioconductor build system has updated a package after the most recent run of the binary build system.

**Methods (by generic)**

- `print(repositoryStats)`: Print a summary of package availability in binary repositories.

**Author(s)**

M. Morgan

**Examples**

```
stats <- repositoryStats() # obtain statistics
stats          # display a summary
stats$container # access an element for further computation
```

---

subgraphByDegree	<i>Subset graph by degree</i>
------------------	-------------------------------

---

**Description**

While the [inducedSubgraphByPkgs](#) returns the subgraph with the minimal connections between named packages, this function takes a vector of package names, a degree (1 or more) and returns the subgraph(s) that are within degree of the package named.

**Usage**

```
subgraphByDegree(g, pkg, degree = 1, ...)
```

**Arguments**

<code>g</code>	an igraph graph, typically created by <a href="#">buildPkgDependencyIgraph</a>
<code>pkg</code>	character(1) package name from which to measure degree.
<code>degree</code>	integer(1) degree, limit search for adjacent vertices to this degree.
<code>...</code>	passed on to <a href="#">distances</a>

**Value**

An igraph graph, with only nodes and their edges within degree of the named package

**Examples**

```
g = buildPkgDependencyIgraph(buildPkgDependencyDataFrame())
g2 = subgraphByDegree(g, 'GEOquery')
plot(g2)
```

---

templatePath	<i>Obtain the location of available email templates</i>
--------------	---

---

**Description**

These templates are used with `biocBuildEmail` to notify maintainers regarding package errors and final deprecation warning.

**Usage**

```
templatePath(  
  type = c("buildemail", "deprecation", "deprecguide", "cranreport", "revdeprnote")  
)
```

**Arguments**

type	character(1) Either one of "buildemail", "deprecation", "deprecguide", "cranreport", or "revdeprnote". See the templates in the resources folder.
------	---

# Index

## \* Internal

generateBiocPkgDOI, 24

getPackageInfo, 26

## \* internal

.getDepGain, 3

dataciteXMLGenerate, 23

.getDepGain, 3

.get\_cre\_orcid, 4

.get\_orcid\_rec, 4

activitySince, 5

anacondaDownloadStats, 6

biocBuildEmail, 7

biocBuildReport, 9, 16

biocBuildReport(), 10

biocBuildReportDB, 10

biocBuildStatusDB, 10

biocDownloadStats, 6, 11, 16, 24

biocExplore, 12

biocMaintained, 13

biocPkgList, 14, 16, 19, 20, 25

biocPkgRanges, 15

BiocPkgTools, 16

BiocPkgTools-cache, 17

BiocPkgTools-package (BiocPkgTools), 16

biocRevDepEmail, 18

buildClassDepData (class-dependencies),  
22

buildClassDepFromPackage  
(class-dependencies), 22

buildClassDepGraph  
(class-dependencies), 22

buildPkgDependencyDataFrame, 19, 21, 34,  
36

buildPkgDependencyIgraph, 20, 20, 30, 41

class-dependencies, 22

CRANstatus, 22

dataciteXMLGenerate, 23

distances, 41

firstInBioc, 24

generateBiocPkgDOI, 24

get\_bioc\_data, 27

get\_cre\_orcids, 27

getBiocVignette, 25

getPackageInfo, 26

gh, 28

githubDetails, 28

githubURLParts, 29

graph\_from\_data\_frame, 21

hasBiocMaint (biocMaintained), 13

inducedSubgraphByPkgs, 21, 29, 41

latestPkgStats, 30

orcid\_table, 31

pkgBiocDeps, 32

pkgBiocRevDeps, 33

pkgCombDependencyGain, 34

pkgDepImports, 35

pkgDepMetrics, 36

pkgDownloadRank, 37

pkgDownloadStats, 38

pkgToolsCache (BiocPkgTools-cache), 17

plotClassDep (class-dependencies), 22

plotClassDepData (class-dependencies),  
22

plotClassDepGraph (class-dependencies),  
22

print.repositoryStats  
(repositoryStats), 39

problemPage, 38

repositoryStats, 39

sentHistory (biocBuildEmail), 7

`setCache (BiocPkgTools-cache)`, [17](#)  
`subgraphByDegree`, [27](#), [41](#)  
`summary.biocrevdeps (pkgBiocRevDeps)`, [33](#)  
`templatePath`, [42](#)