

# Package ‘updateObject’

May 17, 2024

**Title** Find/fix old serialized S4 instances

**Description** A set of tools built around `updateObject()` to work with old serialized S4 instances. The package is primarily useful to package maintainers who want to update the serialized S4 instances included in their package. This is still work-in-progress.

**biocViews** Infrastructure, DataRepresentation

**URL** <https://bioconductor.org/packages/updateObject>

**BugReports** <https://github.com/Bioconductor/updateObject/issues>

**Version** 1.8.0

**License** Artistic-2.0

**Encoding** UTF-8

**Depends** R (>= 4.2.0), methods, BiocGenerics (>= 0.47.1), S4Vectors

**Imports** utils, digest

**Suggests** GenomicRanges, SummarizedExperiment, InteractionSet, SingleCellExperiment, MultiAssayExperiment, TimiRGeN, testthat, knitr, rmarkdown, BiocStyle

**SystemRequirements** git

**VignetteBuilder** knitr

**Collate** capture\_message.R updateSerializedObjects.R bump\_pkg\_version.R  
find\_python.R git-utils.R updatePackageObjects.R  
updateBiocPackageRepoObjects.R

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

**git\_branch** RELEASE\_3\_19

**git\_last\_commit** fa153a7

**git\_last\_commit\_date** 2024-04-30

**Repository** Bioconductor 3.19

**Date/Publication** 2024-05-16

**Author** Hervé Pagès [aut, cre]

**Maintainer** Hervé Pagès <[hpages.on.github@gmail.com](mailto:hpages.on.github@gmail.com)>

## Contents

|                              |    |
|------------------------------|----|
| bump_pkg_version             | 2  |
| git-utils                    | 3  |
| updateBiocPackageRepoObjects | 5  |
| updatePackageObjects         | 9  |
| updateSerializedObjects      | 12 |

|              |           |
|--------------|-----------|
| <b>Index</b> | <b>15</b> |
|--------------|-----------|

---

|                  |                                      |
|------------------|--------------------------------------|
| bump_pkg_version | <i>Bump the version of a package</i> |
|------------------|--------------------------------------|

---

### Description

Use `bump_pkg_version()` to bump the version of a package.

### Usage

```
bump_pkg_version(pkgpath=".", update.Date=FALSE)
```

### Arguments

|                          |   |
|--------------------------|---|
| <code>pkgpath</code>     | The path (as a single string) to the top-level directory of an R package source tree.     |
| <code>update.Date</code> | TRUE or FALSE. If TRUE then the Date field (if present) gets updated to the current date. |

### Value

An invisible NULL.

### See Also

- [updatePackageObjects](#) which uses `bump_pkg_version` internally when `bump.Version=TRUE`.
- [git\\_commit](#) for an example.

### Examples

```
## Create dummy R package:
create_dummy_pkg <- function(desc, pkgpath) {
  dir.create(pkgpath)
  descpath <- file.path(pkgpath, "DESCRIPTION")
  write.dcf(rbind(desc), descpath)
  descpath
}

pkgname <- "Dummy"
desc <- c(
```

```
Package=pkgname,
Title="Not a real package",
Description="I'm not real u know.",
Version="3.0.9",
Date="1969-07-20"
)
pkgpath <- file.path(tempdir(), pkgname)
descpath <- create_dummy_pkg(desc, pkgpath)

## Bump its Version:
bump_pkg_version(pkgpath)
cat(readLines(descpath), sep="\n")

## Bump its Version again and set Date to current date:
bump_pkg_version(pkgpath, update.Date=TRUE)
cat(readLines(descpath), sep="\n")

## Throw it away:
unlink(pkgpath, recursive=TRUE)
```

---

|           |   |
|-----------|---|
| git-utils | <i>Convenience Git-related utility functions used by updateBiocPackageRepoObjects()</i> |
|-----------|---|

---

## Description

`prepare_git_repo_for_work()` and `git_commit()` are used internally by `updateBiocPackageRepoObjects()` to perform Git operations.

## Usage

```
prepare_git_repo_for_work(repopath=".", branch=NULL, git=NULL,
                          use.https=FALSE)
```

```
git_commit(repopath=".", commit_msg, push=FALSE,
           git=NULL, user_name=NULL, user_email=NULL)
```

## Arguments

|          |  |
|----------|--|
| repopath | The path (as a single string) to the local Git repository of a Bioconductor package.<br>If the specified path exists, <code>prepare_git_repo_for_work()</code> will check that it's a workable Git repo (i.e. contains no uncommitted changes). If that's the case then it will call <code>git pull</code> on it, otherwise it will return an error.<br>If the specified path does not exist, <code>prepare_git_repo_for_work()</code> will try to infer the package name from <code>repopath</code> and clone it from <a href="https://git.bioconductor.org">git.bioconductor.org</a> . |
|----------|--|

|                       |   |
|-----------------------|---|
| branch                | The branch (as a single string) of the Git repository to work on.<br>If NULL, then the current branch is used (if repopath already exists) or the default branch is used (if repopath does not exist and needs to be cloned).   |
| git                   | The path (as a single string) to the git command if it's not in the PATH.   |
| use.https             | By default, <code>git clone git@git.bioconductor.org:packages/MyPackage</code> is used to clone a package repo from the Bioconductor Git server. Note that this works only for authorized maintainers of package <b>MyPackage</b> . By setting <code>use.https</code> to TRUE, the package will be cloned instead from <code>https://git.bioconductor.org:package</code> which should work for anybody, but then pushing back the changes to the package won't be possible. |
| commit_msg            | The Git commit message.   |
| push                  | Whether to push the changes or not. Changes are committed but not pushed by default. You need push access to the package Git repository at <a href="https://git.bioconductor.org">git.bioconductor.org</a> in order to use <code>push=TRUE</code> .   |
| user_name, user_email | Set the Git user name and/or email to use for the commit. This overrides the Git user name and/or email that the git command would otherwise use. See the COMMIT INFORMATION section in <code>system2("git", c("commit", "--help"))</code> for the details about where the git command normally takes this information from.  |

### Value

`prepare_git_repo_for_work()` returns FALSE if the supplied path already exists, and TRUE if it didn't exist and needed to be cloned.

`git_commit()` returns an invisible NULL.

### See Also

- [updateBiocPackageRepoObjects](#) which uses `prepare_git_repo_for_work` and `git_commit` internally.
- [bump\\_pkg\\_version](#).

### Examples

```
repopath <- file.path(tempdir(), "IdeoViz")

## We must use HTTPS access to clone the package because we are
## not maintainers of the IdeoViz package. A more realistic situation
## would be to use prepare_git_repo_for_work() on a package that we
## maintain, in which case 'use.https=TRUE' would not be needed:
prepare_git_repo_for_work(repopath, use.https=TRUE)

bump_pkg_version(repopath, update.Date=TRUE)

git_commit(repopath, commit_msg="version bump", push=FALSE)

unlink(repopath, recursive=TRUE)
```

---

 updateBiocPackageRepoObjects

*Update the serialized objects contained in a Bioconductor package Git repository or in a set of Bioconductor package Git repositories*

---

## Description

updateBiocPackageRepoObjects() and updateAllBiocPackageRepoObjects() are wrappers to [updatePackageObjects\(\)](#) and [updateAllPackageObjects\(\)](#) that take care of committing and pushing the changes made to the package(s).

## Usage

```
updateBiocPackageRepoObjects(repopath=".", branch=NULL, filter=NULL,
                             commit_msg=NULL, push=FALSE, remove.clone.on.success=FALSE,
                             git=NULL, use.https=FALSE, user_name=NULL, user_email=NULL)
```

```
updateAllBiocPackageRepoObjects(all_repopaths=".", skipped_repos=NULL, ...)
```

## Arguments

|                         |   |
|-------------------------|---|
| repopath                | The path (as a single string) to the local Git repository of a Bioconductor package. See <a href="#">?prepare_git_repo_for_work</a> for more information.   |
| branch                  | The branch (as a single string) of the Git repository to work on. See <a href="#">?prepare_git_repo_for_work</a> for more information.  |
| filter                  | See <a href="#">?updatePackageObjects</a> .   |
| commit_msg              | The Git commit message. By default "Pass serialized S4 instances thru updateObject()" is used.  |
| push                    | Whether to push the changes or not. Changes are committed but not pushed by default. You need push access to the package Git repository at <a href="#">git.bioconductor.org</a> in order to use push=TRUE.          |
| remove.clone.on.success | Whether to remove the Git clone on success or not. Only applies if repopath does not exist and needs to be cloned.  |
| git, use.https          | See <a href="#">?prepare_git_repo_for_work</a> .  |
| user_name, user_email   | See <a href="#">?git_commit</a> .   |
| all_repopaths           | Character vector of paths to local Git repositories of Bioconductor packages.   |
| skipped_repos           | Character vector of repository paths to ignore.   |
| ...                     | updateAllBiocPackageRepoObjects() walks over the all_repopaths vector and calls updateBiocPackageRepoObjects() on each repository path. All the arguments in ... are passed down to updateBiocPackageRepoObjects(). |

**Value**

updateBiocPackageRepoObjects() and updateAllBiocPackageRepoObjects() are wrappers to [updatePackageObjects\(\)](#) and [updateAllPackageObjects\(\)](#), respectively, and return the same value.

**See Also**

- [updatePackageObjects](#) and [updateAllPackageObjects](#).
- Utility functions [prepare\\_git\\_repo\\_for\\_work](#) and [git\\_commit](#) which are used internally by updateBiocPackageRepoObjects to perform the Git operations.

**Examples**

```
## -----
## updateBiocPackageRepoObjects()
## -----

## Typical use, assuming MyPackage is a Bioconductor package that you
## maintain:
## Not run:
  repopath <- file.path(tempdir(), "MyPackage")
  updateBiocPackageRepoObjects(repopath, push=TRUE)

## End(Not run)

## Note that by default `updateBiocPackageRepoObjects()` does NOT try
## to push the changes to git.bioconductor.org. Only the authorized
## maintainers of MyPackage can do that. In the examples below we
## must use HTTPS access to clone the package because we are not
## maintainers of the CellBench or BiocGenerics packages. Also we
## don't use 'push=TRUE' because we are not allowed to do that (it
## wouldn't work anyways).

## On a package with a mix of RDS and RDA files:
repopath <- file.path(tempdir(), "CellBench")
updateBiocPackageRepoObjects(repopath, branch="RELEASE_3_13",
                             remove.clone.on.success=TRUE,
                             use.https=TRUE)

## On a package with no serialized objects:
repopath <- file.path(tempdir(), "BiocGenerics")
updateBiocPackageRepoObjects(repopath, branch="RELEASE_3_13",
                             remove.clone.on.success=TRUE,
                             use.https=TRUE)

## Note that the RELEASE_3_13 branch of all Bioconductor packages got
## frozen in October 2021. The above examples are for illustrative
## purpose only. A more realistic situation would be to use
## updateBiocPackageRepoObjects() on the development version (i.e.
## the master branch) of a package that you maintain, and to push the
## changes by calling the function with 'push=TRUE'.
```

```
## -----  
## updateAllBiocPackageRepoObjects()  
## -----  
  
## Let's assume that the current directory is populated with the  
## Git repositories of all Bioconductor software packages and that  
## we have push access to them:  
  
ALL_REPOS <- dir() # get list of package repos to update  
  
READ_RDS_FAILURE <- c(  
  "BindingSiteFinder",  
  "ChIPpeakAnno",  
  "drugTargetInteractions"  
)  
  
LOAD_FAILURE <- c(  
  "AlphaBeta",  
  "CellaRepertorium",  
  "CNVRanger",  
  "gscreeend",  
  "HiLDA",  
  "immunotation",  
  "MAST",  
  "midasHLA",  
  "mixOmics",  
  "oligoClasses",  
  "TitanCNA",  
  "Uniquorn"  
)  
  
UPDATEOBJECT_FAILURE <- c(  
  "ACE",  
  "AnnotationHubData",  
  "arrayMvout",  
  "Autotuner",  
  "BASiCS",  
  "bigmelon",  
  "Biobase",  
  "CAMERA",  
  "categoryCompare",  
  "cellHTS2",  
  "cellmigRation",  
  "CEMiTool",  
  "CeTF",  
  "cleanUpdTSeq",  
  "CoGAPS",  
  "CoreGx",  
  "CrisprVariants",  
  "crlmm",  
  "decompTumor2Sig",  
  "DIAalignR",
```

```

    "enhancerHomologSearch",
    "fcoex",
    "geNetClassifier",
    "GreyListChIP",
    "GSgalgoR",
    "hmbdQuery",
    "iCOBRA",
    "MassArray",
    "midasHLA",
    "MinimumDistance",
    "MSnbase",
    "msPurity",
    "multiHiCcompare",
    "musicatk",
    "MutationalPatterns",
    "openPrimerR",
    "OSAT",
    "PharmacoGx",
    "pipeFrame",
    "ProteoDisco",
    "puma",
    "qcmetrics",
    "QDNaseq",
    "r3Cseq",
    "RadioGx",
    "RTN",
    "sangeranalyseR",
    "synapter",
    "tigre",
    "topGO",
    "ToxicoGx",
    "VariantFiltering",
    "watermelon",
    "xcms"
)

## Contain files to push larger than 5 Mb.
PUSH_FAILURE <- c(
  "BiocSklearn",
  "BubbleTree",
  "CINdex",
  "erma",
  "ivygapSE",
  "SplicingGraphs",
  "vtpnet"
)

## Skipped for other reasons e.g. contain objects for which
## updateObject() takes forever or the package needs to be
## installed but cannot at the moment.
OTHER_SKIPPED_REPOS <- c(
  "BaalChIP", "BiGGR", "CytoTree", "gwascat",
  "mirIntegrator", "oposSOM", "PFP", "ROntoTools", "SLGI"
)

```



```

)

SKIPPED_REPOS <- c(
  READ_RDS_FAILURE,
  LOAD_FAILURE,
  UPDATEOBJECT_FAILURE,
  PUSH_FAILURE,
  OTHER_SKIPPED_REPOS
)

FILTER <- "\bDataFrame\b"

## Not run:
system.time(
  codes <- updateAllBiocPackageRepoObjects(ALL_REPOS,
                                           skipped_repos=SKIPPED_REPOS,
                                           branch="master",
                                           filter=FILTER,
                                           push=TRUE)
)

## End(Not run)

```

---

updatePackageObjects *Update the serialized objects contained in a package or in a set of packages*

---

## Description

Use `updatePackageObjects()` to update all the serialized objects contained in a package.

Use `updateAllPackageObjects()` to update all the serialized objects contained in a set of packages.

## Usage

```

updatePackageObjects(pkgpath=".", filter=NULL,
                    dry.run=FALSE, bump.Version=FALSE)

updateAllPackageObjects(all_pkgpaths, skipped_pkgs=NULL, filter=NULL,
                    dry.run=FALSE, bump.Version=FALSE)

```

## Arguments

|  |   |
|--|---|
| <code>pkgpath</code>                       | The path (as a single string) to the top-level directory of an R package source tree.   |
| <code>filter</code> , <code>dry.run</code> | These arguments are passed down to <code>updateSerializedObjects()</code> . See <a href="#">?updateSerializedObjects</a> for the details. |

`bump.Version` TRUE or FALSE. If TRUE and if some RDS or RDA files in the package actually get updated by `updateSerializedObjects()`, then the package version will get bumped, that is, the `Version` field in its DESCRIPTION file will get bumped from X.Y.Z to X.Y.(Z+1). For example, version 2.0.9 will become 2.0.10. Additionally, the `Date` field (if present) will get updated to the current date.

`all_pkgpaths` Character vector of package paths.

`skipped_pkgs` Character vector of package paths to ignore.

### Value

`updatePackageObjects()` returns the value returned by its call to `updateSerializedObjects()`. See [?updateSerializedObjects](#) for the details.

`updateAllPackageObjects()` returns a named integer vector *parallel* to `all_pkgpaths`.

### See Also

- The [updateSerializedObjects](#) function which is the workhorse behind `updatePackageObjects`.
- [updateBiocPackageRepoObjects](#) and [updateAllBiocPackageRepoObjects](#) which are wrapper functions that also take care of committing and pushing the changes made to the packages.
- The [bump\\_pkg\\_version](#) function which is used internally by `updatePackageObjects` and `updateAllPackageObjects` when `bump.Version=TRUE`.

### Examples

```
## -----
## A SIMPLE updatePackageObjects() EXAMPLE
## -----

## DemoPackage is a small demo package (contained in the updateObject
## package) with some old serialized GRanges objects in it.
pkgname <- "DemoPackage"
pkgpath0 <- system.file(pkgname, package="updateObject")

## Let's copy DemoPackage to a writable location.
pkgpath <- file.path(tempdir(), pkgname)
file.copy(pkgpath0, dirname(pkgpath), recursive=TRUE)

## Note that, in order to update the GRanges objects contained in
## DemoPackage, updatePackageObjects() will need to attach the
## GenomicRanges package. That's because this is where the GRanges
## class and updateObject() method for GRanges objects are both
## defined. See '?updateSerializedObjects' for more information.
## Also note that we don't need to perform two passes ("dry run" +
## "real run"), one pass is enough. Here we show the 2-pass procedure
## for illustrative purpose only.

## 1st pass: dry run
code <- updatePackageObjects(pkgpath, dry.run=TRUE)
code # a non-negative code means everything went fine
```

```
## 2nd pass: do it for good!
updatePackageObjects(pkgpath, bump.Version=TRUE)

## An additional run would only confirm that there's nothing left
## to update.
code <- updatePackageObjects(pkgpath)
code # 0 (no files to update)

unlink(pkgpath, recursive=TRUE)

## -----
## FIND CANDIDATE PACKAGES IN CURRENT DIRECTORY
## -----
## Not run:
## In this example we perform a "dry run" with updateAllPackageObjects()
## to find all the packages in a directory that contain old serialized
## objects.

## Let's assume that the current directory is populated with package
## git clones:
all_pkgs <- dir() # get list of packages

## If we know that some packages are going to cause problems, we should
## skip them. Note that we could just do
##
## all_pkgs <- setdiff(all_pkgs, SKIPPED_PKGS)
##
## for this. However, by using the 'skipped_pkgs' argument, all the
## packages in the original 'all_pkgs' will be represented in the
## returned vector, including the skipped packages:
SKIPPED_PKGS <- c(
  "BaalChIP", "BiGGR", "CytoTree", "gwascat",
  "mirIntegrator", "oposSOM", "PFP", "ROntoTools", "SLGI"
)

## --- Without a filter ---

## updateAllPackageObjects() will stop with an error if a package is
## required but not installed. The user is responsible for installing
## all the required packages (this is admittedly hard to know in advance).
codes <- updateAllPackageObjects(all_pkgs, skipped_pkgs=SKIPPED_PKGS,
                                dry.run=TRUE)

sessionInfo() # many packages
table(codes)

## The above code was successfully run in the MEAT0 folder on nebbiol01
## (BioC 3.15, 2067 packages) on Nov 18, 2021:
## - took about 14 min
## - loaded 1190 packages (as reported by sessionInfo())
## - required about 9GB of RAM
##
## > table(codes)
```

```
## codes
## codes
##   -3  -2  -1   0   1   2   3   4   5   6   7   8   9  10
##    4  15  66 1549 240  90  46  28   7   5   4   3   3   1
##   13  18  20  23  125
##    2   1   1   1   1
##
## > sum(codes > 0) / length(codes)
## [1] 0.2094823
## 21

## --- With a filter ---

## We want to filter on the presence of the word "DataFrame" in
## the output of 'updateObject( , check=FALSE, verbose=TRUE)'. We can't
## just set 'filter' to '"DataFrame"' for that as this would also produce
## matches in the presence of strings like "AnnotatedDataFrame":
filter <- "\bDataFrame\b"

codes <- updateAllPackageObjects(all_pkgs, skipped_pkgs=SKIPPED_PKGS,
                                filter=filter,
                                dry.run=TRUE)

## End(Not run)
```

---

updateSerializedObjects

*Update the serialized objects contained in a directory*

---

## Description

Use `updateSerializedObjects()` to find and update all the serialized objects contained in a directory. This is the workhorse behind higher-level functions `updatePackageObjects()` and family (`updateAllPackageObjects()`, `updateBiocPackageRepoObjects()`, and `updateAllBiocPackageRepoObjects()`). `collect_rds_files()`, `collect_rda_files()`, `update_rds_file()`, and `update_rda_file()` are the low-level utilities used internally by `updateSerializedObjects()` to do the job.

## Usage

```
updateSerializedObjects(dirpath=".", recursive=FALSE,
                       filter=NULL, dry.run=FALSE)
```

```
## Low-level utilities upon which updateSerializedObjects() is built:
collect_rds_files(dirpath=".", recursive=FALSE)
collect_rda_files(dirpath=".", recursive=FALSE)
update_rds_file(filepath, filter=NULL, dry.run=FALSE)
update_rda_file(filepath, filter=NULL, dry.run=FALSE)
```

**Arguments**

|                        |   |
|------------------------|---|
| <code>dirpath</code>   | The path (as a single string) to an arbitrary directory.  |
| <code>recursive</code> | TRUE or FALSE. Should the directory be searched recursively to find the objects to update? By default the directory is <i>not</i> searched recursively.   |
| <code>filter</code>    | NULL (the default) or a single string containing a regular expression.<br>When <code>filter</code> is set, only objects for which there is a match in the output of <code>updateObject(object, check=FALSE, verbose=TRUE)</code> actually get replaced with the object returned by the <code>updateObject</code> call. See Details section below for more on this.<br>Note that the pattern matching is <i>case sensitive</i> . |
| <code>dry.run</code>   | TRUE or FALSE. By default, updated objects are written back to their original file. Set <code>dry.run</code> to TRUE to perform a trial run with no changes made.   |
| <code>filepath</code>  | The path (as a single string) to a file containing serialized objects. This must be an RDS file (for <code>update_rds_file</code> ) or RDA file (for <code>update_rda_file</code> ).  |

**Details**

`update_rds_file()` and `update_rda_file()` use `updateObject()` internally to update individual R objects.

If no `filter` is specified (the default), each object is updated with `object <- updateObject(object, check=FALSE)`. If that turns out to be a no-op, then code 0 ("nothing to update") is returned. Otherwise 1 is returned.

If a `filter` is specified (via the `filter` argument) then `updateObject(object, check=FALSE, verbose=TRUE)` is called on each object and the output of the call is captured with `capture.output()`. Only if the output contains a match for `filter` is the object replaced with the object returned by the call. If this replacement turns out to be a no-op, or if the output contained no match for `filter`, then code 0 ("nothing to update") is returned. Otherwise 1 is returned.

The pattern matching is *case sensitive*.

Note that determining whether a call to `updateObject()` is a no-op or not is done by calling `digest::digest()` on the original object and object returned by `updateObject()`, and by comparing the 2 hash values. This is a LOT MORE reliable than using `identical()` which is notoriously unreliable!

**Value**

`updateSerializedObjects()` returns a single integer which is the number of updated files or a negative error code (-2 if loading an RDS or RDA file failed, -1 if `updateObject()` returned an error).

`collect_rds_files()` and `collect_rda_files()` return a character vector of (relative) file paths.

`update_rds_file()` and `update_rda_file()` return a single integer which is one of the following codes:

- -2 if loading the RDS or RDA file failed;
- -1 if `updateObject()` returned an error;
- 0 if there was nothing to update in the file;
- 1 if the file got updated.

**See Also**

- The [updatePackageObjects](#) function which is just a thin wrapper around `updateSerializedObjects`.
- The [updateObject](#) generic function in the **BiocGenerics** package.
- The [capture.output](#) function in the **utils** package.
- The [digest](#) function in the **digest** package.

**Examples**

```

dirpath <- system.file("extdata", package="updateObject")

## -----
## WITHOUT A FILTER
## -----

## updateSerializedObjects() prints one line per processed file:
updateSerializedObjects(dirpath, recursive=TRUE, dry.run=TRUE)

## Note that updateSerializedObjects() needs to attach/load the packages
## in which the classes of the objects to update are defined. These
## packages are: GenomicRanges for GRanges objects, SummarizedExperiment
## for SummarizedExperiment objects, and InteractionSet for GInteractions
## objects. This means that sessionInfo() will typically report more
## attached and loaded packages after a updateSerializedObjects() run
## than before:
sessionInfo()

## Also updateSerializedObjects() will raise an error if it fails to
## attach or load a package (typically because the package is missing).
## It will NOT try to install the package.

## -----
## WITH A FILTER
## -----

## We want to filter on the presence of the word "DataFrame" in
## the output of 'updateObject( , check=FALSE, verbose=TRUE)'. We can't
## just set 'filter' to '"DataFrame"' for that as this would also produce
## matches in the presence of strings like "AnnotatedDataFrame":
filter <- "\bDataFrame\b"

updateSerializedObjects(dirpath, recursive=TRUE, filter=filter,
                       dry.run=TRUE)

```

# Index

## \* utilities

- bump\_pkg\_version, [2](#)
  - git-utils, [3](#)
  - updateBiocPackageRepoObjects, [5](#)
  - updatePackageObjects, [9](#)
  - updateSerializedObjects, [12](#)
- bump\_pkg\_version, [2](#), [4](#), [10](#)
- capture.output, [14](#)
- collect\_rda\_files  
(updateSerializedObjects), [12](#)
- collect\_rds\_files  
(updateSerializedObjects), [12](#)
- digest, [13](#), [14](#)
- git-utils, [3](#)
- git\_commit, [2](#), [5](#), [6](#)
- git\_commit (git-utils), [3](#)
- prepare\_git\_repo\_for\_work, [5](#), [6](#)
- prepare\_git\_repo\_for\_work (git-utils), [3](#)
- set\_git\_user\_email (git-utils), [3](#)
- set\_git\_user\_name (git-utils), [3](#)
- unset\_git\_user\_email (git-utils), [3](#)
- unset\_git\_user\_name (git-utils), [3](#)
- update\_rda\_file  
(updateSerializedObjects), [12](#)
- update\_rds\_file  
(updateSerializedObjects), [12](#)
- updateAllBiocPackageRepoObjects, [10](#), [12](#)
- updateAllBiocPackageRepoObjects  
(updateBiocPackageRepoObjects),  
[5](#)
- updateAllPackageObjects, [5](#), [6](#), [12](#)
- updateAllPackageObjects  
(updatePackageObjects), [9](#)
- updateBiocPackageRepoObjects, [3](#), [4](#), [5](#), [10](#),  
[12](#)
- updateObject, [13](#), [14](#)
- updatePackageObjects, [2](#), [5](#), [6](#), [9](#), [12](#), [14](#)
- updateSerializedObjects, [9](#), [10](#), [12](#)