Package 'rotor'

June 17, 2019

Type Package

Title Log Rotation and Conditional Backups

Version 0.2.3

Maintainer Stefan Fleck <stefan.b.fleck@gmail.com>

Description Conditionally rotate or back-up files based on
their size or the date of the last backup; inspired by the 'Linux'
utility 'logrotate'.

License MIT + file LICENSE

URL https://s-fleck.github.io/rotor/

BugReports https://github.com/s-fleck/rotor/issues

Imports dint, R6, tools

Suggests covr, crayon, rmarkdown, testthat, withr, zip

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

NeedsCompilation no

Author Stefan Fleck [aut, cre] (<https://orcid.org/0000-0003-3344-9851>)

Repository CRAN

Date/Publication 2019-06-17 09:40:04 UTC

R topics documented:

BackupQueue *An R6 class for managing backups*

Description

BackupQueue & co are part of the R6 API of **rotor**. They are used internally by `rotate()` and related functions and are not designed for interactive use. Rather, if you are a package developer and want to integrate rotor in one of your package, the BackupQueue subclasses give you a bit of extra control.

As of now, **the R6 API is still experimental and subject to change**.

Methods

`pad_index()` Pad the indices in the filenames of indexed backups to the number of digits of the largest index. Usually does not have to be called manually.

`prune()` Delete all backups except `max_backups`. See `prune_backups()`

`push_backup()` <BackupQueueIndex> Create a new backup with index 1, push back all other indices. Always calls `$prune()` before it terminates.

`push_backup(overwrite = FALSE, now = Sys.time())` <BackupQueueDate> <BackupQueueDateteime> Create a new backup with a timestamp. The `now` parameter override the real system time. If `overwrite` is `TRUE` existing backups with the same filename (i.e timestamp) are overwritten. Always calls `$prune()` before it terminates.

`backup_dir`, `set_backup_dir(x)` `character` scalar. Set a directory in which to place the backups

`cache_backups`, `set_cache_backups(x)` `TRUE` or `FALSE`. If `TRUE` (the default) the list of backups is cached, if `FALSE` it is read from disk every time this appender triggers. Caching brings a significant speedup for checking whether to rotate or not based on the `age` of the last backup, but is only safe if there are no other programs/functions (except this appender) interacting with the backups.

`compression`, `set_compression` See `compression` argument of `rotate()`

`file`, `set_file(x)` `character` scalar. The file to backup/rotate

`fmt`, `set_fmt(x)` `character` scalar. See `format` argument of `rotate_date()`

`max_backups`, `set_max_backups(x)` See `max_backups` argument of `rotate()`

`should_rotate(size)` <BackupQueueIndex> Should a file of `size` be rotated? See `size` argument of `rotate()`

`should_rotate(size, age, now = Sys.time(), last_rotation = self$last_rotation)` <BackupQueueDate> <B Should a file of `size` and `age` be rotated? See `size` and `age` arguments of `rotate_date()`. `now` overrides the current system time, 'last_rotation" overrides the date of the last rotation.

`update_backups_cache()` Force update of the backups cache. Only does something if `$cache_backups` is `TRUE`.

Usage

```
x <- BackupQueueIndex$new(file, backup_dir = dirname(file), max_backups = Inf,
  compression = FALSE)

x <- BackupQueueDate$new(file, backup_dir = dirname(file), max_backups = Inf,
  compression = FALSE, fmt = "%Y-%m-%d", cache_backups = FALSE)

x <- BackupQueueDateTime$new(file, backup_dir = dirname(file), max_backups =
  Inf, compression = FALSE, fmt = "%Y-%m-%d--%H-%M-%S", cache_backups = FALSE)

x$increment_index(n = 1)
x$pad_index()
x$print()
x$prune(max_backups = self$max_backups)
x$push_backup()
x$push_backup(overwrite = FALSE, now = Sys.time())
x$set_backup_dir(x)
x$set_cache_backups(x)
x$set_compression(x)
x$set_file(x)
x$set_fmt(x)
x$set_max_backups(x)
x$should_rotate(size, age, now = Sys.time(), last_rotation = self$last_rotation %||% file.info(self$f
x$should_rotate(size, verbose = FALSE)
x$update_backups_cache()

x$backup_dir
x$backups
x$cache_backups
x$compression
x$file
x$fmt
x$has_backups
x$last_rotation
x$max_backups
x$n_backups
```

---

backup_info *Discover existing backups*

Description

These function return information on the backups of a file (if any exist)

Usage

```
backup_info(file, backup_dir = dirname(file))

list_backups(file, backup_dir = dirname(file))

n_backups(file, backup_dir = dirname(file))

newest_backup(file, backup_dir = dirname(file))

oldest_backup(file, backup_dir = dirname(file))
```

Arguments

`file`	`character` scalar: Path to a file.
`backup_dir`	`character` scalar. The directory in which the backups of `file` are stored (defaults to `dirname(file)`)

Value

`backup_info()` returns a `data.frame` similar to the output of `file.info()`

`list_backups()` returns the paths to all backups of `file`

`n_backups()` returns the number of backups of `file` as an `integer` scalar

`newest_backup()` and `oldest_backup()` return the paths to the newest or oldest backup of `file` (or an empty `character` vector if none exist)

Intervals

In **rotor**, an interval is a character string in the form `"<number> <interval>"`. The following intervals are possible: `"day(s)"`, `"week(s)"`, `"month(s)"`, `"quarter(s)"`, `"year(s)"`. The plural `"s"` is optional (so `"2 weeks"` and `"2 week"` are equivalent). Please be aware that weeks are ISOweeks and start on Monday (not Sunday as in some countries).

Interval strings can be used as arguments when backing up or rotating files, or for pruning backup queues (i.e. limiting the number of backups of a single) file.

When rotating/backing up `"1 months"` means "make a new backup if the last backup is from the preceding month". E.g if the last backup of `myfile` is from `2019-02-01` then `backup_time(myfile, age = "1 month")` will only create a backup if the current date is at least `2019-03-01`.

When pruning/limiting backup queues, `"1 year"` means "keep at least most one year worth of backups". So if you call `backup_time(myfile, max_backups = "1 year")` on `2019-03-01`, it will create a backup and then remove all backups of `myfile` before `2019-01-01`.

See Also

`rotate()`

Examples

```
# setup example files
tf <- tempfile("test", fileext = ".rds")
saveRDS(cars, tf)
backup(tf)
backup(tf)

backup_info(tf)
list_backups(tf)
n_backups(tf)
newest_backup(tf)
oldest_backup(tf)

# cleanup
prune_backups(tf, 0)
n_backups(tf)
file.remove(tf)
```

`rotate` *Rotate or backup files*

Description

Functions starting with `backup` create backups of a `file`, while functions starting with `rotate` do the same but also replace the original `file` with an empty one (this is useful for log rotation)

`prune_backups()` physically deletes all backups of a file based on `max_backups`

Usage

```
rotate(file, size = 1, max_backups = Inf, compression = FALSE,
  backup_dir = dirname(file), create_file = TRUE, dry_run = FALSE,
  verbose = dry_run)

backup(file, size = 0, max_backups = Inf, compression = FALSE,
  backup_dir = dirname(file), dry_run = FALSE, verbose = dry_run)

prune_backups(file, max_backups, backup_dir = dirname(file),
  dry_run = FALSE, verbose = dry_run)

rotate_date(file, age = 1, size = 1, max_backups = Inf,
  compression = FALSE, format = "%Y-%m-%d",
  backup_dir = dirname(file), overwrite = FALSE, create_file = TRUE,
  now = Sys.Date(), dry_run = FALSE, verbose = dry_run)

backup_date(file, age = 1, size = 1, max_backups = Inf,
  compression = FALSE, format = "%Y-%m-%d",
  backup_dir = dirname(file), overwrite = FALSE, now = Sys.Date(),
```

```
  dry_run = FALSE, verbose = dry_run)

rotate_time(file, age = -1, size = 1, max_backups = Inf,
  compression = FALSE, format = "%Y-%m-%d--%H-%M-%S",
  backup_dir = dirname(file), overwrite = FALSE, create_file = TRUE,
  now = Sys.time(), dry_run = FALSE, verbose = dry_run)

backup_time(file, age = -1, size = 1, max_backups = Inf,
  compression = FALSE, format = "%Y-%m-%d--%H-%M-%S",
  backup_dir = dirname(file), overwrite = FALSE, now = Sys.time(),
  dry_run = FALSE, verbose = dry_run)
```

Arguments

file	character scalar: file to backup/rotate
size	scalar integer, character or Inf. Backup/rotate only if file is larger than this size. Integers are interpreted as bytes. You can pass character vectors that contain a file size suffix like 1k (kilobytes), 3M (megabytes), 4G (gigabytes), 5T (terabytes). Instead of these short forms you can also be explicit and use the IEC suffixes KiB, MiB, GiB, TiB. In Both cases 1 kilobyte is 1024 bytes, 1 megabyte is 1024 kilobytes, etc... .
max_backups	maximum number of backups to keep • an integer scalar: Maximum number of backups to keep In addition for timestamped backups the following value are supported: • a Date scalar: Remove all backups before this date • a character scalar representing a Date in ISO format (e.g. "2019-12-31") • a character scalar representing an Interval in the form "<number> <interval>" (see below for more info)
compression	Whether or not backups should be compressed • FALSE for uncompressed backups, • TRUE for zip compression; uses zip() • a scalar integer between 1 and 9 to specify a compression level (requires the zip package, see its documentation for details) • the character scalars "utils::zip()" or "zip::zipr" to force a specific zip command
backup_dir	character scalar. The directory in which the backups of file are stored (defaults to dirname(file))
create_file	logical scalar. If TRUE create an empty file in place of file after rotating.
dry_run	logical scalar. If TRUE no changes are applied to the file system (no files are created or deleted)
verbose	logical scalar. If TRUE additional informative messages are printed
age	minimum age after which to backup/rotate a file; can be • a character scalar representing an Interval in the form "<number> <interval>" (e.g. "2 months", see *Intervals* section below).

- a Date or a character scalar representing a Date for a fixed point in time after which to backup/rotate. See format for which Date/Datetime formats are supported by rotor.

format — a scalar character that can be a subset of of valid strftime() formatting strings. The default setting is "%Y-%m-%d--%H-%M-%S".

- You can use an arbitrary number of dashes anywhere in the format, so "%Y-%m-%d--%H-%M-%S" and "%Y%m%d%H%M%S" are both legal.
- T and _ can also be used as separators. For example, the following datetime formats are also possible: %Y-%m-%d_%H-%M-%S (Python logging default), %Y%m%dT%H%M%S (ISO 8601)
- All datetime components except %Y are optional. If you leave out part of the timestamp, the first point in time in the period is assumed. For example (assuming the current year is 2019) %Y is identical to 2019-01-01--00-00-00.
- The timestamps must be lexically sortable, so "%Y-%m-%d" is legal, "%m-%d-%Y" and %Y-%d are not.

overwrite — logical scalar. If TRUE overwrite backups if a backup of the same name (usually due to timestamp collision) exists.

now — The current Date or time (POSIXct) as a scalar. You can pass a custom value here to to override the real system time. As a convenience you can also pass in character strings that follow the guidelines outlined above for format, but please note that these differ from the formats understood by as.POSIXct() or as.Date().

Value

file as a character scalar (invisibly)

Side Effects

backup(), backup_date(), and backup_time() may create files (if the specified conditions are met). They may also delete backups, based on max_backup.

rotate(), rotate_date() and rotate_time() do the same, but in addition delete the input file, or replace it with an empty file if create_file == TRUE (the default).

prune_backups() may delete files, depending on max_backups.

Intervals

In **rotor**, an interval is a character string in the form "<number> <interval>". The following intervals are possible: "day(s)", "week(s)", "month(s)", "quarter(s)", "year(s)". The plural "s" is optional (so "2 weeks" and "2 week" are equivalent). Please be aware that weeks are ISOweeks and start on Monday (not Sunday as in some countries).

Interval strings can be used as arguments when backing up or rotating files, or for pruning backup queues (i.e. limiting the number of backups of a single) file.

When rotating/backing up "1 months" means "make a new backup if the last backup is from the preceding month". E.g if the last backup of myfile is from 2019-02-01 then backup_time(myfile, age = "1 month") will only create a backup if the current date is at least 2019-03-01.

When pruning/limiting backup queues, `"1 year"` means "keep at least most one year worth of backups". So if you call `backup_time(myfile, max_backups = "1 year")` on `2019-03-01`, it will create a backup and then remove all backups of `myfile` before `2019-01-01`.

See Also

`list_backups()`

Examples

```
# setup example file
tf <- tempfile("test", fileext = ".rds")
saveRDS(cars, tf)

# create two backups of `tf``
backup(tf)
backup(tf)
list_backups(tf)  # find all backups of a file

# If `size` is set, a backup is only created if the target file is at least
# that big. This is more useful for log rotation than for backups.
backup(tf, size = "100 mb")  # no backup becuase `tf` is to small
list_backups(tf)

# If `dry_run` is TRUE, backup() only shows what would happen without
# actually creating or deleting files
backup(tf, size = "0.1kb", dry_run = TRUE)

# rotate() is the same as backup(), but replaces `tf`` with an empty file
rotate(tf)
list_backups(tf)
file.size(tf)
file.size(list_backups(tf))

# prune_backups() can remove old backups
prune_backups(tf, 1)  # keep only one backup
list_backups(tf)

# rotate/backup_date() adds a date instead of an index
# you should not mix index backups and timestamp backups
# so we clean up first
prune_backups(tf, 0)
saveRDS(cars, tf)

# backup_date() adds the date instead of an index to the filename
backup_date(tf)

# `age` sets the minimum age of the last backup before creating a new one.
# the example below creates no new backup since it's less than a week
# since the last.
backup_date(tf, age = "1 week")
```

```
# `now` overrides the current date.
backup_date(tf, age = "1 year", now = "2999-12-31")
list_backups(tf)

# backup_time() creates backups with a full timestamp
backup_time(tf)

# It's okay to mix backup_date() and backup_time()
list_backups(tf)

# cleanup
prune_backups(tf, 0)
file.remove(tf)
```

Index