NAME

rat — schedule self-mailing reminder

SYNOPSIS

rat [**-Rv**] [**-r** **!**|*interval*|*next*] [**-m** **!**|*max*|*last*] [**-a** **!**|*delay*|*when*] [**-A** *archive*] [[**%**]*when* [*event* [*reminders*] [*body*...]]]

rat [**-Rv**] [**-r** **!**|*interval*|*next*] [**-m** **!**|*max*|*last*] [**-a** **!**|*delay*|*when*] [**-A** *archive*] [*event* [[**%**]*when* [*reminders*] [*body*...]]]

DESCRIPTION

Asks the user for:

- when something happens,
- what the occasion is,
- when they want to be reminded about it,
- additional data they want to remember verbatim,

in that order, then schedules it for delivery via `ratrun`(8), which will conveniently mail them this information at decreasing intervals to the time of the event.

This data may be incrementally partially satisfied via the arguments, instead.

when is in **date -d** format. If you specify it in a less-committal "next thursday"- or "tomorrow 15:00"-style format, which would change on each scan, you may want to prefix it with a '**%**' to save it as a concrete time à la "2022-11-02T04:44+01:00" relative to now. The date and time of the event in a human-readable format are always listed on the standard output stream.

event is used as a file name (but slashes are massaged out to underscores, and, if required, −*when* and −**%***when* are appended to prevent clashes) and is part of the `Subject:` in reminder mails.

reminders are only used if they start with a digit, otherwise the argument is subsumed by the *body*. In addition to the normal **Reminders**, each field may be a clock time (any **date -d** format, but clock time pairs well with a clock *when*).

body arguments are pasted together with spaces.

Non-interactively (if the standard input stream is not a teletype), no input is read, so empty *reminders* and *body* are assumed, and *when* and *event* are required.

If any options are specified, they engage `rerat`(8) for either scheduling a periodic reminder (**-rm**) or archiving the event when it expires (**-aA**):

- **-r**, in **Reminders** format, specifies the time between instances of the event and
- **-m** limits how many times the event will fire overall, while
- **-a**, in **Reminders** format, is the time after the event date after which it should be archived, and
- **-A** sets the file to archive to under `~/.ratrun/old/`.
- **-v** sets **verbose** (you'll be mailed when the event is re-scheduled or archived), and
- **-R** just adds a blank "`rerat:`" line.

In all cases, **!** overrides to default — no re-scheduling, no limit, or no archiving, respectively.
-A is a `date`(1) format string applied to the event date; the default is "`.old`".

The numeric options are used verbatim, if in the correct format; otherwise:

- **-r** may also be a **date -d**-format date, which becomes the second occurrence of the event, from which the period is derived; if
- **-m** is a **date -d**-format date, it's the final ocurrence of the event, with the repeat count rounded up (such that, in most cases, specifying just the day is sufficient to get what you want); if
- **-a** is a **date -d**-format date, it's the exact time to archive the event.

In addition to the date and time of the event, the recurrence and archival configuration is listed, if `rerat`(8) will process this file after it expires — this also takes into account your current default configuration: even if you didn't specify any options, if **all** is set, the event will still be subjected to processing, for example.

Reminders

Start with a digit, are decimal integers, and optionally end with a recursively-expanded multiplicative suffix:

mo = 4**wk**

wk = 7**d**
yr = 365**d**
d = 24**h**
h = 60**m**
m = 60
s = 1

in this order, i.e. 10**h** = 600**m** = 36000 (= 36000**s**).

They correspond to the minimal time before the event, in seconds, to send a reminder — i.e. for reminders "3600 60 0" (equiv. "1**h** 1**m** 0"), a mail will be sent no sooner than an hour, a minute, and at the time of the event.

SIGNALS

`SIGHUP`, `SIGINT`, `SIGQUIT`, and `SIGTERM` are caught, and the in-progress event discarded.

ENVIRONMENT

`RATRUN_REMINDERS` System-wide default list of reminders. Overriden per-user with `.reminders` and per-event with *`reminders`*.

`RERAT_DEFAULT` The default set of configuration words. Overriden per-user with `.rerat` and per-event with "`rerat:`" lines.

`CONFDIR` Replaces "`/etc/default`" below.

FILES

`/etc/default/ratrun` Sourced at the top.

`~/.ratrun/` Scheduled events go here.

`~/.ratrun/old/` And the ones for which all reminders were sent end up here.

`~/.ratrun/.reminders` Overrides `RATRUN_REMINDERS`, if present. One field-split line.

`~/.ratrun/.nevermind` Never ask for per-event reminders if present.

`~/.ratrun/.tz` If present, value exported as `TZ` at the top, cf. `tzset`(3). One line.

`~/.ratrun/.prefix` Overrides the "`ratrun:`" subject prefix, if present. "☞" and empty are good candidates. One line.

`~/.ratrun/.rerat` Overrides `RERAT_DEFAULT`, if present, providing a default configuration for all processed events. One field-split line.

EXAMPLES

```
$ rat 12:00 call-robert 1h\ 0 bring up sales for q4
Happens on Mon 31 Oct 2022 12:00:00 CET
```

or

```
$ rat 12:00 call-robert 11:00\ 12:00 bring up sales for q4
rat: reminders normalised to 1h 0
Happens on Mon 31 Oct 2022 12:00:00 CET
```

```
$ rat 18:30 'meet henry in 201' ''
Happens on Mon 31 Oct 2022 18:30:00 CET
```

or

```
$ rat
Event date/time (start with % to canonicalise): 18:30
Happens on Mon 31 Oct 2022 18:30:00 CET
Event: meet henry in 201
Reminders (clock times OK; empty = default (%s)):
Enter body; ^D to finish
^D
```

```
$ rat 2022-12-12T23:11 1yr-uptime "$(date; uptime)"
Happens on Mon 12 Dec 2022 23:11:00 CET
```

Assuming a default reminder time of "30**m**", the first reminders from each of these will, respectively, produce the following messages:

```
Date: Mon, 31 Oct 2022 11:00:42 +0100
Subject: ratrun: in 1h / on 12:00: call-robert
```

```
    bring up sales for q4
```
and
```
    Date: Mon, 31 Oct 2022 18:00:21 +0100
    Subject: ratrun: in 30m / on 18:30: meet henry in 201
```
and
```
    Date: Mon, 12 Dec 2022 22:41:12 +0100
    Subject: ratrun: in 30m / on 23:11 (2022-12-12T23:11): 1yr-uptime

    Mon 31 Oct 21:00:54 CET 2022
     21:00:54 up 322 days, 12:09,  4 users,  load average: 1.28, 0.69, 0.52
```

To archive the call with Robert after a month to the default (configured) archive file:
```
    $ rat -a 1mo 12:00 call-robert 1h\ 0 bring up sales for q4
    Happens on Mon 31 Oct 2022 12:00:00 CET
    Will be archived in ~/.ratrun/old/.old on Mon 28 Nov 2022 12:00:00 CET
```
or, as the case may be with **archive-name**=*.'%y-Q%q* in `~/.ratrun/.rerat`,
```
    Will be archived in ~/.ratrun/old/.'22-Q4 on Mon 28 Nov 2022 12:00:00 CET
```

To instead make the meeting with Henry weekly:
```
    $ rat -r 1wk 18:30 'meet henry in 201' ''
    Happens on Mon 31 Oct 2022 18:30:00 CET
    Will be re-scheduled ad infinitum;
    next time: Mon 07 Nov 2022 18:30:00 CET
```
(in both cases, the options can be added to both invocations).

Schedule a daily reminder for two weeks, starting tomorrow at 9pm:
```
    $ rat -r 1d -m 14 '%tomorrow 21:00' chrzęść ''
    Happens on Sun 08 Jan 2023 21:00:00 CET
    Will be re-scheduled 13 times; next time: Mon 09 Jan 2023 21:00:00 CET;
                                   last time: Sat 21 Jan 2023 21:00:00 CET
```
or, equivalently:
```
    $ rat -r '2023-01-09 21:00' -m 2023-01-21 '2023-01-08 21:00' chrzęść ''
```
&c.

SEE ALSO

`date`(1), `ratrun.ics`(7), `ratrun`(8), `rerat`(8)

It is quite easy to schedule events "by hand", too: see `ratrun`(8), **EXAMPLES**.
It is also quite easy to manually make events periodic, or vice versa: see `rerat`(8), **EXAMPLES**.
iCalendar/VCALENDAR/`.ics` files can be dropped into `~/.ratrun/` and will work just as well.

NAME

ratrun.ics — iCalendar emulation for ratrun

SYNOPSIS

```
$ head -n1 ~/.ratrun/an-event[.ics]
BEGIN:VCALENDAR
```

DESCRIPTION

Internet Calendaring and Scheduling Core Object Specification (iCalendar), also commonly known by its prescribed extension `.ics` and much less commonly as VCALENDAR, is ubiquitously used for that purpose, and primarily distributed as **text/calendar** parts.

To that end, files in this format may be freely used as `ratrun`(8) events in `~/.ratrun/`.

iCal is complicated and **ratrun** is not, so support is only emulated, by using

`DTSTART`	as the event time (in the proper `TZID` time-zone),
`SUMMARY`	in the `Subject:` header,
`DESCRIPTION`	if any, as the notification mail body, and
`TRIGGER`s	if any, to override the default reminders.

Naturally, there's more to calendar entries, so a full path to the iCal file is appended as the mail footer.

FILES

`/usr/share/zoneinfo/ TZID`s are validated to exist here before being used.

SEE ALSO

`rat`(1), `rerat.ics`(7) – its `X-RATSTART` keys are also understood, `ratrun`(8)

STANDARDS

RFC5545: https://www.rfc-editor.org/rfc/rfc5545

Unrelatedly, this emulator is expected to be compatible with most common commercial calendar front-ends and probably all sane ones; it was tested against Microsoft Outlook (2019), Microsoft Teams, aCalendar, Evolution (GNOME™), and Google Calendar™, with largely favourable results.

BUGS

Besides, of course, not *actually* parsing iCal, in no particular order:

- only single-event files produce favourable results; §3.6. Calendar Components, final para. states: "a complex iCalendar object that is used to capture a complete snapshot of the contents of a calendar is possible. More commonly, an iCalendar object will consist of just a single [...] calendar component.", and most CUAs won't even let you export more than one at a time (the one multi-event object in the test corpus (so far!) comes from a ticket distribution system, and nothing is lost there),
- recurring of events (`RRULE`) isn't handled at all,
- positive and absolute `VALARM TRIGGER`s are ignored, all are assumed to be relative to `DTSTART`,
- all other `VALARM` fields are ignored (this is a feature actually),
- iCal has an opulent system for specifying time-zones. We consider `X-WR-TIMEZONE` equivalent,
- §3.2.19. Time Zone Identifier, Note kindly recommends "Implementers may want to use the naming conventions defined in existing time zone specifications such as the public-domain TZ database", and, indeed, non-Microsoft products largely do, if they don't just spec a UTC time. To support the rest, with decent hit-chance, these are tried, in order, with the first extant used:
  1. the `TZID` itself,
  2. the same with "Standard" removed, or
  3. 2. or 1., but only the upper-case characters (reduced to acronym);
- `printf`(1) `%b` is used to expand \ escapes.

NAME

rerat.ics — (lack of) iCalendar emulation for rerat

SYNOPSIS

```
$ head -n1 ~/.ratrun/old/an-event[.ics]
BEGIN:VCALENDAR
```

DESCRIPTION

Contrary to ratrun.ics(7), iCal RRULE is semantically too complex to be handled correctly in all common cases. As such, no attempt to do so is made.

Since they don't interfere with the event format, normal "rerat:" specs can be appended to the event, and work as expected.

To that end, an X-RATSTART key is appended to the event each time it gets re-scheduled. This functions to both override the original DTSTART, as well as to provide an equivalent for **reratted=**.

The event remains importable and, from the perspective of the iCal, unchanged.

FILES

/usr/share/zoneinfo/ TZIDs are validated to exist here before being used.

SEE ALSO

rat(1), ratrun.ics(7), ratrun(8), rerat(8)

STANDARDS

RFC5545: https://www.rfc-editor.org/rfc/rfc5545

The same emulator is used between **rerat** and **ratrun**, so the same compatibility is achieved for bulk event parsing.

BUGS

See ratrun.ics(7).

NAME

ratrun — scheduled reminder poster

SYNOPSIS

/usr/libexec/ratrun [**-n**[**n**]] [**-d** *date*]
/usr/libexec/ratrun -a

DESCRIPTION

Sends reminders for **Events** in `~/.ratrun/`, to the invoking user, then moves expired events to `old`. With **-n**, doesn't, and lists what *would* be sent to the standard output stream (incl. the bodies if **-nn**). **-d** overrides when "now" is: this is most useful in consort with **-n** to ascertain upcoming events.

With **-a**, runs **rat** for all configured users: those limited to `RATRUN_GROUPS`, plus the individually-named `RATRUN_USERS`, who have a `.ratrun` directory in their home directory. Users are mailed with a summary of the errors for their run, if any.

No user-specified code is ever run, and no root mail is generated (unless, of course, root schedules a reminder): **rat** is essentially just a way for users to schedule (periodic) mail delivery to themselves.

Events

Have a very simple format:

- the first line is the event date in **date -d** format,
- the second line is the per-event reminder override (optional, and taken as part of the body if it doesn't start with a digit),
- the remainder of the file is the event body, sent verbatim in the reminder.

An event is said to be expired if it doesn't have any more **Reminders** left.

iCalendar/VCALENDAR/`.ics` files can be dropped in verbatim, too, cf. `ratrun.ics`(7).

Reminders

Start with a digit, are decimal integers, and optionally end with a recursively-expanded multiplicative suffix:

mo	=	4**wk**
wk	=	7**d**
yr	=	365**d**
d	=	24**h**
h	=	60**m**
m	=	60
s	=	1

in this order, i.e. 10**h** = 600**m** = 36000 (= 36000**s**).

They correspond to the minimal time before the event, in seconds, to send a reminder — i.e. for reminders "3600 60 0" (equiv. "1**h** 1**m** 0"), a mail will be sent no sooner than an hour, a minute, and at the time of the event.

Notification format

For each reminder that expires, users will receive mail with a subject of

`ratrun: in {`*`latest-expired reminder`*`} / at {`*`HH:MM`*`} ({`*`raw event date`*`}):`
`{`*`event filename`*`}`

and body of the rest of the event. The reminder time is folded, in reverse, per the suffix table above; the parenthetical is omitted if it's the same as *HH:MM* (or *H:MM*), and the reminder – if the only one is **0**.

Additionally, users may receive messages with a subject of

`Errors for your ratrun at ...`

from root. These are produced by **-a** runs, and contain the standard error and output streams from the user's run.

ENVIRONMENT

`RATRUN_REMINDERS` System-wide default list of reminders. Overriden per-user with `.reminders` and per-event with the second line.

`RATRUN_GROUPS` If non-empty, in **-a** mode, only check `.ratrun` presence for the specified groups. Field-split (white-space-delimited).

RATRUN_USERS	If non-empty, in **-a** mode, only check .ratrun presence for the specified users, or add them to the result from RATRUN_GROUPS. Field-split.
CONFDIR	Replaces "/etc/default" below.

FILES

/etc/default/ratrun	Sourced at the top.
~/.ratrun/	Contains **Events**. Directory/hidden/unreadable/empty files are ignored.
~/.ratrun/old/	Expired events are moved here. If one already existed, its new name is appended with its date (first raw, then à la 2022-11-02T04:44+01:00).
~/.ratrun/.reminders	Overrides user's RATRUN_REMINDERS, if present. One field-split line.
~/.ratrun/.tz	If present, value exported as TZ per-user (cf. tzset(3)). One line.
~/.ratrun/.prefix	Changes the user's mail subject prefix from "ratrun:"; "☞" and empty are popular choices. One line.
~/.ratrun/.expcnt/	Contains counts for expired reminders for each event. Remove the file corresponding to an event from this directory to re-send its latest reminder. Entries here with no corresponding events are auto-pruned.

EXAMPLES

```
$ cat > .ratrun/call-robert
12:00
1h 0
bring up sales for q4
^D
$ echo 18:30 > .ratrun/'meet henry in 201'
$ { echo 2022-12-12T23:11; date; uptime; } > .ratrun/1yr-uptime
```

Assuming a default reminder time of "30**m**", the first reminders from each of these will, respectively, produce the following messages:

```
Date: Mon, 31 Oct 2022 11:00:42 +0100
Subject: ratrun: in 1h / on 12:00: call-robert

bring up sales for q4
```

and

```
Date: Mon, 31 Oct 2022 18:00:21 +0100
Subject: ratrun: in 30m / on 18:30: meet henry in 201
```

and

```
Date: Mon, 12 Dec 2022 22:41:12 +0100
Subject: ratrun: in 30m / on 23:11 (2022-12-12T23:11): 1yr-uptime

Mon 31 Oct 21:00:54 CET 2022
 21:00:54 up 322 days, 12:09,  4 users,  load average: 1.28, 0.69, 0.52
```

SEE ALSO

date(1), mail(1), rat(1) – interactive scheduling, ratrun.ics(7) – iCalendar emulation, rerat(8) – event periodisation and archiving

It is safe to run multiple instances of **rat** for any given user at any given time, for example via system and per-user crontab(5)s.

☞

NAME

rerat — reminder rescheduler and archiver

SYNOPSIS

/usr/libexec/rerat [**-n**[**n**]] [**-d** *date*]
/usr/libexec/rerat -a

DESCRIPTION

Re-schedules expired `ratrun`(8) **Events** from `~/.ratrun/old/` back to `~/.ratrun/` or archives them if they're old enough. With **-n**, doesn't, and lists what *would* be moved to the standard output stream (incl. the precise edits if **-nn**). **-d** overrides when "now" is.

With **-a**, runs **rat** for all configured users: those limited to `RATRUN_GROUPS`, plus the individually-named `RATRUN_USERS`, who have a `.ratrun/old` directory in their home directory. Users are mailed with a summary of the errors (always) and motions (if **verbose**) for their run, if any.

Events

Are scanned for lines in the form (whitespace around the colon optional):

`rerat : [`*word*`...]`

from which field-split (white-space-delimited) configuration words are taken:

!|*interval*	re-schedule the event for the event date + *interval*; **!** – don't re-schedule;
max=!|*max*	re-schedule the event no more than *max* times; **!** – no limit (default);
reratted=*count*	counter maintained by **rat**, *count* is incremented every time the event is re-scheduled;
original-time=...	the original event's date line (this skips the rest of the "`rerat:`" line);
archive=!|*delay*	move the event to *arch* after the event date + *delay*; **!** – don't archive;
archive-name=*arch*	the file to save this event to, relative to `~/.ratrun/old/`; *arch* is formatted via `date`(1) with the event date; the default is "`.old`";
[**!**]**verbose**	note the re-scheduling and archival of this event to the standard output stream (sent to the user by mail in **-a** mode); **!** – be quiet (default);
[**!**]**all**	process all events (this is useful to enforce an archival policy for all events, for example); **!** – just those with "`rerat:`" branding (default).

interval and *delay* use the **Reminders** format, recounted below.

Re-scheduled events, naturally, aren't archived.

Reminders

Start with a digit, are decimal integers, and optionally end with a recursively-expanded multiplicative suffix:

mo = 4**wk**
wk = 7**d**
yr = 365**d**
d = 24**h**
h = 60**m**
m = 60
s = 1

in this order, i.e. 10**h** = 600**m** = 36000 (= 36000**s**).

They correspond to the minimal time before the event, in seconds, to send a reminder — i.e. for reminders "3600 60 0" (equiv. "1**h** 1**m** 0"), a mail will be sent no sooner than an hour, a minute, and at the time of the event.

ENVIRONMENT

`RERAT_DEFAULT` The default set of configuration words. Overriden per-user with `.rerat` and per-event with "`rerat:`" lines. [**!**]**all** is only meaningful when set here.

`RATRUN_GROUPS` If non-empty, in **-a** mode, only check `.ratrun` presence for the specified groups. Field-split (white-space-delimited).

`RATRUN_USERS` If non-empty, in **-a** mode, only check `.ratrun` presence for the specified users, or add them to the result from `RATRUN_GROUPS`. Field-split.

`CONFDIR` Replaces "`/etc/default`" below.

FILES

`/etc/default/ratrun` Sourced at the top.

`~/.ratrun/old/` Scanned for expired **Events** produced by **ratrun**. Directory/hidden/unreadable/empty files are ignored.

`~/.ratrun/` Re-scheduled events are moved here. If one already existed, its new name is appended with its date (à la 2022-11-02T04:44+01:00).

`~/.ratrun/.rerat` Overrides `RERAT_DEFAULT` per-user, if present, providing a default configuration for all processed events. One field-split line.

`~/.ratrun/.tz` If present, value exported as `TZ` per-user (cf. `tzset`(3)). One line.

EXAMPLES

Make the meeting with Henry weekly:

```
$ echo rerat :7d >> .ratrun/old/'meet henry in 201'
```

Schedule a daily reminder for two weeks, starting tomorrow at 9pm:

```
$ printf '%s\n' '2023-1-8 21:00' 'rerat: 1d max=13' > .ratrun/chrzęść
```

but abort it after a week:

```
$ echo rerat : ! >> .ratrun/chrzęść
$ tail -n2 .ratrun/chrzęść
rerat: 1d max=13 reratted=7 original-time=2023-1-8 21:00
rerat : !
```

Start archiving all events to "**.**{*year*}**-Q**{*quarter*}" (`~/.ratrun/old/.2022-Q4`, for example) after a quarter, except Robert-related ones; archive Henry-related ones to ("`.old`") after a year:

```
$ tee -a .ratrun/*obert* .ratrun/old/*obert*
rerat: archive=!
^D
$ tee -a .ratrun/*enry* .ratrun/old/*enry*
rerat:archive=1yr archive-name=.old
^D
$ echo all archive=3mo archive-name=.%Y-Q%q > .ratrun/.rerat
```

(without the **all** word, only the events *already containing* a "`rerat:`" line would be subject to archival).

SEE ALSO

`date`(1), `mail`(1), `rat`(1) – interactive scheduling, `rerat.ics`(7) – (lack of) iCalendar emulation, `ratrun`(8) – reminding about events

It is safe to run multiple instances of **rat** for any given user at any given time, for example via system and per-user `crontab`(5)s.

BUGS

Without **all** (or if archiving with **pax**/**cpio**; this is not the default), events in files with new-lines cannot be handled correctly. They aren't meaningful anyway, since mail subjects are single-line.