

## Scheduling

Scheduling in databases means setting up automated tasks (called *jobs*) that run at specific times or intervals — instead of you having to run them manually.

### Main Reasons for Scheduling Jobs

1. Resource Management
  - Goal: Prevent heavy operations from slowing down systems during busy hours.
  - Example: Run *intensive updates* or *data model training* at night or early morning when few users are online.
    - ◆ *Analogy:* Like running big downloads at night so your internet isn't slow during the day.
2. Up-to-Date Reports and Visualizations
  - Goal: Keep dashboards, BI tools, and analytics current.
  - Example: Automatically refresh data every morning so managers always see the latest figures.
    - ◆ *Think of it as:* Scheduling daily refreshes for Power BI or Tableau dashboards.
3. Auditing or Compliance Requirements
  - Goal: Meet mandatory reporting schedules required by regulations.
  - Example: Pharmaceutical firms may need *monthly audit reports* generated automatically.
    - ◆ *Key point:* Automation ensures reports are always produced on time, with a traceable schedule.
4. Automated Backups
  - Goal: Protect data without human intervention.
  - Example: Schedule nightly database backups to prevent data loss.
    - ◆ *Why it matters:* Manual backups are easy to forget — automation makes it reliable.
5. Avoid Manual Updating
  - Goal: Save time and reduce human error.
  - Example: Automate fetching new data, training models, and updating reports daily.
    - ◆ *Benefit:* More consistent and less error-prone than manual routines.

## Using the cronR Package

### Purpose:

cronR lets you automate the execution of R scripts — for example, to refresh data, run analyses, or generate reports at regular intervals (daily, hourly, weekly, etc.).

### 1. What is cronR?

- It's an R package that provides an easy interface to the cron scheduler (a standard job scheduler on Linux/Unix systems).
- Instead of writing raw cron commands (which can be tricky), cronR allows you to set up, edit, and manage cron jobs directly from R or RStudio.

### 2. Relation to Cron

- Cron = a system process that runs commands or scripts at scheduled times.
- cronR = a user-friendly R wrapper that lets you communicate with that system scheduler.

### 3. Before Using cronR: Start the Cron Service

- If you're working in a container (like a cloud RStudio or a Linux droplet), you need to make sure cron is running.

◆ In your RStudio terminal, run:

```
bash  
sudo cron start
```

 Code kopieren

This starts the cron service inside your container.

### 4. How to Use cronR

- You have two ways:
- Option A — *Visual Interface (Easiest)*
  - Go to the “Addins” menu in RStudio.
  - Choose “Schedule R scripts...” (from cronR).
  - Select your R script and set the time/frequency (e.g., every morning at 6:00 a.m.).
  - Save it — your job is now scheduled!
- Option B — *Command Line (Programmatic)*
  - You can use commands in R like:

```
R  
  
library(cronR)  
  
cmd <- cron_rscript("path/to/your_script.R")  
cron_add(cmd, frequency = "daily", at = "06:00", id = "daily_script")
```

 Code kopieren

This schedules your script to run daily at 6 a.m.

### 5. Resource Management Tips

- If you're using a small droplet (1GB RAM), watch your resource icon in RStudio (top-right corner).
- When resources get tight:
  - Clear objects from your workspace.
  - Restart RStudio to free memory.
- This ensures your scheduled jobs won't crash due to low memory.

## Create first Cron job

Part 1 – The R logic (the work you want to automate)

- To show how the cronR package works, we will use the following R script, which we will put on different schedules:

```
# Get the current datetime
date_time <- format(Sys.time(), digits = 0)
# Check if "increment_one.rds" exists
if(file.exists("/home/rstudio/scheduling/increment_one.rds")){
  # If "increment_one.rds" exists then we read it into memory
  increment_one <- readRDS(file = "/home/rstudio/scheduling/increment_one.rds")
  # We add one to the R object
  increment_one <- increment_one + 1
  # The R object is saved to the disk
  saveRDS(increment_one, file = "/home/rstudio/scheduling/increment_one.rds")
  # We print the datetime and the value of increment_one.
  # This will be captured by the cronR Logger and written to the .log file
  print(paste0(date_time, ": Value of increment_one.rds is ", increment_one))
}else{
  # If "increment_one.rds" does not exist we begin by 1
  increment_one <- 1
  # The R object is saved to the disk
  saveRDS(increment_one, file = "/home/rstudio/scheduling/increment_one.rds")
  # We print the datetime and the value of increment_one.
  # This will be captured by the cronR Logger and written to the .log file
  print(paste0(date_time, ": Value of increment_one.rds is ", increment_one))
}
```

Part 2 – The scheduling setup (the actual cron job command)

Now, the actual cron job is defined by these lines:

```
cmd <- cron_rscript(rscript="/home/rstudio/scheduling/Scheduling/increment_one.rds")
30 cmd <- cron_rscript(rscript="/home/rstudio/scheduling/Scheduling/increment_one.rds")
```

- This tells cronR *what script* to run (the .R file that contains the code above).
- It creates a command that cron understands (a wrapper to run your R script non-interactively).
- We give the name of the script we wish to execute on a schedule (we include the script path if our script is not located directly in our R working directory). The “cron\_rscript” function, constructs a command we can use in the “cron\_add” function. The “cron\_rscript” includes information of, e.g., where a log file will be stored and whether output from the R script should be stored in the log file.

By default, the log file will have the same name as the R script but ending in “.log”

```
cron_add(cmd, frequency = 'minutely', id='job1', description='Our first cronR job')
31
32 cron_add(cmd, frequency = 'minutely', id='job1', description='Our first cronR job')
```

Select y

```
> cron_add(cmd, frequency = 'minutely', id='job1', description='Our first cronR job')
Are you sure you want to add the specified cron job: '/usr/local/lib/R/bin/Rscript '/home/rstudio/scheduling/Scheduling/increment_on
e.rds' >> '/home/rstudio/scheduling/Scheduling/increment_one.log' 2>&1? [y/n]:
1: y
Adding cronjob:
-----
## cronR job
## id: job1
## tags:
## desc: Our first cronR job
0-59 * * * * /usr/local/lib/R/bin/Rscript '/home/rstudio/scheduling/Scheduling/increment_one.rds' >> '/home/rstudio/scheduling/Sche
duling/increment_one.log' 2>&1
```

- This is the line that creates the cron job.
- frequency = 'minutely' → tells cron to run it every minute.
- id = 'job1' → gives your job a name for tracking.
- description → adds a short explanation.

## Inspecting cronR jobs

Cron ls()

Calling this function, lists all cronR jobs in the console. We could have added further arguments as “id” or “tag” to only show a subset of the conR jobs.

```

19 print(paste0(date_time, " : Value of increment_one.rds is ", increment_one))
20 job1
21 # If "increment_one.rds" does not exist we begin by 1
22 increment_one <- 1
23 # The R object is saved to the disk
24 save(increment_one, file = "/home/rstudio/scheduling/Scheduling/increment_one.rds")
25 # We print the datetime and the value of increment_one
26 print(paste0(date_time, " : Value of increment_one.rds is ", increment_one))
27 }
28
29 cmd <- cron_rscript(rscript = "/home/rstudio/scheduling/Scheduling/increment_one.rds")
30 cron_add(cmd, frequency = 'minutely', id = 'job1', description = 'Our first cronR job')
31
32 cron_ls()
33
34 cron_is()
35
36
37 (Top Level)

```

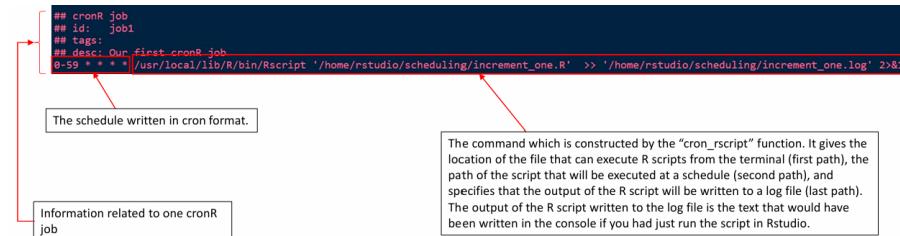
Console: cron\_ls()

```

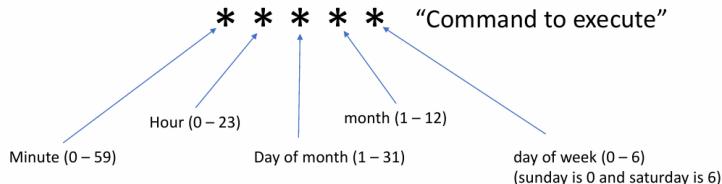
## R 4.5.1 --> /scheduling/Scheduling/
## id: job1
## tags:
## desc: Our first cronR job
0-59 * * * * /usr/local/lib/R/bin/Rscript '/home/rstudio/scheduling/increment_one.R' >> '/home/rstudio/scheduling/Scheduling/increment_one.log' 2>&1

```

### Outcome



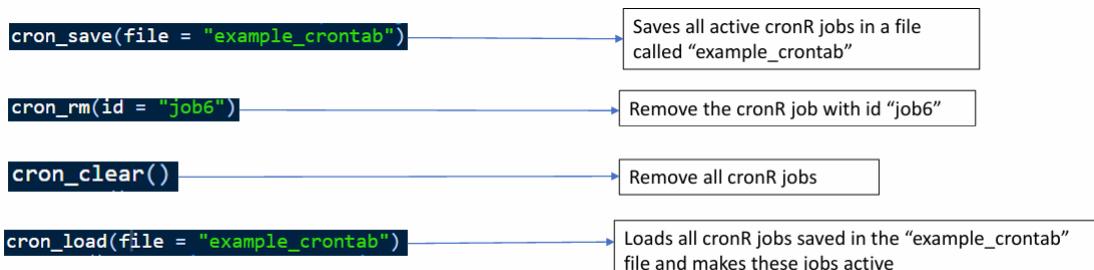
## CronR schedule



More details [Understanding Cron Syntax in the Job Scheduler - Cloud Manager Administrator Reference Examples](#)



## Saving/loading Crontab and removing cronR jobs



## Using the visual cronR interface

The visual cronR interface allows you to schedule R scripts using the cronR package. It provides a graphical user interface for selecting the script, setting launch parameters, and defining the execution schedule.

**Top Panel:** Shows the RStudio environment with the script `increment\_one.R` open. The `Addins` menu is open, showing options like `Schedule R scripts on Linux/Unix`.

**Middle Panel:** A modal dialog titled "Cron job scheduler" is displayed. It contains fields for "Selected Rscript" (No R script selected yet), "Job description" (I execute things), "Launch date" (2025-10-29), "Launch hour" (10:05), "Additional arguments to Rscript" (empty), "Job identifier" (job\_2e5d3b1945602e9aab08fb4c0d7ff7fb), and "Schedule" (ONCE). Buttons for "Create job" and "Manage existing jobs" are at the bottom.

**Bottom Panel:** The RStudio environment shows the scheduled job in the Global Environment pane. The file `increment\_one.rds` has been created and its contents are displayed.

**Annotations:**

- A red box highlights the "Selected Rscript" field in the "Cron job scheduler" dialog.
- A red box highlights the "Schedule" section in the "Cron job scheduler" dialog.
- A red box highlights the "Manage existing jobs" button in the "Cron job scheduler" dialog.
- A callout box labeled "The script we wish to add to our cronR schedule" points to the "Selected Rscript" field.
- A callout box labeled "How often should the script be run" points to the "Schedule" section.
- A callout box labeled "Inspect existing jobs" points to the "Manage existing jobs" button.

## Full example to run CronR

### To run CronR

#### First create a new image with the CronR image

To do that:

#### Create new folder in Filezilla where the file later will be saved

The screenshot shows the Filezilla interface with two panes. The left pane (Local site) shows a directory structure under 'C:\Users\dnok\'. The right pane (Remote site) shows a directory structure under '/'. A new folder named 'scheduling' is being created in the remote directory. The status bar at the bottom indicates '8 files and 23 directories. Total size: 19,095,572 bytes'.

#### Move file in the newly created folder (from Local to remote)

The screenshot shows the Filezilla interface with two panes. The left pane (Local site) shows a directory structure under 'C:\Users\dnok\Downloads'. The right pane (Remote site) shows a directory structure under '/scheduling'. A file named 'dockerfile\_scheduling' is selected in the local Downloads folder and is being transferred to the remote '/scheduling' folder. A red arrow points from the local file to the remote destination. The status bar at the bottom indicates 'Selected 1 file. Total size: 736 bytes' and 'Transfers finished All files have been successfully transferred'.

#### In Digital Ocean go into the folder we just created and saved the file

pwd

current working directory

cd ..

moves one level up

ls -l

information about files and directories in the current directory

cd scheduling^

in this case to get into the scheduling folder

```
root@ubuntu-s-1vcpu-1gb-fral-01:/# cd scheduling  
root@ubuntu-s-1vcpu-1gb-fral-01:/scheduling#
```

## Build the Image

```
docker image build --tag rstudio:1.0.3 -f dockerfile_scheduling .
```

```
Run 'docker buildx build --help' for more information  
root@ubuntu-s-1vcpu-1gb-fral-01:/scheduling# docker image build --tag rstudio:1.0.3 -f dockerfile_scheduling .  
[+] Building 81.0s (8/8) FINISHED  
=> [internal] load build definition from dockerfile_scheduling  
=> transferring dockerfile: 786B  
=> [internal] load metadata for docker.io/rocker/tidyverse:latest  
=> [internal] load .dockerrcignore  
=> [internal] transfer context: 2B  
[1/4] FROM docker.io/rocker/tidyverse:latest@sha256:b340dd5c2867463cb51380a490d2e4af5c7fe24b5def6c8b445fe08b46088043  
=> resolving docker.io/rocker/tidyverse:latest@sha256:b340dd5c2867463cb51380a490d2e4af5c7fe24b5def6c8b445fe08b46088043  
=> sha256:b340dd5c2867463cb51380a490d2e4af5c7fe24b5def6c8b445fe08b46088043 1.61kB / 1.61kB  
=> sha256:7b787d29323db800599067ec28533c7c38847899821a88802f42ad5d75 15.20kB / 15.20kB  
=> sha256:ae0cac034e9e3d9dc5ef12fbf1eb5be1d9c1cd87b13d33321lef6f5e73ac7 1.85kB / 1.85kB  
=> sha256:12beedb787d29323db800599067ec28533c7c38847899821a88802f42ad5d75 4.10kB / 4.10kB  
=> sha256:c41a1d575df5a52381c98411842ff66523d14f1805671145e7fdh446e60f4d 308.87MB / 308.87MB  
=> sha256:ala2l1c9ebc16121569dd937bcd1c745a5081629b3b08a64446602ded91e10a4 29.72MB / 29.72MB  
=> sha256:13bf3fc8a7c61ad1c82aae0c29b7dbef7d5531e1004f656cd01c007 2.83kB / 2.83kB  
=> sha256:alib2d2c6659e49507f78ebc08bla1c7rc45439428fa1f47236433b69a4146 1.50kB / 1.50kB  
=> sha256:a325d5976d1e2c660d71e8592d1efc9ac81777e2198b162aa0e0930c2154bf 28.49MB / 28.49MB  
=> extracting sha256:ala2l1c9ebc16121569dd937bcd1c745a5081629b3b08a64446602ded91e10a4  
=> sha256:42de0d7b5d73bda21f36d6e4765c2d4f7a11fdf52f03b103016c793aa09347 200.33MB / 200.33MB  
=> sha256:d4f619dc1c0ca75c0e0ad458e03e2099ded105dff0d3b89fc3e903ef5c5c98549 1.92kB / 1.92kB  
=> sha256:33a2fd53b3ee131a29dd21428f23c1bae4832406205709a2407417cb3620f50 750B / 750B  
=> sha256:a5aab7f2c8b20aa87b5a727c6740b8c9a5278b6c8bc01d7602f8994d463e145d 717B / 717B  
=> sha256:699f3e7f8e694039f06e0a863f1a72ad651e30c479a25ea45a7f4a8107b2f 524B / 524B  
=> sha256:39f051aca4afc783239a0f1c074a1379574b7bfld978aa2f5b4994579d0851 3.20kB / 3.20kB  
=> sha256:7367d8fc6ce1d4b70eedb19820c108fd541bcoa850c1932b8ec41e4dd53a0c 347B / 347B  
=> sha256:ed80e004b039420f99aa60a80ae0d0161b31449d878c29011ccbbaac3 331.69MB / 331.69MB  
=> sha256:1343c2783f405f7c276ca970074128a7b76ed40caff48e4468fa341145af66 1.28kB / 1.28kB  
=> sha256:40b07c7975aaacf24a5d597d0e05f208d1343453e471dcea63a5fc3e667454bce 34.45MB / 34.45MB  
=> sha256:47242d0f399527b438479ff6f6deedec6af522b0d85b16cc020b62a80b64535 1.08kB / 1.08kB  
=> sha256:285753150bc02ae68c8115358eb27c6ee2zc5b0f5549974acdca2c7650781e5 1.87MB / 1.87MB  
=> sha256:535014c5e51577f2a8ed83d49da1264f8deab54fb6963660940119d1c6e0af 13.51kB / 13.51kB  
=> extracting sha256:mee0cac034e8e03d9dc5ef12fbf1elb5b5e1d9c1cd87b136333721erf6f5e73ac7  
=> extracting sha256:c41a1d575dfaas52381c98411842ff8ea52d14f1805671145e7fdh446e60f4d  
=> extracting sha256:f3b13rc094f9b68e0c36ra1c7r8zae2c987dbef7d053181004f656cd02c07  
=> extracting sha256:a11b2d2c655e469507778e0bc8la1cf455439428fa1f47236433b69a4146  
=> extracting sha256:0a325d5976d1e2c660b716852d51e0c9a8c1777e2198b162aa0e0930c2154bf  
=> extracting sha256:bb1be7cafda745d0a4d30e226818a0529da8b910ed7c83e9ea0za50150a5256  
=> extracting sha256:73bd2a221f36d664765c2d4f7a117df52f283b303016c793aa09347  
=> extracting sha256:d4f62cd1c0ca75c0ead45e3e2099ded105dff0d3b89fc3e903ef5c5c598549  
=> extracting sha256:7367d8fc6ce1d4b70eedb19820a8e7b5a5f27e6740b8c5a278b6c8d17602f8994d463e145d  
=> extracting sha256:99f83e7f0e694039f06e0a85b63f1a72ad651e30c479a25ea45a7f4a8107b2f  
=> extracting sha256:33a2fd53b3ee131a29dd21428f23c1bae4832406205709a2407417cb3620f50  
=> extracting sha256:73e7bfcc6c1d4b70eedb3198e208fd541bcoa850c1932b8ec41e4dd53a0c  
=> extracting sha256:2ed80e04b039420f99aa60a80ae0901615b131449d9d0878c29011ccbbaac3  
=> extracting sha256:c5e1343c2783t405f7c276ca970074128a7b76ed40caff48e4468fa341145af66  
=> extracting sha256:47242d0f399527b438479ff6f6deedec6af522b0d85b16cc020b62a80b64535  
=> extracting sha256:39f051aca4afc783239a0f1c074a1379574b7bfld978aa2f5b4994579d0851
```

## build a new rstudio container using the image we just created:

```
docker run -d --network db_r_shiny -p 8787:8787 -e PASSWORD=xxxxxxxxxxxx --name rstudio -v  
rstudio_data:/home/rstudio rstudio:1.0.3
```

## Inspect the network

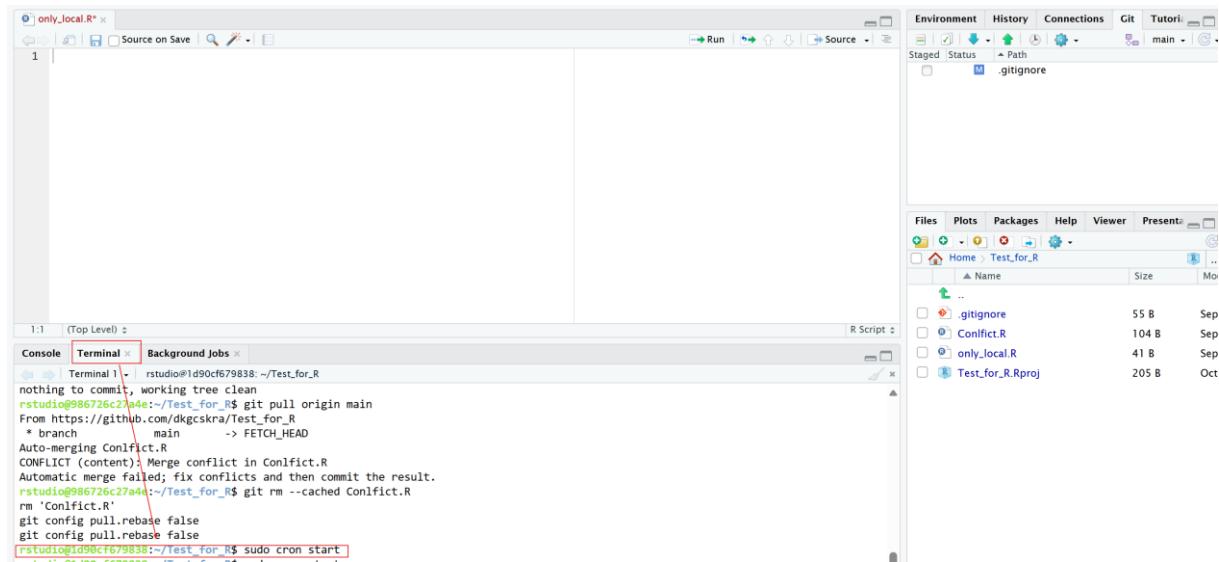
```
docker network inspect db_r_shiny
```

```

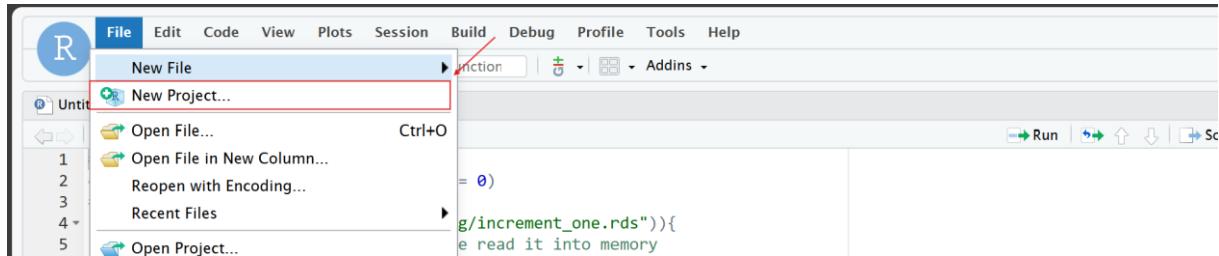
root@ubuntu-s-lvcpu-1gb-fral-01:/scheduling# docker network inspect db_r_shiny
[{"Name": "db_r_shiny", "Id": "72bf397433b7dad563e545a13ec866902d910799a362fdc3b68c12ba8192140", "Created": "2025-09-06T10:02:07.167438936Z", "Scope": "local", "Driver": "bridge", "EnableIPv4": true, "EnableIPv6": false, "IPAM": {"Driver": "default", "Options": {}, "Config": [{"Subnet": "10.0.42.0/24", "IPRange": "10.0.42.128/25"}]}, "Internal": false, "Attachable": true, "Ingress": false, "ConfigFrom": {"Network": ""}, "ConfigOnly": false, "Containers": {"16da7902dcde0f0510d3bb5cf47c81cled4ce093e07591571a45071ca068559c": {"Name": "postgres", "EndpointID": "3a75a7ba7837e13ab4517f91bbaab421b0c6413ee00f87f1dfbccaf2398baa7", "MacAddress": "46:8e:78:a9:c3:44", "IPv4Address": "10.0.42.130/24", "IPv6Address": ""}, "1d90cf679838c43fc03bceea4e240a6517af54493f3a7bd077b913ff400b0ea6": {"Name": "rstudio", "EndpointID": "3c6166f65df9a6cecd3b406b59a96d2069b36b53b432cc76e94003971ae517bc", "MacAddress": "fe:d5:24:49:a8:49", "IPv4Address": "10.0.42.129/24", "IPv6Address": ""}}, "Options": {}, "Labels": {}}

```

### Log into R and start Crone from the terminal



### Create new R Project



## Upload the scripts into R

The screenshot shows the RStudio interface. On the left, the 'Console' tab displays the R startup message and a natural language support note. The 'File Explorer' panel on the right shows a project structure under 'Home > scheduling > Scheduling'. A red box highlights the file 'increment\_one.R'. The 'Environment' panel at the top right shows an empty environment.

## Open the script

The screenshot shows the RStudio interface with the 'increment\_one.R' script open in the 'Code Editor'. The 'Console' tab at the bottom shows the command 'rscript increment\_one.R' being run. The 'Environment' panel on the right shows the variable 'increment\_\_' with a value of 2. A red arrow points from the 'increment\_one.R' file in the 'File Explorer' to the 'increment\_\_' entry in the 'Environment' panel.

## Define the cron job

```
cmd <- cron_rscript(rscript="/home/rstudio/scheduling/Scheduling/increment_one.rds")
```

```
30 cmd <- cron_rscript(rscript="/home/rstudio/scheduling/Scheduling/increment_one.rds")
```

## Add the cron job

```
cron_add(cmd, frequency = 'minutely', id='job1', description='Our first cronR job', days_of_week = 3)
```

```
31 cron_add(cmd, frequency = 'minutely', id='job1', description='Our first cronR job', days_of_week = 3)
```

Schedule the script "increment\_one.R" to execute every minute for this current weekday  
(Wednesday)

## Add another cron job

```
cron_add(cmd, frequency = 'daily', id='job2', description='Our second cronR job', at= '04:00',  
days_of_month = 15)
```

```
32 cron_add(cmd, frequency = 'daily', id='job2', description='Our second cronR job', at= '04:00', days_of_month = 15)
```

In this case Schedule the script "increment\_one.R" to execute at 04:00 but only on the 15th of each month.

## List the cron jobs

`cron_ls()`

```
> cron_ls()

## cronR job
## id: job1
## tags:
## desc: Our first cronR job
0-59 * * * 3 /usr/local/lib/R/bin/Rscript '/home/rstudio/scheduling/Scheduling/increment_one.rds' >> '/home/rstudio/scheduling/Scheduling/increment_one.log' 2>&1

## cronR job
## id: job2
## tags:
## desc: Our second cronR job
0 4 15 * * /usr/local/lib/R/bin/Rscript '/home/rstudio/scheduling/Scheduling/increment_one.rds' >> '/home/rstudio/scheduling/Scheduling/increment_one.log' 2>&1
```

## Save cron jobs to a file

`cron_save(file="my_BI_schedule")`

→ my\_BI\_schedule is the name of the file

The screenshot shows the RStudio interface with two panes. The left pane is the 'Console' showing the R code and its execution:

```
35 cron_save(file="my_BI_schedule")
36
37
38 cron_ls()
39 ## cronR job
40 ## id: job1
41 ## tags:
42 ## desc: Our first cron job
43 0-59 * * * 3 /usr/local/lib/R/bin/Rscript '/home/rstudio/scheduling/Scheduling/increment_one.rds' >> '/home/rstudio/scheduling/Scheduling/increment_one.log' 2>&1
44 ## cronR job
45 ## id: job2
46 ## tags:
47 ## desc: Our second cron job
48 0 4 15 * * /usr/local/lib/R/bin/Rscript '/home/rstudio/scheduling/Scheduling/increment_one.rds' >> '/home/rstudio/scheduling/Scheduling/increment_one.log' 2>&1
49 > cron_save(file="my_BI_schedule")
50
51 Saved crontab to file: /home/rstudio/scheduling/Scheduling/my_BI_schedule
52 [1] TRUE
53 >
```

The right pane is the 'File Browser' showing the contents of the directory:

- ignore (40 B, Oct)
- cron\_schedule\_examples.R (1.2 KB, Oct)
- increment\_one.log (19.8 KB, Oct)
- increment\_one.R (1.1 KB, Oct)
- increment\_one.rds (51 B, Oct)
- Scheduling.kproj (205 B, Oct)
- my\_BI\_schedule (462 B, Oct)

## Clear cron jobs

The screenshot shows the RStudio interface with two panes. The left pane is the 'Console' showing the R code and its execution:

```
27 print(paste0(date_time, " Value of increment_one.rds is ", increment_one))
28
29
30 cmd <- cron_script(script='/home/rstudio/scheduling/Scheduling/increment_one.rds')
31 cron_add(cmd, frequency = "minutely", id="job1", description="Our first cron job", days_of_week = 3)
32 cron_add(cmd, frequency = "daily", id="job2", description="Our second cron job", at = "04:00", days_of_month = 15)
33
34 cron_ls()
35 cron_save(file="my_BI_schedule")
36
37 cron_clear()
38
39
40 [1] (Top Level) Background jobs
41
42 (R 4.5.1 - scheduling) - scheduling
43 0-59 * * * 3 /usr/local/lib/R/bin/Rscript '/home/rstudio/scheduling/Scheduling/increment_one.rds' >> '/home/rstudio/scheduling/Scheduling/increment_one.log' 2>&1
44
45 ## cronR job
46 ## id: job2
47 ## tags:
48 ## desc: Our second cron job
49 0 4 15 * * /usr/local/lib/R/bin/Rscript '/home/rstudio/scheduling/Scheduling/increment_one.rds' >> '/home/rstudio/scheduling/Scheduling/increment_one.log' 2>&1
50
51 > cron_save(file="my_BI_schedule")
52
53 Saved crontab to file: /home/rstudio/scheduling/Scheduling/my_BI_schedule
54
55 > cron_clear()
56 Are you sure you want to clear all your cron jobs? [y/n]:
57 1: y
58 Crontab cleared.
59 >
```

The right pane is the 'File Browser' showing the contents of the directory:

- ignore (40 B, Oct)
- cron\_schedule\_examples.R (1.2 KB, Oct)
- increment\_one.log (19.8 KB, Oct)
- increment\_one.R (1.1 KB, Oct)
- increment\_one.rds (51 B, Oct)
- Scheduling.kproj (205 B, Oct)
- my\_BI\_schedule (462 B, Oct)

## Load Cron jobs from a file

`cron_load(file="my_BI_schedule")`

The screenshot shows the RStudio interface with two panes. The left pane is the 'Console' showing the R code and its execution:

```
28
29
30 cmd <- cron_script(script='/home/rstudio/scheduling/Scheduling/increment_one.rds')
31 cron_add(cmd, frequency = "minutely", id="job1", description="Our first cron job", days_of_week = 3)
32 cron_add(cmd, frequency = "daily", id="job2", description="Our second cron job", at = "04:00", days_of_month = 15)
33
34 cron_ls()
35 cron_save(file="my_BI_schedule")
36
37 cron_clear()
38
39 > cron_load(file="my_BI_schedule")
40
41 [1] (Top Level) Background jobs
42
43 (R 4.5.1 - scheduling) - scheduling
44
45 No cron jobs available for user 'rstudio'.
46 > cron_load(file="increment_one.log")
47 Are you sure you want to load the cron jobs available at 'increment_one.log'? [y/n]:
48 1: y
49 "increment_one.log":0: bad minute
50 errors in crontab file, can't install.
51 Error in parse_crontab(user ~ user) : No crontab available
52
53 > cron_load(file="my_BI_schedule")
54 Are you sure you want to load the cron jobs available at 'my_BI_schedule'? [y/n]:
55 1: y
56
57 Crontab with 2 cron jobs and 1 other job loaded.
58 >
```

## verify that they are active

cron\_ls()

The screenshot shows the RStudio interface with the following details:

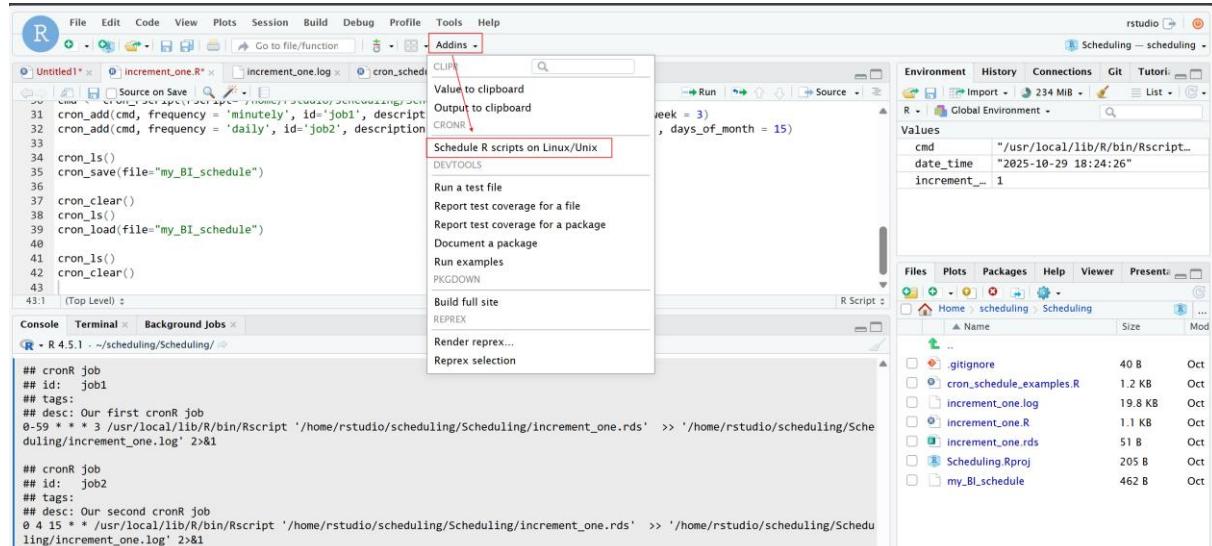
- Console:** Displays the command `> cron\_ls()` and its output:

```
## cronR job
## id: job1
## tags:
## desc: Our first cronR job
0-59 * * * * /usr/local/lib/R/bin/Rscript '/home/rstudio/scheduling/Scheduling/increment_one.rds' >> '/home/rstudio/scheduling/Scheduling/increment_one.log' 2>&1

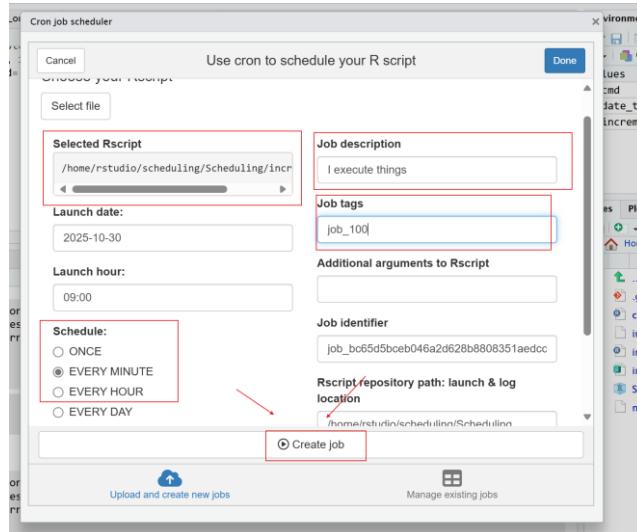
## cronR job
## id: job2
## tags:
## desc: Our second cronR job
0 4 15 * * /usr/local/lib/R/bin/Rscript '/home/rstudio/scheduling/Scheduling/increment_one.rds' >> '/home/rstudio/scheduling/Scheduling/increment_one.log' 2>&1
```
- Background Jobs:** Shows two jobs listed: job1 and job2.
- File Explorer:** Shows the directory structure under `~/scheduling/Scheduling`:

Name	Size	Last Modified
.gitignore	40 B	Oct
cron_schedule_examples.R	1.2 KB	Oct
increment_one.log	19.8 KB	Oct
increment_one.R	1.1 KB	Oct
increment_one.rds	51 B	Oct
Scheduling.Rproj	205 B	Oct
my_BI_schedule	462 B	Oct

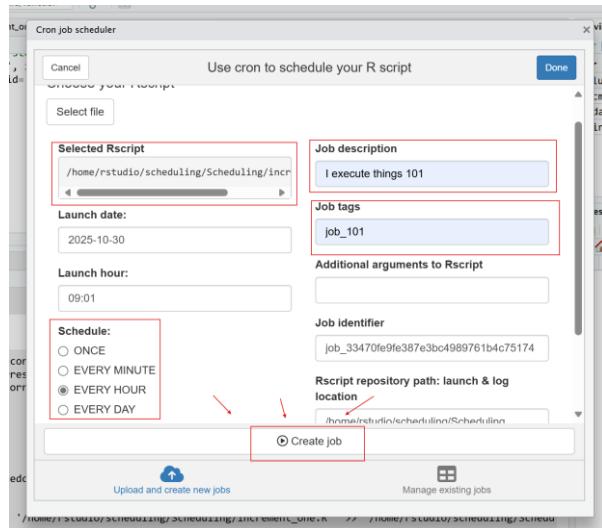
## Cron Job from the cronR addin



### set the “increment\_one.R” script to execute every minute



### set the “increment\_one.R” script to execute every hour

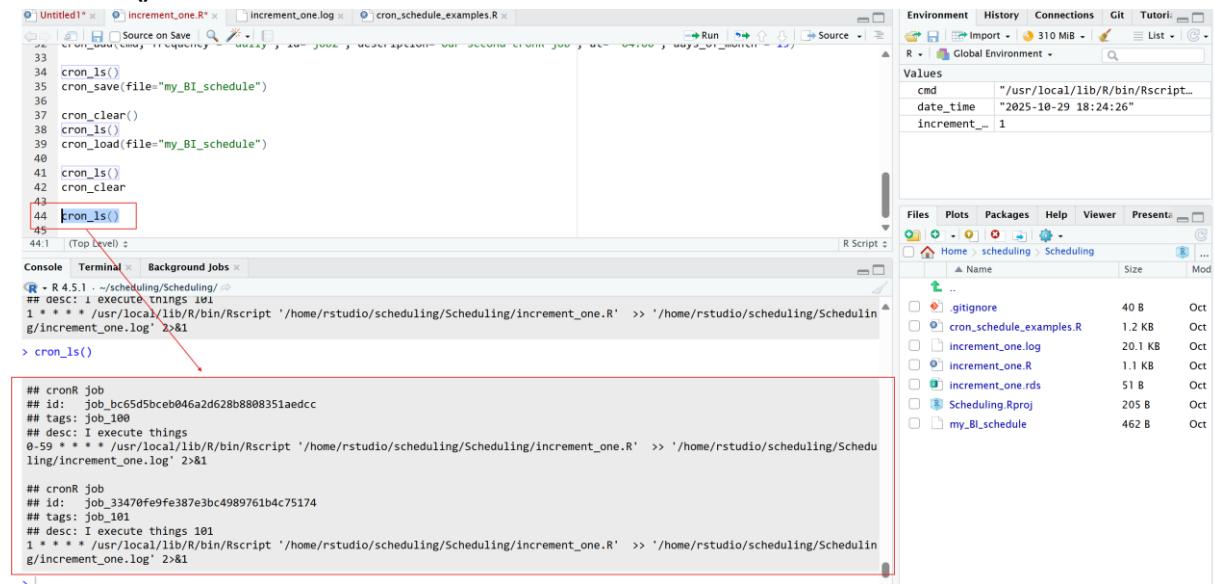


## Manage existing jobs, and confirm that you can see both jobs

The screenshot shows the RStudio interface with the following components visible:

- Top Bar:** File, Edit, Code, View, Plots, Session, Build, Debug, Profile, Tools, Help.
- Addins Menu:** Shows options like Value to clipboard, Output to clipboard, CRONR, SCHEDULE R scripts on Linux/Unix, DEVTOOLS, Run a test file, Report test coverage for a file, Report test coverage for a package, Document a package, Run examples, PKGDOWN, Build full site, REPREX, Render reprex..., Reprex selection.
- Console:** Displays R code and its output. The code includes cron-related functions like `cron\_id`, `cron\_clear()`, and `cron\_load`. It also shows the execution of two cron jobs: `job1` and `job2`.
- Scheduler Panel:**
  - Cron job scheduler:** A dialog box titled "Use cron to schedule your R script". It contains fields for "Selected Rscript" (No R script selected yet), "Job description" (I execute things), "Launch date" (2025-10-30), "Launch hour" (08:57), "Schedule" (radio buttons for ONCE, EVERY MINUTE, EVERY HOUR, with ONCE selected), "Job identifier" (job\_499093acc75267bff193b91e942d9ef), and "Rscript repository path: launch & log" (Create job button). A red arrow points from the "Manage existing jobs" button at the bottom right of this panel to the "Manage existing jobs" button in the bottom panel.
  - Environment Panel:** Shows the global environment with objects like `cmd`, `date\_time`, and `increment\_...`.
- Bottom Panel:**
  - Cron job scheduler:** Similar to the top panel, but for managing existing crontab schedules. It shows an "Existing crontab" section with a "Show current crontab schedule" button, a "Save crontab" section with a "Save current crontab schedule to" field (set to /home/rstudio/my\_schedule.cron), and a "Load crontab" section with "Select crontab schedule" and "Load selected schedule" buttons. A red box highlights the "Select job" dropdown menu which contains the job identifier "job\_bc65d5bceb046a2d628b8808351aedc".
  - Environment Panel:** Shows the global environment with objects like `cmd`, `date\_time`, and `increment\_...`.

## Or cron\_ls()



The screenshot shows the RStudio interface with several tabs open: 'Untitled1\*', 'increment\_one.R\*', 'increment\_one.log', and 'cron\_schedule\_examples.R'. The 'increment\_one.R\*' tab contains R code for scheduling tasks. The 'Console' tab shows the output of running the script, including cron entries. The 'Files' tab shows the project structure with files like 'gitignore', 'cron\_schedule\_examples.R', 'increment\_one.log', 'increment\_one.R', 'increment\_one.rds', 'Scheduling.Rproj', and 'my\_BI\_schedule'. The 'Environment' tab displays global variables. A red box highlights the 'cron\_ls()' function call in the code editor.

```
33
34 cron_ls()
35 cron_save(file="my_BI_schedule")
36
37 cron_clear()
38 cron_ls()
39 cron_load(file="my_BI_schedule")
40
41 cron_ls()
42 cron_clear
43
44 cron_ls()
45
```

```
R 4.5.1 .../scheduling/Scheduling/ >
## DESC: I execute things 1&1
1 * * * * /usr/local/lib/R/bin/Rscript '/home/rstudio/scheduling/Scheduling/increment_one.R' >> '/home/rstudio/scheduling/Scheduling/increment_one.log' 2>&1
> cron_ls()

## cronR job
## id: job_bc65d5bceb046a2d628b8808351aedcc
## tags: job_100
## desc: I execute things
0-59 * * * * /usr/local/lib/R/bin/Rscript '/home/rstudio/scheduling/Scheduling/increment_one.R' >> '/home/rstudio/scheduling/Scheduling/increment_one.log' 2>&1

## cronR job
## id: job_33470fe9fe387e3bc4989761b4c7517a
## tags: job_101
## desc: I execute things 1&1
1 * * * * /usr/local/lib/R/bin/Rscript '/home/rstudio/scheduling/Scheduling/increment_one.R' >> '/home/rstudio/scheduling/Scheduling/increment_one.log' 2>&1
```