Inav Documentation

Release 0.14.0

Tim Stack

Apr 12, 2026

CONTENTS

The Log File Navigator (**lnav**) is an advanced log file viewer for the console. If you have a bunch of log files that you need to look through to find issues, **lnav** is the tool for you.

Contents:

CHAPTER

ONE

INTRODUCTION

The Log File Navigator, **lnav**, is an advanced log file viewer for the terminal. It provides an *easy-to-use interface* for monitoring and analyzing your log files with little to no setup. Simply point **lnav** at your log files and it will automatically detect the *Log Formats*, index their contents, and display a combined view of all log messages. You can navigate through your logs using a variety of *hotkeys*. *Commands* give you additional control over **lnav**'s behavior for doing things like applying filters, tagging messages, and more. You can then analyze your log messages using the *SQLite Interface*.

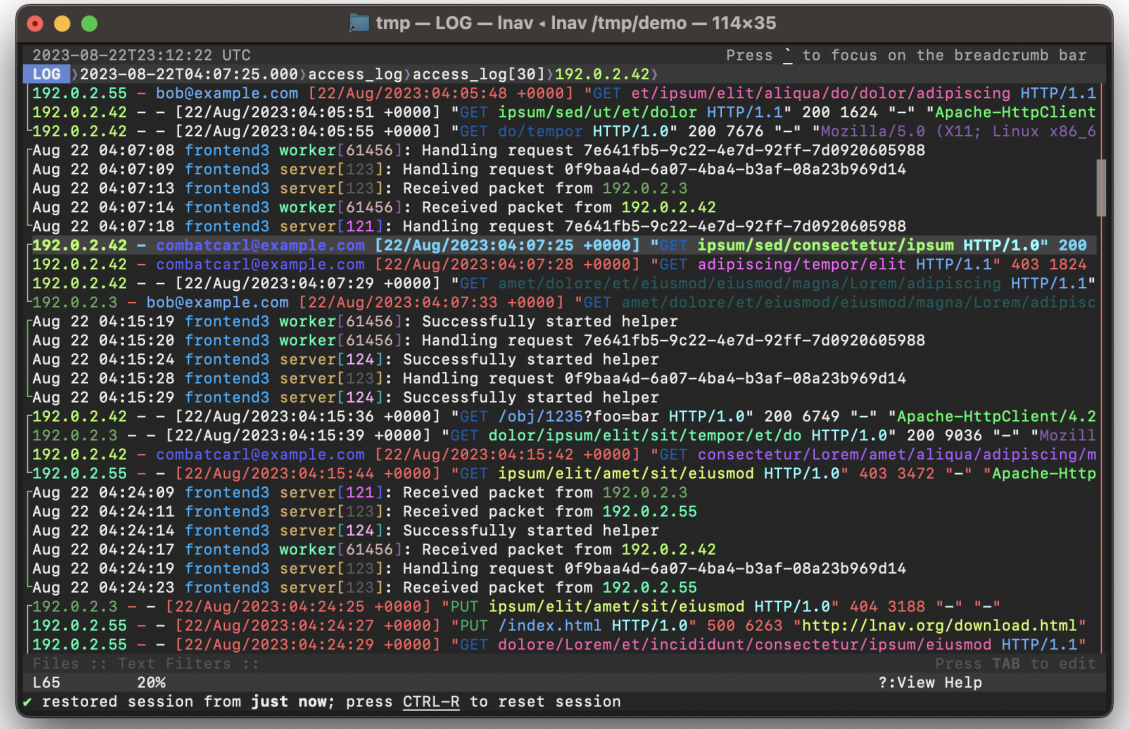

Fig. 1: Screenshot of **lnav** viewing syslog and web access_log messages.

1.1 Downloads

Binaries and source code for lnav can be downloaded from the releases page.

When building from source, follow the steps in *Development*.

1.2 Viewing Logs

The arguments to **lnav** are the log files, directories, or URLs to be viewed. For example, to view all of the CUPS logs on your system:

```
lnav /var/log/cups
```

The formats of the logs are determined automatically and indexed on-the-fly. See *Log Formats* for a listing of the predefined formats and how to define your own.

If no arguments are given, **lnav** will try to open the syslog file on your system:

```
lnav
```

1.3 Setup

After starting **lnav**, you might want to set the *configuration options* mentioned below. Configuration in **lnav** is done using the `:config` command. To change a configuration option, start by pressing `:` to enter the command prompt. Then, type "config" followed by the option name and value.

> **Note**
>
> Tab-completion is available for these configuration options and, in some cases, their values as well.

1.3.1 Keymap

The keymap defines the mapping from *hotkeys* to commands to execute. The default mapping is for "U.S." keyboards. The following command can be used to change the keymap:

```
:config /ui/keymap <keymap-name>
```

The builtin keymaps are:

de
: German

fr
: French

sv
: Swedish

uk
: United Kingdom

us
: United States

To create or customize a keymap, consult the *Keymap Definitions* section.

1.3.2 Theme

The visual styling of **lnav** can be customized using a theme. The following command can be used to the change the theme:

```
:config /ui/theme <theme-name>
```

The builtin themes are: default, dracula, eldar, grayscale, modus-operandi, monocai, night-owl, solarized-dark, and solarized-light.

To create or customize a theme, consult the *Theme Definitions* section.

1.3.3 Cursor Mode (v0.11.2+)

The default mode for scrolling in **lnav** is "cursor" mode where there is a cursor line in the view that is moved by the arrow keys and other interactions. Any interactions, such as jumping to a search hit, are then focused on that line.

Alternatively, you can enable "top" mode where the contents of the main view are moved when the arrow keys are pressed. Any interactions, such as jumping to a search hit, are then focused on the top line in the view. You can change to "top" mode with the following command:

```
:config /ui/movement/mode top
```

1.3.4 Mouse Support (v0.12.2+)

Mouse support can be enabled temporarily by pressing `F2` and can be set as the default by executing the following command:

```
:config /ui/mouse/mode enabled
```

See *Mouse Support (v0.12.2+)* for more details.

1.3.5 Log Formats

In order for **lnav** to understand your log files, it needs to be told how to parse the log messages using a log format definition. There are many log formats builtin and **lnav** will automatically determine the best format to use. In case your log file is not recognized, consult the *Log Formats* section for information on how to create a format.

1.4 Development

Development of lnav is hosted on GitHub.

Issues should be used for bugs and feature requests.

Discussions should be used for asking questions and sharing tips.

1.4.1 Dependencies

When compiling from source, the following dependencies are required:

- PCRE2
- SQLite
- ZLib
- Bzip2
- libcurl

- libarchive
- libunistring
- cargo/rust

1.4.2 Installation

Check the downloads page to see if there are packages for your operating system. To compile from source, use the following commands:

> **Note**
>
> Run `./autogen.sh` if compiling from the git repository.

```
./configure
make
sudo make install
```

CHAPTER

TWO

USER INTERFACE

The **lnav** TUI displays the content of the current "view" in the middle, with status bars above and below, and the interactive prompt as the last line.

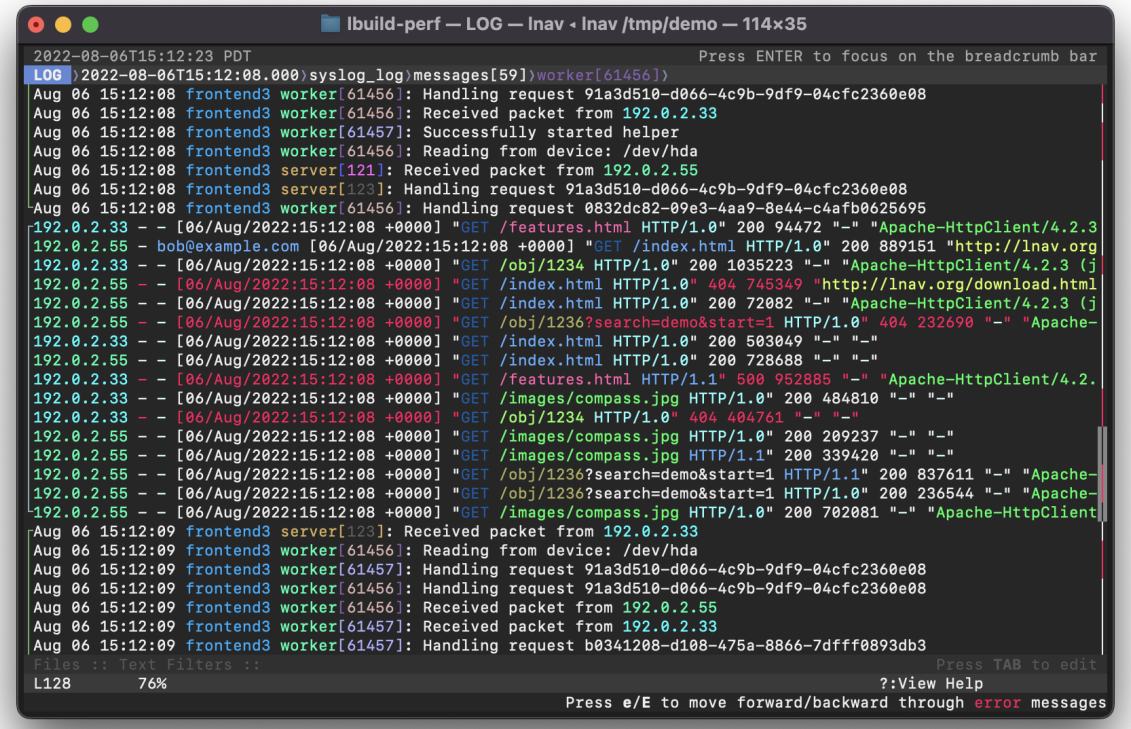

Fig. 1: Screenshot of **lnav** viewing syslog and web access_log messages.

The default view shows the log messages from the log files that have been loaded. There are other views for displaying content like plaintext files and SQL results. The *Views* section describes the characteristics of each view in more detail. You can switch to the different views using the hotkeys described in the *Display* section or by pressing ` to activate the breadcrumb bar, moving to the first crumb, and then selecting the desired view. You can switch back to the previous view by pressing `q`. You can switch forward to the new view by pressing `a`. If the views are time-based (e.g. log and histogram), pressing `Shift` + `q` and `Shift` + `a` will synchronize the top times in the views.

lnav provides many operations to work with the log/text data in the main view. For example, you can add comments and tags to log messages. The highlighted cursor line is used as the reference point to edit the comment or tags.

Alternatively, you can press `Ctrl + x` to switch to "top" mode where the "focused" line is the top line in the view and most operations now work with that line. When in "cursor" mode, the ↑ and ↓ keys now move the focused line instead of scrolling the view. Jumping to bookmarks, like errors, will also move the focused line instead of moving the next error to the top of the view.

The right side of the display has a proportionally sized 'scrollbar' that shows:

- the current position in the file;
- the locations of errors/warnings in the log files by using red or yellow coloring;
- the locations of search hits by using a tick-mark pointing to the left;
- the locations of bookmarks by using a tick-mark pointing to the right.

2.1 Top Status Bar

The top status bar shows the current time and messages stored in the *lnav_user_notifications* table.

Below the top status bar is the breadcrumb bar that displays the semantic location of the focused line in the main view. For example, within a pretty-printed JSON document, it will show the path to property at the top of the view. The actual content of the bar depends on the current view and will be updated as you navigate around the main view. The bar can also be used to navigate around the document by focusing on it.

2.2 Breadcrumb Bar

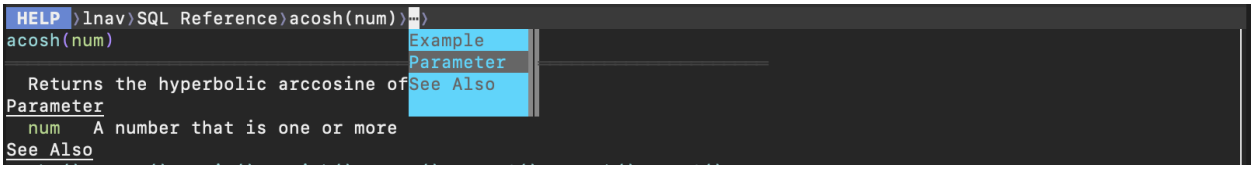

Fig. 2: Screenshot of the breadcrumb bar focused and navigating the help text

To focus on the breadcrumb bar, press `. The ←/→ cursor keys can be used to select a crumb and the ↑/↓ keys can be used select a value of that crumb. To accept a value and drop focus on the bar, press `ENTER`. To accept a value and move to the next crumb, press →. Using → makes it quicker to drill down into a document without having to constantly switch focus. To drop focus on the bar without accepting anything, press `Escape`.

There are three types of crumbs:

- a dropdown where one of a limited set of values can be selected;
- a combobox where a value can be entered directly or selected;
- a numeric input for entering array indexes.

When a dropdown or combobox is selected, you can type part of the desired value to filter the list of values. For example, the first crumb is always the current view, typing in "hi" will filter the list down to the "HIST" value.

2.3 Background Tasks

Some long-running tasks can be run in the background without blocking the UI. For example, the in-memory SQLite DB can be backed up to a file, which can take some time if the DB is large. The progress of these tasks will be displayed between the main view and the status bars at the bottom. While running, an hourglass will be shown with the current status of the progress. If the task finishes with some messages, they will be displayed in the same area for a short time before being hidden.

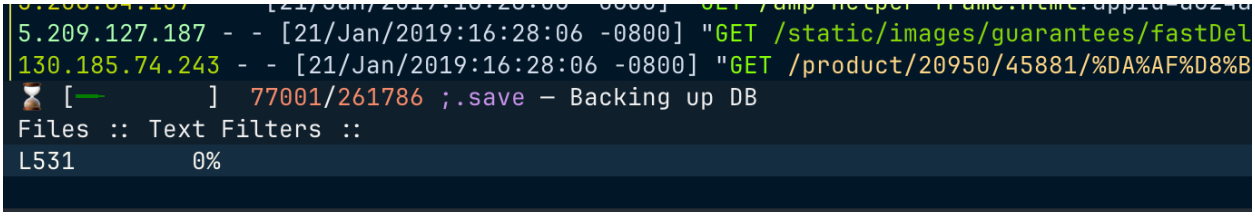

Fig. 3: Screenshot of the background task list showing a backup in progress

2.4 Configuration Panels

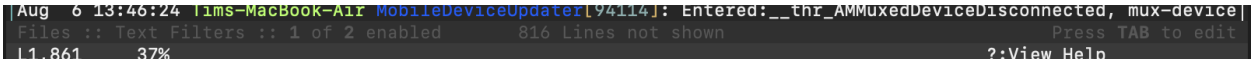

Fig. 4: Screenshot of the header for the configuration panels when they are hidden.

After the main view content, there is a header bar for two configuration panels: Files and Filters. These panels provide visual access to parts of lnav's configuration. To access the panels, press the `TAB` key. To hide the panels again, press `q`.

2.4.1 Files

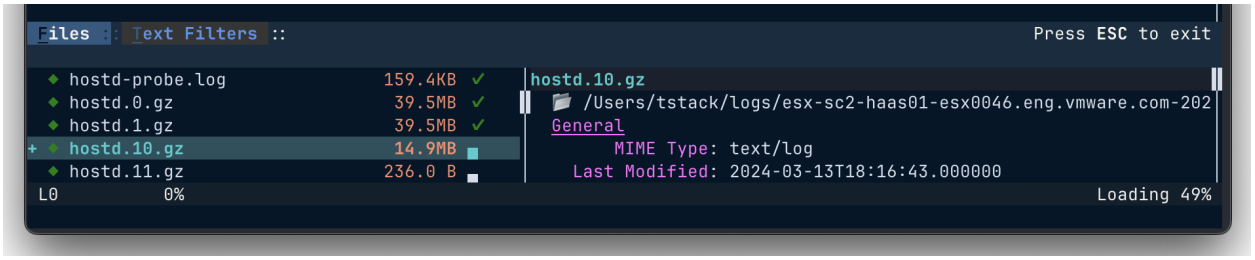

Fig. 5: Screenshot of the files panel showing the loaded files.

The Files panel is open initially to display progress in indexing files. The following information is displayed for each file:

- the "unique" portion of the path relative to the other files;
- the amount of data that has been indexed;
- and, a progress bar or an icon to show the status of the file. The following icons may appear:
  - green check mark (✓) if the file was successfully indexed;
  - yellow warning () if there are notes for the file, such as why it was hidden
  - red X () if there was an error reading the file.

To the right of the file list is a panel that shows details for each file. You can focus on the details view by pressing `CTRL + ]`. Press `Esc` to return focus to the file listing.

The following information is shown in the details view:

- the date range of log messages contained in the file;
- the errors that were encountered while trying to index the file;

- the notes recorded for files where some automatic action was taken, like hiding the file if it was seen as a duplicate of another file;
- the details of the demultiplexing and log format matching process.

2.4.2 Filters

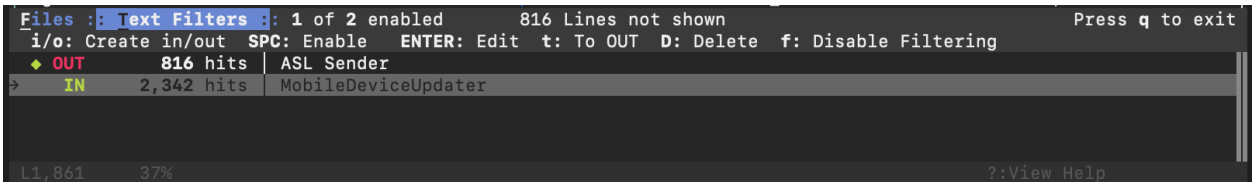

Fig. 6: Screenshot of the filters panel showing an OUT and a disabled IN filter.

If the view supports filtering, there will be a status line showing the following:

- the number of enabled filters and the total number of filters;
- the number of lines that are **not** displayed because of filtering.

To edit the filters, you can press TAB to change the focus from the main view to the filter editor. The editor allows you to create, enable/disable, and delete filters easily.

2.5 Bottom Status Bar

The second to last line is the bottom status bar, which shows the following:

- the line number of the focused line, starting from zero;
- the location within the view, as a percentage;
- the current search hit, the total number of hits, and the search term;
- the loading indicator.

When the interactive prompt is active, this bar can show the prompt description, help text, or error message. You can also click and drag this bar to resize the prompt when in multi-line mode.

2.6 Prompt

Finally, the last line on the display is where you can enter search patterns and execute internal commands, such as converting a unix-timestamp into a human-readable date. The following key-presses will activate a corresponding prompt:

- `/` - The search prompt. You can enter a PCRE2-flavored regular expression to search for in the current view.
- `:` - The command prompt. Commands are used to perform common operations.
- `;` - The SQL prompt. SQL queries can be used for log analysis and manipulating **lnav**'s state.
- `|` - The script prompt. Enter a path to the lnav script to execute, along with the arguments to pass in.

The prompt has emacs-like keyboard shortcuts that can be used for editing and moving within the command-line.

To cancel the prompt, you must press `Esc` twice. This is to prevent accidentally closing the prompt when closing a completion popup.

> **Note**
>
> Commands can also be executed by pasting a snippet when focused on the main view, without opening a prompt. The pasted snippet should start with the appropriate sigil, for example, the colon in `:echo Hello, World!`. This functionality requires the terminal to support bracketed paste mode.

2.7 Views

The accessible content within lnav is separated into multiple views. The LOG view is shown by default when log files are detected. You can switch to the TEXT view by pressing `t` and to the HELP view by pressing `?`. Other views can be reached through the breadcrumb bar (press ` and select a view from the first crumb) or with the *:switch-to-view* command. Pressing `q` will return to the previous view and `a` will go forward again. For time-based views (e.g. LOG, HIST, TIMELINE), pressing `Shift` + `Q` and `Shift` + `A` will switch views while keeping the focused time in sync.

The LOG, TEXT, and TIMELINE views support pinning important log lines to the top of the view so they remain visible as you scroll. This is useful for keeping context visible, such as a request header, session start message, or a CSV header.

To pin the focused line as a sticky header, press `CTRL` + `s` or use the *:toggle-sticky-header* command. Pinned lines are displayed at the top of the view with the status bar styling. Running the command again on a pinned line will unpin it. To remove all sticky headers at once, use the *:clear-all-sticky-headers* command.

2.7.1 LOG

The log view displays the log messages from any loaded log files in time order. This view will be shown by default if any log files were detected. If plain text files were also loaded, they will be available in the TEXT view, which you can switch to by pressing `t`.

On color displays, the log messages will be highlighted as follows:

- Errors will be colored in red;
- warnings will be yellow;
- search hits are reverse video;
- various color highlights will be applied to: IP addresses, SQL keywords, XML tags, file and line numbers in Java backtraces, and quoted strings;
- "identifiers" in the messages will be randomly assigned colors based on their content (works best on "xterm-256color" terminals).

> **Tip**
>
> If the coloring is too much for your tastes, you can change to the "grayscale" theme by entering the following command:
>
> ```
> :config /ui/theme grayscale
> ```

Timestamps in log messages will be rewritten to the local timezone (or the timezone specified by *TZ*) automatically if they include a timezone component. If a file's timestamps do not include a timezone, they will be treated as if they are from the local zone. You can change the zone to use for these types of files using the *:set-file-timezone* command.

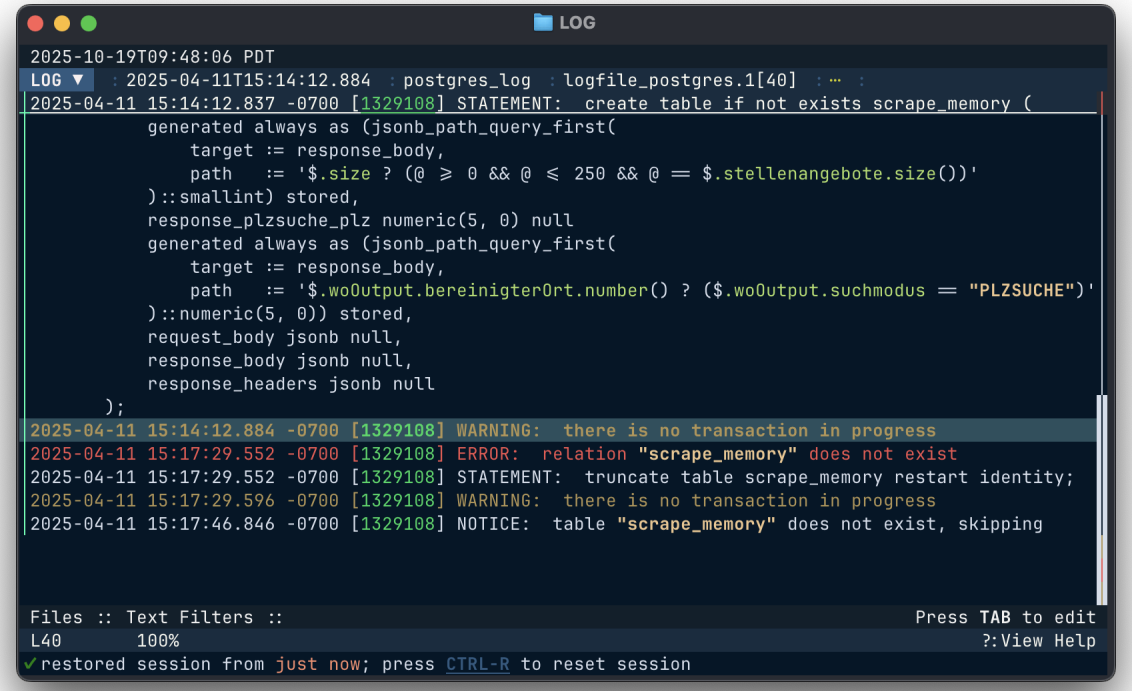

Fig. 7: Screenshot of the **lnav** LOG view displaying a PostgreSQL log. Since the log message at the top of the view is partially scrolled off the screen, the first line of the message is displayed at the top. You can left-click the header to scroll to the first line of the message or press {.

> **Note**
>
> If a log message has a timestamp that is out-of-order with its neighboring messages, the timestamp will be highlighted in yellow. When one of these messages is focused, an overlay will display the difference between the "actual time" and the "received time". The "actual time" is the original textual timestamp. The "received time" is the time of an earlier message that is larger than this log message's time.

To get more details about a log message, move to the starting line of the message and press `p` to open an overlay with information about the timestamp and individual fields. You then can focus into the overlay panel by pressing `Ctrl + ]` to operate on individual fields in the message. Inside the overlay, the following hotkeys are supported:

- `SPC` to toggle field visibility
- `c` to copy the value of a field
- `#` to open the SQL prompt with a query to create a chart of this field's values over time.

> **Tip**
>
> The "Permalink" line in the details overlay contains a string that you can use to reference this particular message. It can be used as an argument to the *:goto* command, the target of a Markdown link in a log message comment, or compared to the `log_line_link` link in log tables.

The log file name for each message can be displayed by scrolling left. Scrolling left once will show the shortened version of the file name relative to the other files that are loaded. In the shortened version, the unique portion of the file name will be in square brackets. Scrolling left a second time will show the full path.

The breadcrumb bar will show the following crumbs:

- the timestamp for the focused line;
- the log format for the focused line;
- the name of the file the focused line was pulled from;
- the "operation ID" of the focused log message, if it is supported by the log format.

These crumbs are interactive and can be used to navigate to different parts of the log view. For example, selecting a different value in the log format crumb will jump to the first message with that format.

The file crumb will show a "" icon if the file is from the output of a FIFO, `:sh` command, or data that was piped into the standard input. When the pipe is closed, the icon will disappear.

2.7.2 TEXT

The text view displays files for which lnav could not detect any log messages.

Press `t` to switch to the text view. While in the text view, you can press `f` or `Shift + F` to switch to the next / previous text file.

The breadcrumb bar will show the name of the file currently displayed file. To switch to a different file, you can select the file crumb and search for the file you want to focus on. If the file is the output of a FIFO, `:sh` command, or data that was piped into the standard input, the file crumb will show a "" icon. When the pipe is closed, the icon will disappear.

If the content is piped into lnav through standard input, a FIFO, or a `:sh` command, the time that lines are received are recorded. You can press `Shift + T` to view the elapsed time like in the LOG view. The breadcrumb bar will also show the received time of the focused line after the file name crumb. If the output being shown is from a `:sh` command, you can press `Ctrl + C` to send a SIGINT to the child process without killing **lnav** itself.

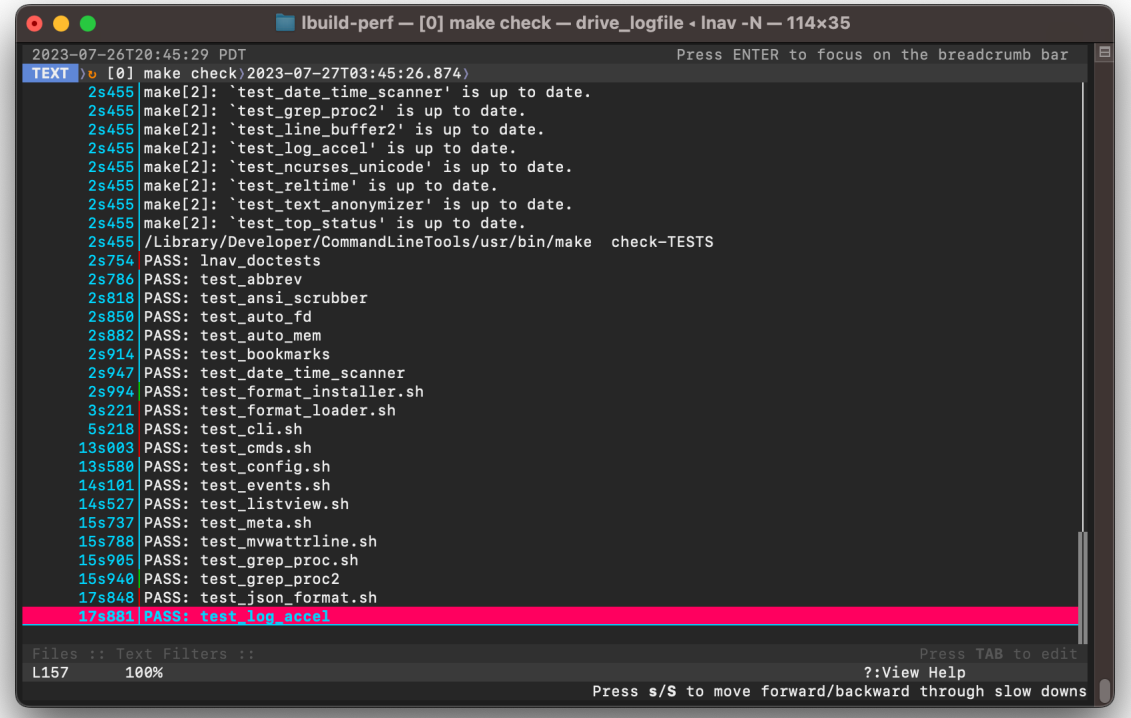

Fig. 8: Screenshot of the TEXT view showing the output of `sh make check`. Each line is timestamped internally when it was received so it's possible to view how long each test is taking to run. The "" icon next to the file name in the breadcrumb bar means that the make is still running.

If the document has some recognizable structure, the breadcrumb bar will be updated with the path to the focused line. The following document types are recognized: JSON, XML, TOML, man, diff, and Markdown.

Binary Files

Files that contain binary data will be displayed as a hexdump. Each line shows 16 bytes as hex values followed by their character representations. The following byte types are highlighted to make them easier to identify:

- **NULL bytes** (0x00) – displayed as a diamond () in the character column.
- **ASCII control characters and whitespace** – whitespace characters are displayed as an underscore (_) and control characters as a bullet (•).
- **Non-ASCII bytes** – displayed as a multiplication sign (×) in the character column.

Printable ASCII characters are displayed as-is without any special styling.

Markdown

Files with an `.md` (or `.markdown`) extension will be treated as Markdown files and rendered separately.

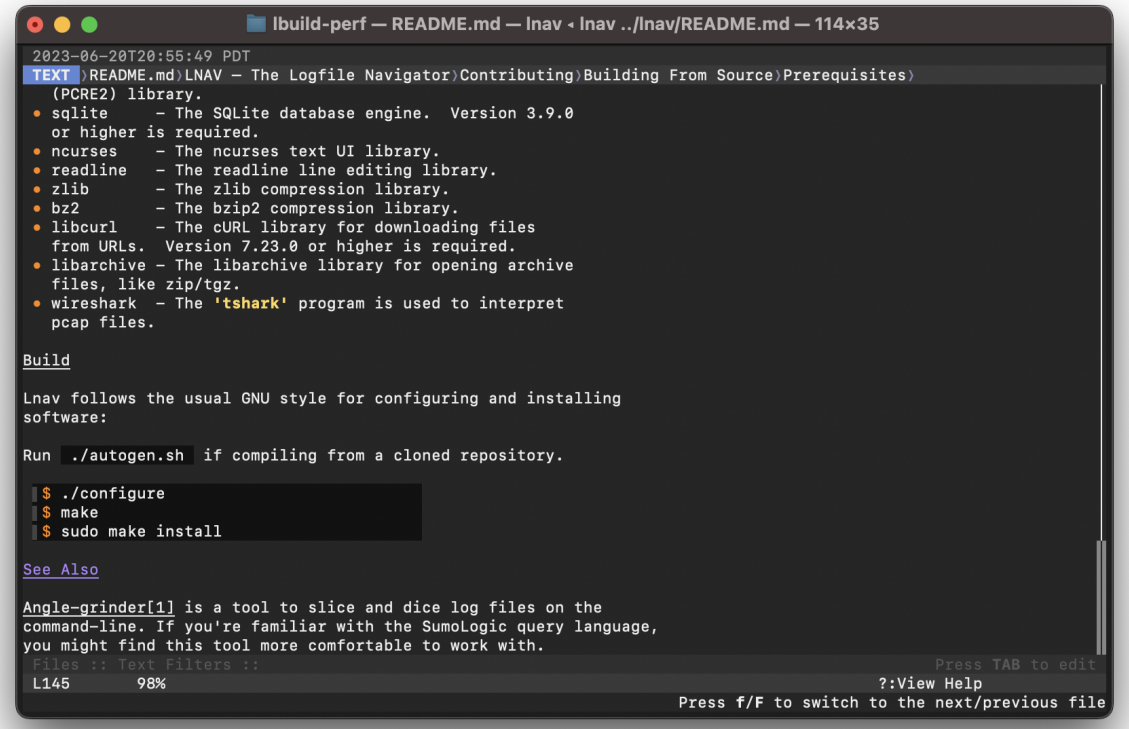

Fig. 9: Viewing the **lnav** `README.md` file.

In addition to the standard Markdown syntax, lnav supports the following:

- GitHub Alerts
- **HTML tags with the following CSS properties:**
  - `color` and `background-color` with CSS color names

  - `font-weight` with a value of `bold` or `bolder`
  - `text-decoration` with `underline`
  - `border-left` and `border-right` with the `solid`,
  - `dashed` and `dotted` line styles and colors.
  - `white-space` with a value of `nowrap`
- Anchors can be explicitly declared with `<a name="...">`.
- Internal links can be clicked to move to that section of the document.
- Code blocks with the language set to `lnav` will have a play button () next to it that can be clicked to execute the given code.

2.7.3 DB

The DB view shows the results of queries done through the SQLite interface. You can execute a query by pressing `;` and then entering a SQL or PRQL[1] statement.

Press `v` to switch to the database result view.

2.7.4 HELP

The help view displays the builtin help text. While in the help view, the breadcrumb bar can be used to navigate to different sections of the document.

Press `?` to switch to the help view or use the command `:help`.

2.7.5 HIST

The histogram view displays a stacked bar chart of messages over time classified by their log level and whether they've been bookmarked.

Press `i` to switch back and forth to the histogram view. You can also press `Shift + i` to toggle the histogram view while synchronizing the top time. While in the histogram view, pressing `z` / `Shift + z` will zoom in/out.

2.7.6 TIMELINE

> **Note**
>
> This feature is available in v0.12.0+.

The timeline view[2] visualizes operations, log files, threads, tags, and partitions over time. The items are ordered top-to-bottom by their start time. So, scrolling down will move forward in time.

An operation is identified by an ID that can come from multiple sources:

- If the ID is in the log message, the log format can set the `opid-field` property. This option is useful if the software explicitly keeps track of the operations.
- If an ID should be generated from multiple parts of the log message, the log format should create an `opid/description`. This description contains a `format` array that specifies the fields in the message to hash together. For example, web log formats can use the client IP and User-Agent that will roughly correspond to a unique visitor.

[1] lnav must be compiled in an environment with Rust/Cargo available for PRQL support.
[2] Formerly called the "Gantt Chart" view.

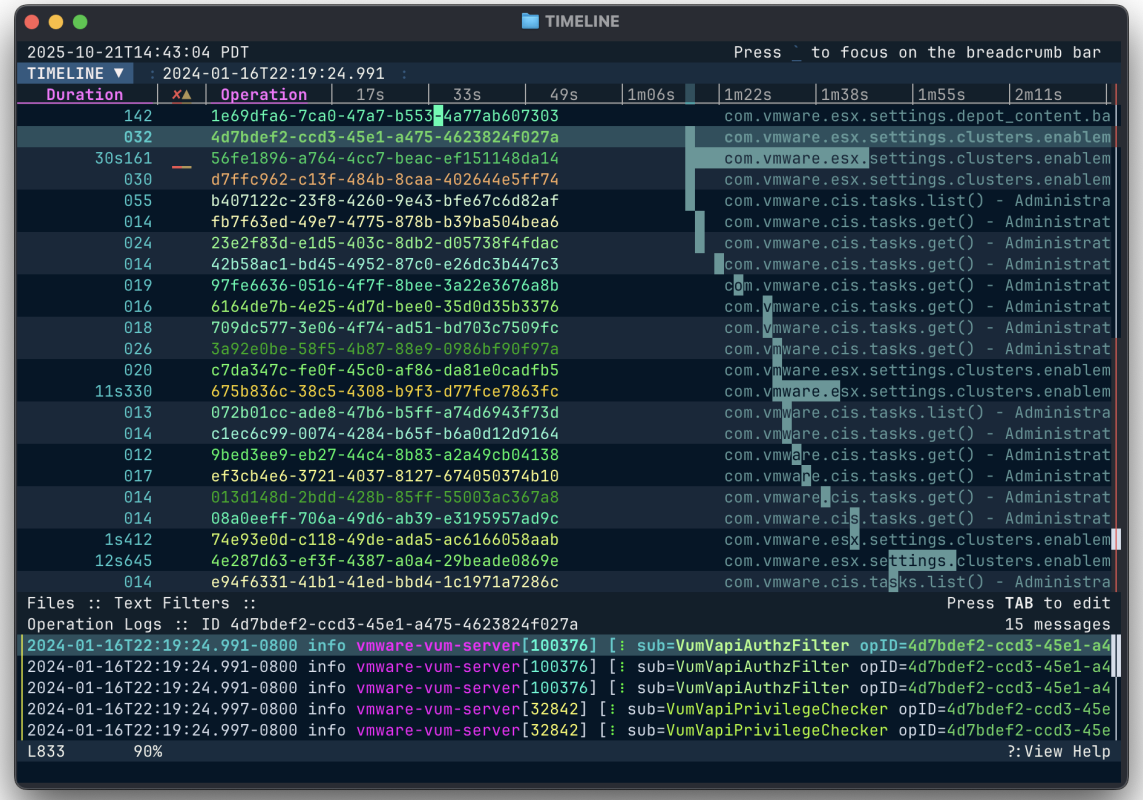

Fig. 10: Screenshot of the timeline view when viewing logs from the VMWare Update Manager. Most rows show API requests as they are received and processed.

- If a log format defines a duration, an ID will be generated from a hash of the entire log message. This option is useful for low-level logs where there is no overarching operation to tie them together.
- If an ID cannot be generated through these simple methods, one can be set manually by doing an `UPDATE` of the `log_opid` column on the log vtable.

The time span of an item is determined by the earliest and latest timestamps of messages associated with the item. If the log messages contain a duration, that will also be used in the calculation. The span is shown in the view using a reverse-video bar. The time scale of the view is automatically adjusted to fit the item at the top and bottom. The current scale is shown in the header.

Each row in the view shows:

- The duration of the item.
- Sparklines showing the number of errors and warnings relative to the total number of messages associated with the item.
- The ID of the item.
- A description of the item. For operations, the description is captured from the log message or a description can be set by doing an `UPDATE` of the `all_opids` table.

The preview panel at the bottom of the display will show the messages associated with the focused item.

The following hotkeys can be useful in this view:

- `ENTER` – Focus on the preview panel. Pressing `q` or `Escape` will change the focus back to the main view.
- `p` – If the log format defined sub-operations with the `opid/subid` property, this will toggle an overlay panel that displays the sub-operation descriptions.
- `Shift + q` – Return to the previous view and change its focused line to match the time that was focused in the timeline view.
- `Shift + a` – After leaving the timeline view, pressing these keys will return to the timeline view while keeping the focused time in sync.

2.7.7 PRETTY

The pretty-print view takes the text displayed in the current view and shows the result of a pretty-printer run on that text. For example, if a log message contained an XML message on a single line, the pretty-printer would break the XML across multiple lines with appropriate indentation.

Press `Shift + P` to switch to the pretty-print view.

2.7.8 SCHEMA

The schema view displays the current schema of the builtin SQLite database.

Press `;` to enter the SQL prompt and then enter `.schema` to open the schema view.

2.7.9 SPECTRO

The spectrogram view is a "three"-dimensional display of data points of a log field or a SQL query column. The dimensions are time on the Y axis, the range of data point values on the X axis, and the number of data points as a color. For example, if you were to visualize process CPU usage over time, the range of values on the X axis would be CPU percentages and there would be colored blocks at each point on the line where a process had that CPU percentage, like so

The colors correspond to the relative number of data points in a bucket. The legend overlaid at the top line in the view shows the counts of data points that are in a particular color, with green having the fewest number of data points, yellow

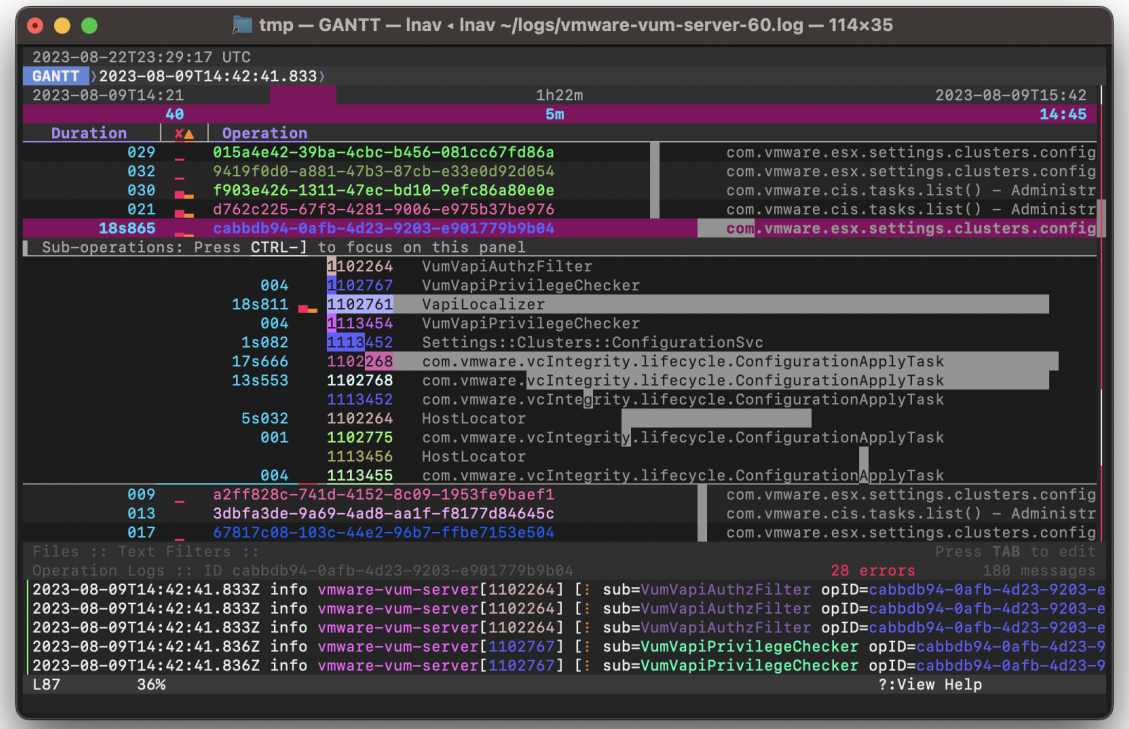

Fig. 11: Screenshot showing the same log as above after pressing p. The overlay panel shows a breakdown of sub-operations performed while processing the main operation.

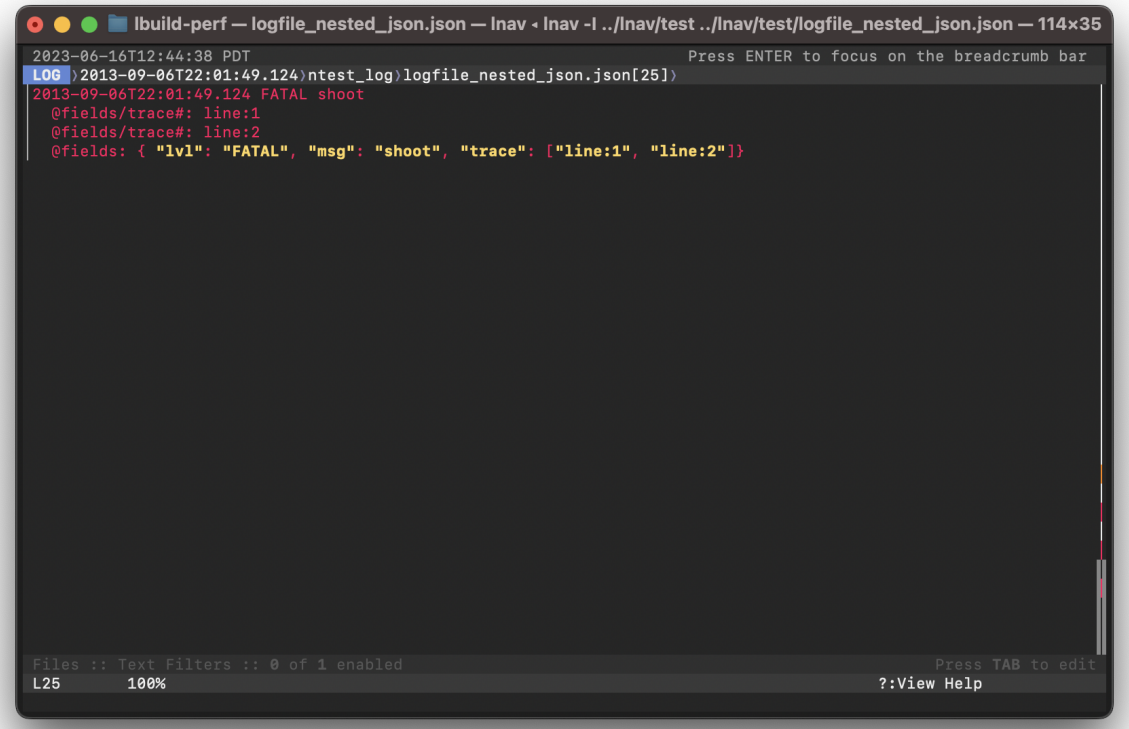

Fig. 12: Screenshot of a log message with a flat JSON object.

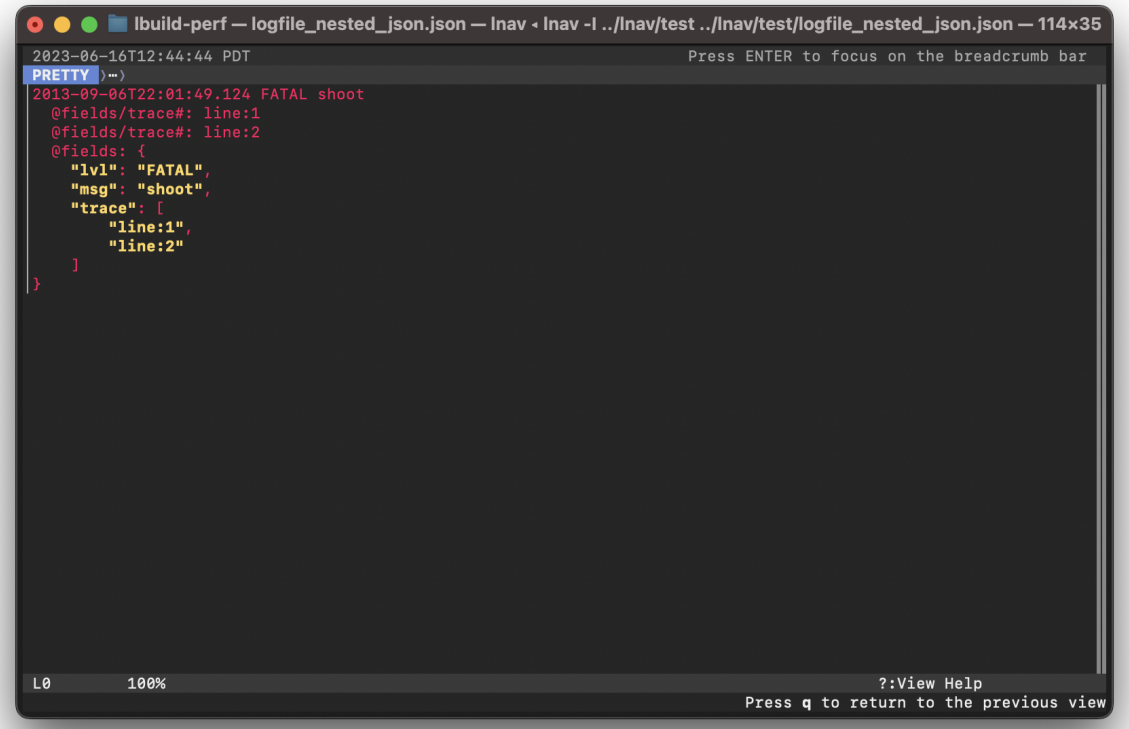

Fig. 13: Screenshot of the same log message in the PRETTY view. The JSON object is now indented for easier reading.

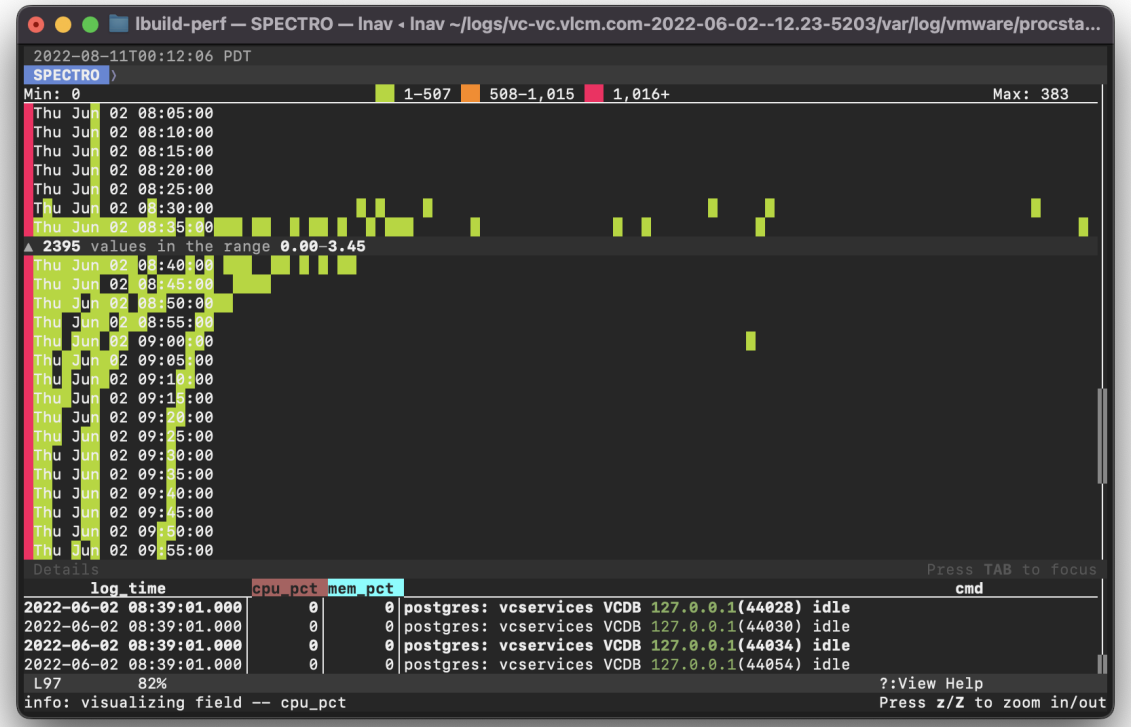

Fig. 14: Screenshot of the **lnav** spectrogram view showing CPU usage of processes.

the middle, and red the most. You can select a particular bucket using the cursor keys to see the exact number of data points and the range of values. The panel at the bottom of the view shows the data points themselves from the original source, the log file or the SQL query results. You can press `TAB` to focus on the details panel so you can scroll around and get a closer look at the values.

2.8 Mouse Support (v0.12.2+)

With mouse support enabled, either through the */ui/mouse/mode* configuration option or by pressing `F2`, many of the UI elements will respond to mouse inputs:

- clicking on the main view will move the cursor to the given row and dragging will scroll the view as needed;
- `Shift` (or `CTRL`) clicking/dragging in the main view will highlight lines and then toggle their bookmark status on release;
- double-clicking in the main view will select the underlying text and drag-selecting within a line will select the given text;
- when double-clicking text: if the mouse pointer is inside a quoted string, the contents of the string will be selected; if the mouse pointer is on the quote, the quote will be included in the selection; if the mouse pointer is over a bracket (e.g. [],{},()) where the matching bracket is on the same line, the selection will span from one bracket to the other;
- when text is selected, a menu will pop up that can be used to filter based on the current text, search for it, or copy it to the clipboard;
- right-clicking the start of a message in the LOG view or a row in the DB view will open the details overlay;
- the details overlay displays a diamond next to fields to indicate whether they are shown/hidden and this can be clicked to toggle the state;
- the log details will show a bar chart icon for fields with values which, when clicked, will open either the spectrogram view for the given field or open the DB query prompt with a PRQL query to generate a histogram of the field values;
- left-clicking a local link in a Markdown document will move the cursor to that part of the document;
- right-clicking a link in a Markdown document will open an overlay menu with relevant options;
- clicking in the scroll area will move the view by a page, double-clicking will move the view to that area, and dragging the scrollbar will move the view to the given spot;
- clicking on the breadcrumb bar will select a crumb and selecting a possibility from the popup will move to that location in the view;
- clicking on portions of the bottom status bar will trigger a relevant action (e.g. clicking the line number will open the command prompt with `:goto <current-line>`);
- clicking on the configuration panel tabs (i.e. Files/Filters) will open the selected panel and clicking parts of the display in there will perform the relevant action (e.g. clicking the diamond will enable/disable the file/filter);
- clicking in a prompt will move the cursor to the location;
- clicking on a column in the spectrogram view will select it.

Note

A downside of enabling mouse support is that normal text selection and copy will no longer work. While lnav has some support for selection in the main view, there are still likely to be cases where that is insufficient. In those

cases, you can press `F2` to quickly switch back-and-forth. Or, some terminals have support for switching while a modifier is pressed:

Key	Terminal
`Option`	iTerm, Hyper
`Fn`	Terminal.app

CHAPTER

THREE

HOTKEY REFERENCE

This reference covers the keys used to control **lnav**. Consult the built-in help in **lnav** for a more detailed explanation of each key.

3.1 Global

Keypress	Command
`Ctrl` + `C`	If the focused line is from a file connected to an open pipe (the "" icon will be next to the file name), a SIGINT will be sent to the child process. Otherwise, **lnav** will quickly exit. If **lnav** seems to be stuck in a loop, pressing `Ctrl` + `C` three times will trigger an abort exit.

3.2 Spatial Navigation

The majority of these hotkeys should be available in all views.

Keypress			Command
Space	PgDn		Down a page
Ctrl + d			Down by half a page
b	Backspace	PgUp	Up a page
Shift + b			Move to the first line of a multi-line log message. If the currently focused line is the first, move up by a tenth of the content height.
Ctrl + u			Up by half a page
j	↓		Down a line
k	↑		Up a line
h	←		Left half a page. In the log view, pressing left while at the start of the message text will reveal the shortened source file name for each line. Pressing again will reveal the full path.
Shift + h	Shift + ←		Left ten columns
l	→		Right half a page. In the log view, pressing right while at the start of the message text can hide the timestamp/level fields in the message and insert a shorter timestamp column on the left side. The column should take less space than the existing field and aligns all timestamps across all log formats. This feature is gated by the */ui/views/log/time-column* setting.
Shift + l	Shift + →		Right ten columns
Home	g		Top of the view
End	G		Bottom of the view
e	Shift + e		Next/previous error
w	Shift + w		Next/previous warning
n	Shift + n		Next/previous search hit
>	<		Next/previous search hit (horizontal)
f	Shift + f		Next/previous file
u	Shift + u		Next/previous bookmark
o	Shift + o		Forward/backward through log messages with a matching "opid" field
s	Shift + s		Next/previous slow down in the log message rate
{	}		Previous/next section in the view. For the LOG view, if the cursor is in the middle of a multi-line log message, this will move to the start of the message or the next message. Otherwise, it will move to the adjacent *partition*. For the TEXT view, the following document types have recognizable sections: JSON, man, diff, and Markdown.
F7	F8		Previous/next breakpoint
CTRL + b			Toggle a breakpoint for the focused log line

3.3 Chronological Navigation

These hotkeys are only functional on views that are time-based, like the log view or the histogram view.

Keypress		Command
d	Shift + d	Forward/backward 24 hours
1 - 6	Shift + 1 - 6	Next/previous n'th ten minute of the hour
7	8	Previous/next minute
0	Shift + 0	Next/previous day
r	Shift + r	Forward/backward by the relative time that was last used with the goto command.

3.4 Breadcrumb Navigation

The following hotkeys are related to the breadcrumb bar that is below the top status bar.

Keypress	Description
`	Focus on the breadcrumb bar.
ENTER	If the bar is currently focused, accept the selected value and drop focus.
Escape	Drop focus on the breadcrumb bar.
←	Select the crumb to the left. If the first crumb is selected, the selection will wrap around to the last crumb.
→	Accept the current value, which might mean navigating to the value in the view, then selecting the crumb to the right.
Ctrl + a	Select the first crumb.
Ctrl + e	Select the last crumb.
↓	Select the next value in the crumb dropdown.
↑	Select the previous value in the crumb dropdown.
Home	Select the first value in the crumb dropdown.
End	Select the last value in the crumb dropdown.

While a crumb is selected, you can perform a fuzzy search on the possible values by typing in the value you are interested in.

3.5 Bookmarks

Keypress	Command
m	Mark/unmark the top line or focused line when in cursor mode
Shift + m	Mark/unmark the range of lines from the last marked to the top
Shift + j	Mark/unmark the next line after the previously marked
Shift + k	Mark/unmark the previous line
c	Copy marked lines to the clipboard
Shift + c	Clear marked lines
Ctrl + s	Toggle a sticky header for the focused line. Sticky headers remain visible at the top of the view as you scroll past them.

3.6 Display

Keypress	Command
? or F1	View/leave builtin help
q	Return to the previous view/quit
Shift + q	Return to the previous view/quit while matching the top times of the two views
a	Restore the view that was previously popped with 'q/Q'
Shift + a	Restore the view that was previously popped with 'q/Q' and match the top times of the views
Shift + p	Switch to/from the pretty-printed view of the displayed log or text files
Shift + t	Display the elapsed time from a bookmark to a given line. In the TEXT view, this only works for content that was captured from stdin or a :sh command.
t	Switch to/from the text file view
i	Switch to/from the histogram view
Shift + i	Switch to/from the histogram view while keeping the time of the focused line in sync
z	Zoom in to finer-grained times in the HIST or SPECTRO views.
Shift + Z	Zoom out to larger-grained times in the HIST or SPECTRO views.
v	Switch to/from the SQL result view
Shift + v	Switch to/from the SQL result view and move to the corresponding in the log_line column
p	Toggle the display of the log parser results
Tab	In the log/text views, focus on the configuration panel for editing filters and examining the list of loaded files.
Ctrl + l	Switch to lo-fi mode. The displayed log lines will be dumped to the terminal without any decorations so they can be copied easily.
Ctrl + w	Toggle word-wrap.
Ctrl + p	Show/hide the data preview panel that may be opened when entering commands or SQL queries.
Ctrl + f	Toggle the enabled/disabled state of all filters in the current view.
x	Toggle the hiding of log message fields. The hidden fields will be replaced with three bullets and highlighted in yellow.
Ctrl + x	Toggle the cursor mode. Allows moving the focused line instead of keeping it fixed at the top of the current screen.
=	Pause/unpause loading of new file data.
X	Close the current text file or log file.
F2	Toggle mouse support.

3.7 Session

Keypress	Command
Ctrl + R	Reset the current *session* state. The session state includes things like filters, bookmarks, and hidden fields.

3.8 Query Prompts

Keypress	Command
/	Search for lines matching a regular expression
;	Open the *SQLite Interface* to execute SQL statements/queries
:	Execute an internal command, see *Commands* for more information
|	Execute an lnav script located in a format directory
Ctrl +]	Abort the prompt

3.9 Prompt

After activating the prompt, the following shortcuts are available for editing:

Keypress		Command
Escape		Depending on the mode: close the completion popup; exit search mode; cancel the prompt
Enter		In single line mode, exit the prompt and perform the operation. In multi-line mode, insert a new line.
Ctrl + X		Exit the prompt and perform the operation
F1		Open the help text for the prompt
←		Move left one character
→		Move right one character
Shift + ←		Move left one character and include it in the selection
Shift + →		Move right one character and include it in the selection
↑	Ctrl + P	In single-line mode, search the history with the current contents. In multi-line mode, move the cursor up a line. If a popup is open, move the selection up.
↓	Ctrl + N	In multi-line mode, move the cursor down a line. If a popup is open, move the selection down.
Ctrl + A		Move to the beginning of the line
Ctrl + E		Move to the end of the line
Ctrl + K		Cut to the end of the line into the clipboard
Ctrl + U		Cut from the beginning of the line to the cursor into the clipboard
Ctrl + W		Cut from the beginning of the previous word into the clipboard
Ctrl + Y		Paste the clipboard contents
Tab	Enter	Accept a completion suggestion
Ctrl + _		Undo a change
Ctrl + L		In the SQL prompt, reformat the contents of the prompt and switch to multi-line mode
Ctrl + O		Save the contents of the prompt in a script file named `saved-prompt` and, if available, open the file in the *configured external editor*
Ctrl + S		In multi-line mode, switch to search mode. If already in search mode, find the next occurrence.
Ctrl + R		In search mode, search for the previous occurrence. Otherwise, search history using current contents of the prompt

3.10 Customizing

New hotkeys can be defined and existing behaviors changed by updating lnav's configuration. Since most of lnav's functionality is exposed through *SQLite functions* and *SQLite virtual tables*, quite a lot is possible. For example, if you regularly want to find a log message with a particular pattern, like "Starting execution". The following command will find the next occurrence after the currently focused message:

```
;SELECT log_line FROM all_logs WHERE log_line > log_msg_line() AND log_body LIKE '
↪%Starting execution%' LIMIT 1
```

To then move the focus in the log view to the line found by the above command, you can run the following:

```
;UPDATE lnav_views SET selection = (SELECT log_line FROM all_logs WHERE log_line > log_
↪msg_line() AND log_body LIKE '%Starting execution%' LIMIT 1) WHERE name = 'log'
```

To bind that command to a key, like F9, you can run the following `:config` command:

```
:config /ui/keymap-defs/default/f9/command ;UPDATE lnav_views SET selection = (SELECT␣
↪log_line FROM all_logs WHERE log_line > log_msg_line() AND log_body LIKE '%Starting␣
↪execution%' LIMIT 1) WHERE name = 'log'
```

Consult the *Keymaps* configuration section for more information.

CHAPTER

FOUR

COMMAND LINE INTERFACE

There are two command-line interfaces provided by **lnav**, one for viewing files and one for managing **lnav**'s configuration. The file viewing mode is the default and is all that most people will need. The management mode can be useful for those that are developing log file formats and is activated by passing the `-m` option as the first argument.

4.1 File Viewing Mode

The following options can be used when starting **lnav**. There are not many flags because the majority of the functionality is accessed using the `-c` option to execute *commands* or *SQL queries*.

4.1.1 Options

-h

Print these command-line options and exit.

-H

Start lnav and switch to the help view.

-C

Check the given files against the configuration, report any errors, and exit. This option can be helpful for validating that a log format is well-formed.

-S `<relative-time/absolute-local-time>`

Only index log content since the given time. The time can be relative (e.g. "2pm yesterday", "30 min ago") or an absolute time (e.g. "2020-01-02T03:04:05"). If the log content lies completely outside of this time cutoff, the file will be closed out to reduce resource usage.

-U `<relative-time/absolute-local-time>`

Only index log content until the given time. The time can be relative (e.g. "2pm yesterday", "30 min ago") or an absolute time (e.g. "2020-01-02T03:04:05"). If the log content lies completely outside of this time cutoff, the file will be closed out to reduce resource usage.

-c `<command>`

Execute the given lnav command, SQL query, or lnav script. The argument must be prefixed with the character used to enter the prompt to distinguish between the different types (i.e. `:`, `;`, `|`, `/`). This option can be given multiple times.

-f `<path>`

Execute the given command file. This option can be given multiple times.

-e `<command-line>`

Execute the given shell command-line and display its output. This is equivalent to executing the `:sh` command and passing the `-N` flag. This option can be given multiple times.

-I <path>
: Add a configuration directory.

-i
: Install the given files in the lnav configuration directories. Format files, SQL, and lnav scripts will be installed in the `formats/installed`. Configuration files will be installed in the `configs/installed` directory. Git repository URIs will be cloned with a directory name based on their repository URI.

-u
: Update formats installed from git repositories.

-d <path>
: Write debug messages to the given file.

-n
: Run without the curses UI (headless mode).

-N
: Do not open the default syslog file if no files are given.

-r
: Recursively load files from the given base directories.

-t
: For data that was piped into lnav, show the timestamp for each line.

-V
: Print the version of lnav.

-v
: Print extra information during operations.

-q
: Do not print informational messages and behave in a manner compatible with being used as a PAGER.

4.2 Management Mode (v0.11.0+)

The management CLI mode provides functionality for query **lnav**'s log format definitions.

4.2.1 Options

-m
: Switch to management mode. This must be the first option passed on the command-line.

-I <path>
: Add a configuration directory.

4.2.2 Subcommands

apps `create` <name>
: Create a new app with the given name. This command will create the configuration directory, an `app.json` configuration file, and an `app-files` directory with a basic `index.md`. The publisher value will be taken from the `USER` environment variable or the current user name.

config `get`

Print out the current configuration as JSON on the standard output.

config `blame`

Print out the configuration options as JSON-Pointers and the file/line-number where the configuration is sourced from.

config `file-options <path>`

Print out the options that will be applied to the given file. The options are stored in the `file-options.json` file in the **lnav** configuration directory. The only option available at the moment is the timezone to be used for log message timestamps that do not include a zone. The timezone for a file can be set using the *:set-file-timezone* command and cleared with the *:clear-file-timezone* command.

format `<format-name> get`

Print information about the given log format.

format `<format-name> source`

Print the name of the first file that contained this log format definition.

format `<format-name> regex <regex-name> push`

Push a log format regular expression to regex101.com .

format `<format-name> regex <regex-name> pull`

Pull changes to a regex that was previously pushed to regex101.com .

format `<format-name> test <path>`

Test this format against the given file.

piper `clean`

Remove all of the files that stored data that was piped into **lnav** through stdin or a FIFO.

piper `list`

List all of the data that was piped into **lnav** from oldest to newest. The listing will show the creation time, the URL you can use to reopen the data, and a description of the data. Passing the *-v* option will print out additional metadata that was captured, such as the current working directory of **lnav** and the environment variables.

regex101 `import <regex101-url> <format-name> [<regex-name>]`

Convert a regex101.com entry into a skeleton log format file.

4.3 Environment Variables

XDG_CONFIG_HOME

If this variable is set, lnav will use this directory to store its configuration in a sub-directory named `lnav`.

HOME

If *XDG_CONFIG_HOME* is not set, lnav will use this directory to store its configuration in a sub-directory named `.lnav`.

APPDATA

On Windows, lnav will use this directory instead of HOME to store its configuration in a sub-directory named `.lnav`.

TZ

The timezone setting is used in some log formats to convert timestamps with a timezone to the local timezone.

4.4 Examples

To load and follow the system syslog file:

```
lnav
```

To load all of the files in `/var/log`:

```
lnav /var/log
```

To watch the output of make:

```
lnav -e 'make -j4'
```

CHAPTER

FIVE

USAGE

This chapter contains an overview of how to use **lnav**.

5.1 Basic Controls

Like most file viewers, scrolling through files can be done with the usual *hotkeys*. For non-trivial operations, you can enter the *command* prompt by pressing `:`. To analyze data in a log file, you can enter the *SQL prompt* by pressing `;`.

Tip

Check the bottom right corner of the screen for tips on hotkeys that might be useful in the current context.

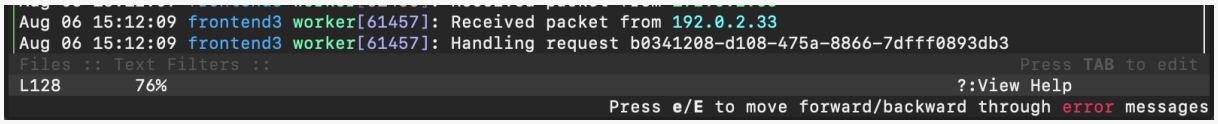

Fig. 1: When **lnav** is first open, it suggests using `e` and `Shift` + `e` to jump to error messages.

5.2 Viewing Files

The files to view in **lnav** can be given on the command-line or passed to the *:open* command. A glob pattern can be given to watch for files with a common name. If the path is a directory, all of the files in the directory will be opened and the directory will be monitored for files to be added or removed from the view. If the path is an archive or compressed file (and lnav was built with libarchive), the archive will be extracted to a temporary location and the files within will be loaded. The files that are found will be scanned to identify their file format. Files that match a log format will be collated by time[1] and displayed in the LOG view. Plain text files can be viewed in the TEXT view, which can be accessed by pressing `t`.

5.2.1 Standard Input

Content piped into **lnav** will be captured and displayed as it comes in. Each line is timestamped as it arrives and is stored as part of the capture. To display the timestamps, pass the `-t` option when invoking lnav. The time elapsed between lines can also be viewed by pressing `Shift` + `T`. For example, if you wanted a basic measurement of what is taking a long time in a run of `make`, you can pipe the output into lnav and enable the elapsed time column.

By default, the captured output is saved in the work directory and a command is printed out to reopen the content. You can get a listing of all captures by running the following *management* command:

[1] Log message timestamps are stored in the index with a resolution of microseconds.

```
lnav -m piper list
```

Captures that are older than 48 hours will be deleted on the next lnav invocation or you can run the following command to clean them up immediately:

```
lnav -m piper clean
```

In order to limit the amount of data captured, lnav will rotate the underlying files as needed. To capture more/less data, adjust the options detailed in the *piper configuration* section.

5.2.2 Archive Support

If **lnav** is compiled with libarchive, any files to be opened will be examined to see if they are a supported archive type. If so, the contents of the archive will be extracted to the `$TMPDIR/lnav-user-${UID}-work/archives/` directory. Once extracted, the files within will be loaded into lnav. To speed up opening large amounts of files, any file that meets the following conditions will be automatically hidden and not indexed:

- Binary files
- Plain text files that are larger than 128KB
- Duplicate log files

The unpacked files will be left in the temporary directory after exiting **lnav** so that opening the same archive again will be faster. Unpacked archives that have not been accessed in the past two days will be automatically deleted the next time **lnav** is started.

5.2.3 Remote Files

Files on remote machines can be viewed and tailed if you have access to the machines via SSH. First, make sure you can SSH into the remote machine without any interaction by: 1) accepting the host key as known and 2) copying your identity's public key to the `.ssh/authorized_keys` file on the remote machine. Once the setup is complete, you can open a file on a remote host using the same syntax as **`scp(1)`** where the username and host are given, followed by a colon, and then the path to the files, like so:

```
[user@]host:/path/to/logs
```

For example, to open `/var/log/syslog.log` on "host1.example.com" as the user "dean", you would write:

```
lnav dean@host1.example.com:/var/log/syslog.log
```

Remote files can also be opened using the *:open* command. Opening a remote file in the TUI has the advantage that the file path can be `TAB`-completed and a preview is shown of the first few lines of the file.

Note

If lnav is installed from the snap, you will need to connect it to the ssh-keys plug using the following command:

```
sudo snap connect lnav:ssh-keys
```

Note

Remote file access is implemented through a "tailer" executable that is transferred to the host. The tailer handles requests from lnav for synchronizing file blocks, completing path names, previewing files, and so on.

There are two implementations of the tailer: Python3 and an cμlly prbl xcμbl. If the target host does not have Python3 installed, the APE binary will be used. An APE binary can run on most any x86_64 machine and OS (i.e. MacOS, Linux, FreeBSD, Windows). The implementations are baked into the lnav executable itself. So, there is no extra setup that needs to be done on the remote machine.

The tailer is copied to the user's home directory and is named `tailer.bin.XXXXXX` where *XXXXXX* is 6 random digits. Under normal circumstances, the file should be deleted immediately.

5.2.4 Command Output

The output of commands can be captured and displayed in **lnav** using the *:sh* command or by passing the *-e* option on the command-line. The captured output will be displayed in the TEXT view. The lines from stdout and stderr are recorded separately so that the lines from stderr can be shown in the theme's "error" highlight. The time that the lines were received are also recorded internally so that the "time-offset" display (enabled by pressing `Shift + T`) can be shown and the "jump to slow-down" hotkeys (`s` / `Shift + S`) work. Since the line-by-line timestamps are recorded internally, they will not interfere with timestamps that are in the commands output.

5.2.5 Docker Logs

To make it easier to view docker logs within **lnav**, a `docker://` URL scheme is available. Passing the container name in the authority field will run the `docker logs` command. If a path is added to the URL, then **lnav** will execute `docker exec <container> tail -F -n +0 /path/to/file` to try and tail the file in the container.

5.2.6 strace

The default output of `strace` does not have enough information for lnav to make use of it. The `-ttt -T` flags need to be passed to add a high-resolution timestamp and the duration of the call. The timestamp is needed for lnav to recognize the content as a log file and the duration is useful for visualizing how long it takes a syscall to run in the TIMELINE view. It is also advisable to use the `-f` flag so all of the threads/sub-processes are traced.

You can use the `strace://localhost/<pid>` URL to run strace on an existing process and have the appropriate flags passed.

5.2.7 Custom URL Schemes

Custom URL schemes can be defined using the */tuning/url-schemes* configuration. By adding a scheme name to the tuning configuration along with the name of an **lnav** handler script, you can control how the URL is interpreted and turned into **lnav** commands. This feature is how the **`Docker Logs`_** functionality is implemented.

Custom URLs can be passed on the command-line or to the *:open* command. When passed on the command-line, an `:open` command with the URL is added to the list of initial commands. When the `:open` command detects a custom URL, it checks for the definition in the configuration. If found, it will call the associated handler script with the URL as the first parameter. The script can parse the URL using the *parse_url(url)* SQL function, if needed. The script should then execute whatever commands it needs to open the destination for viewing in **lnav**. For example, the docker URL handler uses the *:sh* command to run `docker logs` with the container.

5.2.8 Using as a PAGER

Setting **lnav** as your `PAGER` can have some advantages, like basic syntax highlighting and discovering sections in a document. For example, when viewing a man page, the current section is displayed in the breadcrumb bar and you can jump to a section with the *:goto* command.

When setting the `PAGER` variable, pass the *-q* option to modify lnav's behavior to make it more amenable for this purpose:

```
export PAGER="lnav -q"
```

With this option, lnav will:

- Not save the piped content after exiting.
- Print any marked lines to the screen OR all lines if the total height is less than the terminal height.

5.3 Searching

Any log messages that are loaded into **lnav** are indexed by time and log level (e.g. error, warning) to make searching quick and easy with *hotkeys*. For example, pressing `e` will jump to the next error in the file and pressing `Shift + e` will jump to the previous error. Plain text searches can be done by pressing `/` to enter the search prompt. A regular expression can be entered into the prompt to start a search through the current view.

5.4 Filtering

To reduce the amount of noise in a log file, **lnav** can hide log messages that match certain criteria. The following sub-sections explain ways to go about that.

5.4.1 Regular Expression Match

If there are log messages that you are not interested in, you can do a "filter out" to hide messages that match a pattern. A filter can be created using the interactive editor, the *:filter-out* command, or by doing an `INSERT` into the *lnav_view_filters* table.

If there are log messages that you are only interested in, you can do a "filter in" to only show messages that match a pattern. The filter can be created using the interactive editor, the *:filter-in* command, or by doing an `INSERT` into the *lnav_view_filters* table.

5.4.2 SQLite Expression

Complex filtering can be done by passing a SQLite expression to the *:filter-expr* command. The expression will be executed for every log message and if it returns true, the line will be shown in the log view.

5.4.3 Time

To limit log messages to a given time frame, the *:hide-lines-before* and *:hide-lines-after* commands can be used to specify the beginning and end of the time frame.

5.4.4 Log level

To hide messages below a certain log level, you can use the *:set-min-log-level* command.

5.5 Log-oriented Debugging

5.5.1 Breakpoints

Breakpoints let you mark log messages that originate from a particular source file location (e.g. `main.cc:42`) or that share a common message schema. Log messages that have a source location and match a breakpoint will be highlighted with a red circle, making it easy to spot related messages as you scroll through logs.

To toggle a breakpoint on the focused log message, press `Ctrl + B` or use the *:breakpoint* command. You can also set breakpoints explicitly with the *:breakpoint* command, specifying a `[format:]file:line` location.

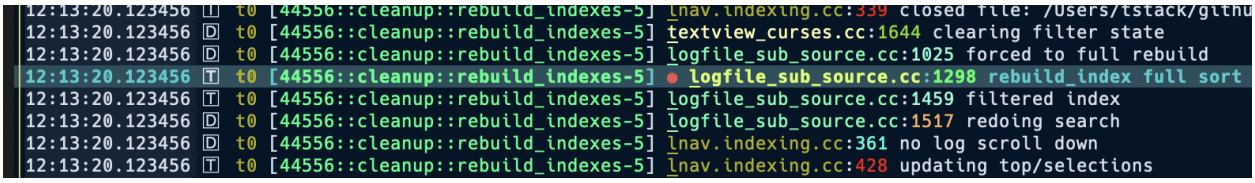

Fig. 2: A screenshot of lnav's logs where a breakpoint is set. The red circle next to the source file name and line indicates the location of the breakpoint.

Once breakpoints are set, you can jump between matching log messages using the `F7` and `F8` keys to move to the previous and next breakpoint hit, respectively.

Breakpoints can be disabled without deleting them using the *:toggle-breakpoint* command. The *lnav_log_breakpoints* SQL table, which allows you to query, insert, update, and delete breakpoints. For example, to disable all breakpoints matching a file:

```
;UPDATE lnav_log_breakpoints SET enabled = 0 WHERE description LIKE '%main.cc%'
```

Breakpoints are saved as part of the session, so they will be restored the next time you open the same files.

5.6 Search Tables

Search tables allow you to access arbitrary data in log messages through SQLite virtual tables. If there is some data in a log message that you can match with a regular expression, you can create a search-table that matches that data and any capture groups will be plumbed through as columns in the search table.

Creating a search table can be done interactively using the *:create-search-table* command or by adding it to a *log format definition*. The main difference between the two is that tables defined as part of a format will only search messages from log files with that format and the tables will include log message columns defined in that format. Whereas a table created with the command will search messages from all different formats and no format-specific columns will be included in the table.

5.7 Taking Notes

As you are looking through logs, you might find that you want to leave some notes of your findings. **lnav** can help here by saving information in the session without needing to modify the actual log files. Thus, when you re-open the files in lnav, the notes will be restored. The following types of information can be saved:

tags
: Log messages can be tagged with the *:tag* command as a simple way to leave a descriptive mark. The tags attached to a message will be shown underneath the message. You can press `u` and `Shift` + `u` to jump to the next/previous marked line. A regular search will also match tags.

comments
: Free-form text can be attached to a log message with the *:comment* command. The comment will be shown underneath the message. If the text contains Markdown syntax, it will be rendered to the best of the terminal's ability. The comment can reference other log messages using Markdown links with the target being the permalink of the other log message. You can then click the link to jump to the other message and then press `Ctrl` + `O` to jump back. You can press `u` and `Shift` + `u` to jump to the next/previous marked line. A regular search will also match the comment text.

partitions
: The LOG view can be partitioned to provide some context about where you are in a collection of logs. For example, in logs for a test run, partitions could be created with the name for each test. The

current partition is shown in the breadcrumb bar and prefixed by the "" symbol. You can select the partition breadcrumb to jump to another partition. Pressing { and } will jump to the next/previous partition.

5.7.1 Accessing notes through the SQLite interface

The note taking functionality in lnav can also be accessed through the log tables exposed through SQLite. The majority of the columns in a log table are read-only since they are backed by the log files themselves. However, the following columns can be changed by an `UPDATE` statement:

- **log_part** - The "partition" the log message belongs to. This column can also be changed by the *:partition-name* command.
- **log_mark** - Indicates whether the line has been bookmarked.
- **log_comment** - A free-form text field for storing commentary. This column can also be changed by the *:comment* command.
- **log_tags** - A JSON list of tags associated with the log message. This column can also be changed by the *:tag* command.

While these columns can be updated by through other means, using the SQL interface allows you to make changes automatically and en masse. For example, to bookmark all lines that have the text "something interesting" in the log message body, you can execute:

```
;UPDATE all_logs SET log_mark = 1 WHERE log_body LIKE '%something interesting%'
```

As a more advanced example of the power afforded by SQL and **lnav**'s virtual tables, we will tag log messages where the IP address bound by dhclient has changed. For example, if dhclient reports "bound to 10.0.0.1" initially and then reports "bound to 10.0.0.2", we want to tag only the messages where the IP address was different from the previous message. While this can be done with a single SQL statement[2], we will break things down into a few steps for this example. First, we will use the *:create-search-table* command to match the dhclient message and extract the IP address:

```
:create-search-table dhclient_ip bound to (?<ip>[^ ]+)
```

The above command will create a new table named `dhclient_ip` with the standard log columns and an `ip` column that contains the IP address. Next, we will create a view over the `dhclient_ip` table that returns the log message line number, the IP address from the current row and the IP address from the previous row:

```
;CREATE VIEW IF NOT EXISTS dhclient_ip_changes AS SELECT log_line, ip, lag(ip) OVER␣
↪(ORDER BY log_line) AS prev_ip FROM dhclient_ip
```

Finally, the following `UPDATE` statement will concatenate the tag "#ipchanged" onto the `log_tags` column for any rows in the view where the current IP is different from the previous IP:

```
;UPDATE syslog_log SET log_tags = json_concat(log_tags, '#ipchanged') WHERE log_line IN␣
↪(SELECT log_line FROM dhclient_ip_changes WHERE ip != prev_ip)
```

Since the above can be a lot to type out interactively, you can put these commands into a *script* and execute that script with the | hotkey.

[2] The expression `regexp_match('bound to ([^ ]+)', log_body) as ip` can be used to extract the IP address from the log message body.

5.8 Sharing Sessions With Others

After setting up filters, bookmarks, and making notes, you might want to share your work with others. If they have access to the same log files, you can use the *:export-session-to* command to write an executable **lnav** script that will recreate the current session state. The script contains various SQL statements and **lnav** commands that capture the current state. So, you should feel free to modify the script or use it as a reference to learn about more advanced uses of lnav.

The script will capture the file paths that were explicitly specified and not the files that were actually opened. For example, if you specified "/var/log" on the command line, the script will include `:open /var/log/*` and not an individual open for each file in that directory.

Also, in order to support archives of log files, lnav will try to find the directory where the archive was unpacked and use that as the base for the `:open` command. Currently, this is done by searching for the top "README" file in the directory hierarchy containing the files[3]. The consumer of the session script can then set the `LOG_DIR_0` (or 1, 2, ...) environment variable to change where the log files will be loaded from.

[3] It is assumed a log archive would have a descriptive README file. Other heuristics may be added in the future.

CHAPTER

SIX

SCRIPTS

Commands and SQL queries can be combined into scripts to automate tasks, like finding log messages or creating reports. For example, **lnav** includes a `partition-by-boot` script that partitions the log view based on boot messages from the Linux kernel. A script can have a mix of SQL and **lnav** commands, as well as include other scripts. The type of statement to execute is determined by the leading character on a line: a semi-colon begins a SQL statement; a colon starts an **lnav** command; and a pipe `|` denotes another script to be executed. Lines beginning with a hash are treated as comments. Script files should be named with a `.lnav` extension.

Scripts can be located anywhere in the file system and executed by giving their full path in the `|` prompt. Or, scripts can be *installed* in format directories so they can be called by just their base name. You can also create `.sql` files that contain SQL statements to be executed on startup. A good use for SQL files is creating a helper `TABLE` or `VIEW` to be used for other queries.

The following variables are defined in a script:

#
: The number of arguments passed to the script.

__all__
: A string containing all the arguments joined by a single space.

0
: The path to the script being executed.

1-N
: The arguments passed to the script.

LNAV_HOME_DIR
: The path to the directory where the user's **lnav** configuration is stored.

LNAV_WORK_DIR
: The path to the directory where **lnav** caches files, like archives that have been unpacked or piper captures.

Remember that you need to use the *:eval* command when referencing variables in most **lnav** commands. Scripts can provide help text to be displayed during interactive usage by adding the following tags in a comment header:

@synopsis
: The synopsis should contain the name of the script and any parameters to be passed. For example:

```
# @synopsis: hello-world <name1> [<name2> ... <nameN>]
```

@description
: A one-line description of what the script does. For example:

```
# @description: Say hello to the given names.
```

@output-format
: The MIME type of the output generated by the script. When set to `text/markdown` the *:write-table-to* will generate Markdown tables.

Tip

The *:eval* command can be used to do variable substitution for commands that do not natively support it. For example, to substitute the variable, `pattern`, in a *:filter-out* command:

```
:eval :filter-out ${pattern}
```

6.1 VSCode Extension

The lnav VSCode Extension can be installed to add syntax highlighting to lnav scripts.

6.2 Builtin Scripts

The following scripts are built into lnav.

6.2.1 find-msg

This script can help in finding log messages that are related to the focused message. lnav already provides hotkeys for finding messages based on common fields, but this script can be used for arbitrary fields.

find-msg source

6.2.2 find-chained-msg

This script is similar to `find-msg`, except instead of matching a single field in the focused and next/previous message, the matched value can be in separate fields.

find-chained-msg source

6.2.3 report-access-log

This script generates a report from the `access_log` table that is similar to the output of the goaccess tool.

report-access-log souurce

CHAPTER

SEVEN

COOKBOOK

This chapter contains recipes for common tasks that can be done in **lnav**. These recipes can be used as a starting point for your own needs after some adaptation.

7.1 Custom Web Access Log Highlights

To highlight log message fields with custom colors based on their values, you can install a file that patches the format with the desired highlights. The following patch updates the `access_log` format. A `highlights` entry is added to the `value/sc_status` object that applies a color for codes of interest.

Listing 1: access_log_highlights.json

```
{
    "$schema": "https://lnav.org/schemas/format-v1.schema.json",
    "access_log": {
        "value": {
            "sc_status": {
                "highlights": {
                    "ok": {
                        "pattern": "^200$",
                        "color": "green"
                    },
                    "forbidden": {
                        "pattern": "^403$",
                        "color": "#af0000"
                    },
                    "redirect": {
                        "pattern": "^(?:301|302)$",
                        "color": "#00afff"
                    }
                }
            }
        }
    }
}
```

7.2 Annotating Logs

Log messages can be annotated in a couple of different ways in **lnav** to help you get organized.

7.2.1 Create partitions for Linux boots

When digging through logs that can be broken up into multiple sections, **lnav**'s *partitioning feature* can be used to keep track of which section you are in. For example, if a collection of Linux logs covered multiple boots, the following script could be used to create partitions for each boot. After the partition name is set for the log messages, the current name will show up in the top status bar next to the current time.

Listing 2: partition-by-boot.lnav

```
#
# DO NOT EDIT THIS FILE, IT WILL BE OVERWRITTEN!
#
# @synopsis: partition-by-boot
# @description: Partition the log view based on boot messages from the Linux kernel.
#

;UPDATE syslog_log
     SET log_part = 'Boot: ' || log_time
     WHERE log_text LIKE '%kernel:%Linux version%';

;SELECT 'Created ' || changes() || ' partitions(s)';
```

7.2.2 Tagging SSH log messages

Log messages can be tagged interactively with the *:tag* command or programmatically using the *SQLite Interface*. This example uses a script to search for interesting SSH messages and automatically adds an appropriate tag.

Listing 3: tag-ssh-msgs.lnav

```
#
# @synopsis: tag-ssh-msgs
# @description: Tag interesting SSH log messages
#

;UPDATE all_logs
     SET log_tags = json_concat(log_tags, '#ssh.invalid-user')
     WHERE log_text LIKE '%Invalid user from%'

;SELECT 'Tagged ' || changes() || ' messages';
```

7.3 Log Analysis

Most log analysis within **lnav** is done through the *SQLite Interface*. The following examples should give you some ideas to start leveraging this functionality. One thing to keep in mind is that if a query gets to be too large or multiple statements need to be executed, you can create a `.lnav` script that contains the statements and execute it using the `|` command prompt.

7.3.1 Count client IPs in web access logs

To count the occurrences of an IP in web access logs and order the results from highest to lowest:

```
;SELECT c_ip, count(*) as hits FROM access_log GROUP BY c_ip ORDER BY hits DESC
```

7.3.2 Show only lines where a numeric field is in a range

The *:filter-expr* command can be used to filter web access logs to only show lines where the number of bytes transferred to the client is between 10,000 and 40,000 bytes like so:

```
:filter-expr :sc_bytes BETWEEN 10000 AND 40000
```

7.3.3 Generating a Report

Reports can be generated by writing an **lnav** *script* that uses SQL queries and commands to format a document. A basic script can simply execute a SQL query that is shown in the DB view. More sophisticated scripts can use the following commands to generate customized output for a report:

- The *:echo* command to write plain text
- *SQL queries* followed by a "write" command, like *:write-table-to*.

Listing 4: report-demo.lnav

```
#
# @synopsis: report-demo [<output-path>]
# @description: Generate a report for requests in access_log files
#

# Figure out the file path where the report should be written to, default is
# stdout
;SELECT CASE
          WHEN $1 IS NULL THEN '-'
          ELSE $1
        END AS out_path

# Redirect output from commands to $out_path
:redirect-to $out_path

# Print an introductory message
;SELECT printf('\n%d total requests', count(1)) AS msg FROM access_log
:echo $msg

;WITH top_paths AS MATERIALIZED (
        SELECT
            cs_uri_stem,
            count(1) AS total_hits,
            sum(sc_bytes) as bytes,
            count(distinct c_ip) as visitors
          FROM access_log
          WHERE sc_status BETWEEN 200 AND 300
          GROUP BY cs_uri_stem
          ORDER BY total_hits DESC
          LIMIT 50),
      weekly_hits_with_gaps AS MATERIALIZED (
        SELECT timeslice(log_time_msecs, '1w') AS week,
               cs_uri_stem,
               count(1) AS weekly_hits
          FROM access_log
          WHERE cs_uri_stem IN (SELECT cs_uri_stem FROM top_paths) AND
```

(continues on next page)

(continued from previous page)

```
                sc_status BETWEEN 200 AND 300
          GROUP BY week, cs_uri_stem),
      all_weeks AS (
        SELECT week
          FROM weekly_hits_with_gaps
          GROUP BY week
          ORDER BY week ASC),
      weekly_hits AS (
        SELECT all_weeks.week,
               top_paths.cs_uri_stem,
               ifnull(weekly_hits, 0) AS hits
          FROM all_weeks
          CROSS JOIN top_paths
          LEFT JOIN weekly_hits_with_gaps
            ON all_weeks.week = weekly_hits_with_gaps.week AND
               top_paths.cs_uri_stem = weekly_hits_with_gaps.cs_uri_stem)
  SELECT weekly_hits.cs_uri_stem AS Path,
         printf('%,9d', total_hits) AS Hits,
         printf('%,9d', visitors) AS Visitors,
         printf('%9s', humanize_file_size(bytes)) as Amount,
         sparkline(hits) AS Weeks
    FROM weekly_hits
    LEFT JOIN top_paths ON top_paths.cs_uri_stem = weekly_hits.cs_uri_stem
    GROUP BY weekly_hits.cs_uri_stem
    ORDER BY Hits DESC
    LIMIT 10

:write-table-to -

:echo
:echo Failed Requests
:echo

;SELECT printf('%,9d', count(1)) AS Hits,
        printf('%,9d', count(distinct c_ip)) AS Visitors,
        sc_status AS Status,
        cs_method AS Method,
        group_concat(distinct cs_version) AS Versions,
        cs_uri_stem AS Path,
        replicate('|', (cast(count(1) AS REAL) / $total_requests) * 100.0) AS "% of␣
↪Requests"
    FROM access_log
    WHERE sc_status >= 400
    GROUP BY cs_method, cs_uri_stem
    ORDER BY Hits DESC
    LIMIT 10

:write-table-to -
```

7.3.4 Creating a Runbook

If you have a collection of lnav scripts, you can create a markdown file that links to the scripts and use it as a runbook. Clicking on a link to a `.lnav` file will open an "Actions" menu with an "Execute" menu item. Clicking on "Execute" will launch the script.

CHAPTER

EIGHT

CONFIGURATION

The configuration for **lnav** is stored in JSON files that are read on startup. Files are consumed from multiple locations and update the internal configuration state as they are processed. File names must end with `.json` if they are in a `configs` directory or, if they are in a `formats` directory[1], start with `config.` and end with `.json`. The following enumerates the order in which files are processed (where <lnav-home> refers to the location in the *HOME* directory where files are stored, either: `~/.lnav` or `~/.config/lnav`):

1. Builtin – The default configuration is shipped inside the **lnav** binary.
2. `/etc/lnav/configs/*/*.json`, `/etc/lnav/formats/*/config.*.json` – System-wide configuration files can be installed in these locations to make it available to all users.
3. <lnav-home>`/configs/default/*.json` – The default configuration files that are built into lnav are written to this directory with `.sample` appended. Removing the `.sample` extension and editing the file will allow you to do basic customizations.
4. <lnav-home>`/configs/*/*.json`, <lnav-home>`/formats/*/config.*.json` – Other directories under home will be scanned for JSON files. This structure is convenient for installing **lnav** configurations, like from a git repository. The `configs/installed` directory is reserved for files that are installed using the *-i* flag (e.g. `$ lnav -i /path/to/config.json`).
5. `-I` <path>`/configs/*/*.json`, `-I` <path>`/formats/*/*.json` – Include directories passed on the command-line can have a `configs` directory that will also be searched.
6. <lnav-home>`/config.json` – Contains local customizations that were done using the `:config` command.

A valid **lnav** configuration file must contain an object with the `$schema` property, like so:

```
{
    "$schema": "https://lnav.org/schemas/config-v1.schema.json"
}
```

Note

Log format definitions are stored separately in the `~/.lnav/formats` directly. See the *Log Formats* chapter for more information.

Note

[1] Putting log format files and configuration files in the same directory can be convenient for organizations that wish to have a single git repository that contains all of the files needed for folks to do their work.

Configuration files are read in the above directory order and sorted by path name. The internal configuration is updated as files are parsed, so one file can overwrite the settings from another. You can use the *Management CLI* to get the final configuration and where the value came from for a particular configuration option.

8.1 Options

The following configuration options can be used to customize the **lnav** UI to your liking. The options can be changed using the `:config` command.

8.1.1 /ui/keymap

The name of the keymap to use.	
type	*string*

8.1.2 /ui/theme

The name of the theme to use.	
type	*string*

8.1.3 /ui/clock-format

The format for the clock displayed in the top-left corner using strftime(3) conversions	
type	*string*
examples	%a %b %d %H:%M:%S %Z

8.1.4 /ui/dim-text

Reduce the brightness of text (useful for xterms). This setting can be useful when running in an xterm where the white color is very bright.	
type	*boolean*

8.1.5 /ui/default-colors

Use default terminal background and foreground colors instead of black and white for all text coloring. This setting can be useful when transparent background or alternate color theme terminal is used.	
type	*boolean*

8.1.6 /ui/views/log/time-column

Display a column with the log message time and hide the timestamp/level in the message. Possible values: disabled - never display the column; enabled - display the column when initially scrolling right; default - display the column initially.	
type	*string*
enum	disabled, enabled, default
examples	enabled

8.1.7 /tuning/external-editor

Settings related to opening content in an external editor				
type	*object*			
properties				
• impls	*/tuning/external-editor/impls*			
	External editor implementations			
	type	*object*		
	patternProperties			
	• ^([\w\-\.]+)$	*/tuning/external-editor/impls/<editor_impl_name>*		
		External editor implementation		
		type	*object*	
		properties		
		• test	*/tuning/external-editor/impls/<editor_impl_name>/test*	
			The command that checks if an external editor is available	
			type	*string*
			examples	command -v open
		• command	*/tuning/external-editor/impls/<editor_impl_name>/command*	
			The command used to open text for editing	
			type	*string*
			examples	code -
		• config-dir	*/tuning/external-editor/impls/<editor_impl_name>/config-dir*	
			The name of the directory where editor configuration is stored	
			type	*string*
			examples	.idea
		• prefers	*/tuning/external-editor/impls/<editor_impl_name>/prefers*	
			type	*string*
			examples	^.*(?:\.cpp)$
		additionalProperties	False	
	additionalProperties	False		
additionalProperties	False			

8.1.8 /tuning/external-opener

Settings related to opening external files/URLs				
type	*object*			
properties				
• impls	*/tuning/external-opener/impls*			
	External opener implementations			
	type	*object*		
	patternProperties			
	• ^([\w\-\.]+)$	*/tuning/external-opener/impls/<opener_impl_name>*		
		External opener implementation		
		type	*object*	
		properties		
		• test	*/tuning/external-opener/impls/<opener_impl_name>/test*	
			The command that checks if an external opener is available	
			type	*string*
			examples	command -v open
		• command	*/tuning/external-opener/impls/<opener_impl_name>/command*	
			The command used to open a file or URL	
			type	*string*
			examples	open
		additionalProperties	False	
	additionalProperties	False		
additionalProperties	False			

8.2 Theme Definitions

User Interface themes are defined in a JSON configuration file. A theme is made up of the style definitions for different types of text in the UI. A *definition* can include the foreground/background colors and the bold/underline attributes. The style definitions are broken up into multiple categories for the sake of organization. To make it easier to write a definition, a theme can define variables that can be referenced as color values.

8.2.1 Variables

The `vars` object in a theme definition contains the mapping of variable names to color values. These variables can be referenced in style definitions by prefixing them with a dollar-sign (e.g. `$black`). The following variables can also be defined to control the values of the ANSI colors that are log messages or plain text:

Table 1: ANSI colors

Variable Name	ANSI Escape
black	ESC[30m
red	ESC[31m
green	ESC[32m
yellow	ESC[33m
blue	ESC[34m
magenta	ESC[35m
cyan	ESC[36m
white	ESC[37m

8.2.2 Specifying Colors

Colors can be specified using hexadecimal notation by starting with a hash (e.g. `#aabbcc`) or using a color name as found at http://jonasjacek.github.io/colors/. If colors are not specified for a style, the values from the `styles/text` definition.

Note

When specifying colors in hexadecimal notation, you do not need to have an exact match in the XTerm 256 color palette. A best approximation will be picked based on the CIEDE2000 color difference algorithm.

8.2.3 Example

The following example sets the black/background color for text to a dark grey using a variable and sets the foreground to an off-white. This theme is incomplete, but it works enough to give you an idea of how a theme is defined. You can copy the code block, save it to a file in `~/.lnav/configs/installed/` and then activate it by executing `:config /ui/theme example` in lnav. For a more complete theme definition, see one of the definitions built into **lnav**, like monocai.

```
{
    "$schema": "https://lnav.org/schemas/config-v1.schema.json",
    "ui": {
        "theme-defs": {
            "example1": {
                "vars": {
                    "black": "#2d2a2e"
                },
                "styles": {
                    "text": {
                        "color": "#f6f6f6",
                        "background-color": "$black"
                    }
                }
            }
        }
    }
}
```

8.2.4 Reference

/ui/theme-defs/<theme_name>/vars

Variables definitions that are used in this theme.	
type	*object*
patternProperties	
• ^(\w+)$	*/ui/theme-defs/<theme_name>/vars/<var_name>*
	A theme variable definition
	type *boolean / integer / string*
additionalProperties	False

/ui/theme-defs/<theme_name>/styles

Styles for log messages.	
type	*object*
properties	
• identifier	*/ui/theme-defs/<theme_name>/styles/identifier*
	Styling for identifiers in logs
	style
• text	*/ui/theme-defs/<theme_name>/styles/text*
	Styling for plain text
	style
• selected-text	*/ui/theme-defs/<theme_name>/styles/selected-text*
	Styling for text selected in a view
	style
• fuzzy-match	*/ui/theme-defs/<theme_name>/styles/fuzzy-match*
	Styling for characters found in fuzzy match
	style
• alt-text	*/ui/theme-defs/<theme_name>/styles/alt-text*
	Styling for plain text when alternating
	style
• error	*/ui/theme-defs/<theme_name>/styles/error*
	Styling for error messages
	style
• ok	*/ui/theme-defs/<theme_name>/styles/ok*
	Styling for success messages
	style
• info	*/ui/theme-defs/<theme_name>/styles/info*
	Styling for informational messages
	style
• warning	*/ui/theme-defs/<theme_name>/styles/warning*
	Styling for warning messages
	style
• hidden	*/ui/theme-defs/<theme_name>/styles/hidden*
	Styling for hidden fields in logs
	style
• cursor-line	*/ui/theme-defs/<theme_name>/styles/cursor-line*
	Styling for the cursor line in the main view
	style
• disabled-cursor-line	*/ui/theme-defs/<theme_name>/styles/disabled-cursor-line*

continues on next page

Table 2 – continued from previous page

	Styling for the cursor line when it is disabled
	style
• adjusted-time	*/ui/theme-defs/<theme_name>/styles/adjusted-time*
	Styling for timestamps that have been adjusted
	style
• skewed-time	*/ui/theme-defs/<theme_name>/styles/skewed-time*
	Styling for timestamps that are different from the received time
	style
• file-offset	*/ui/theme-defs/<theme_name>/styles/file-offset*
	Styling for a file offset
	style
• offset-time	*/ui/theme-defs/<theme_name>/styles/offset-time*
	Styling for the elapsed time column
	style
• time-column	*/ui/theme-defs/<theme_name>/styles/time-column*
	Styling for the time column
	style
• invalid-msg	*/ui/theme-defs/<theme_name>/styles/invalid-msg*
	Styling for invalid log messages
	style
• popup	*/ui/theme-defs/<theme_name>/styles/popup*
	Styling for popup windows
	style
• popup-border	*/ui/theme-defs/<theme_name>/styles/popup-border*
	Styling for the borders of a popup window
	style
• focused	*/ui/theme-defs/<theme_name>/styles/focused*
	Styling for a focused row in a list view
	style
• disabled-focused	*/ui/theme-defs/<theme_name>/styles/disabled-focused*
	Styling for a disabled focused row in a list view
	style
• scrollbar	*/ui/theme-defs/<theme_name>/styles/scrollbar*
	Styling for scrollbars
	style
• h1	*/ui/theme-defs/<theme_name>/styles/h1*
	Styling for top-level headers
	style
• h2	*/ui/theme-defs/<theme_name>/styles/h2*
	Styling for 2nd-level headers
	style
• h3	*/ui/theme-defs/<theme_name>/styles/h3*
	Styling for 3rd-level headers
	style
• h4	*/ui/theme-defs/<theme_name>/styles/h4*
	Styling for 4th-level headers
	style
• h5	*/ui/theme-defs/<theme_name>/styles/h5*
	Styling for 5th-level headers
	style
• h6	*/ui/theme-defs/<theme_name>/styles/h6*

continues on next page

Table 2 – continued from previous page

	Styling for 6th-level headers
	style
• hr	*/ui/theme-defs/<theme_name>/styles/hr*
	Styling for horizontal rules
	style
• hyperlink	*/ui/theme-defs/<theme_name>/styles/hyperlink*
	Styling for hyperlinks
	style
• list-glyph	*/ui/theme-defs/<theme_name>/styles/list-glyph*
	Styling for glyphs that prefix a list item
	style
• breadcrumb	*/ui/theme-defs/<theme_name>/styles/breadcrumb*
	Styling for the separator between breadcrumbs
	style
• table-border	*/ui/theme-defs/<theme_name>/styles/table-border*
	Styling for table borders
	style
• table-header	*/ui/theme-defs/<theme_name>/styles/table-header*
	Styling for table headers
	style
• quote-border	*/ui/theme-defs/<theme_name>/styles/quote-border*
	Styling for quoted-block borders
	style
• quoted-text	*/ui/theme-defs/<theme_name>/styles/quoted-text*
	Styling for quoted text blocks
	style
• footnote-border	*/ui/theme-defs/<theme_name>/styles/footnote-border*
	Styling for footnote borders
	style
• footnote-text	*/ui/theme-defs/<theme_name>/styles/footnote-text*
	Styling for footnote text
	style
• snippet-border	*/ui/theme-defs/<theme_name>/styles/snippet-border*
	Styling for snippet borders
	style
• indent-guide	*/ui/theme-defs/<theme_name>/styles/indent-guide*
	Styling for indent guide lines
	style
• timeline-bar	*/ui/theme-defs/<theme_name>/styles/timeline-bar*
	Styling for timeline duration bars
	style
additionalProperties	False

/ui/theme-defs/<theme_name>/syntax-styles

Styles for syntax highlighting in text files.	
type	*object*
properties	
• inline-code	*/ui/theme-defs/<theme_name>/syntax-styles/inline-code*
	Styling for inline code blocks

continues on next page

Table 3 – continued from previous page

	style
• quoted-code	*/ui/theme-defs/<theme_name>/syntax-styles/quoted-code*
	Styling for quoted code blocks
	style
• code-border	*/ui/theme-defs/<theme_name>/syntax-styles/code-border*
	Styling for quoted-code borders
	style
• object-key	*/ui/theme-defs/<theme_name>/syntax-styles/object-key*
	Styling for a key in an object
	style
• keyword	*/ui/theme-defs/<theme_name>/syntax-styles/keyword*
	Styling for keywords in source files
	style
• string	*/ui/theme-defs/<theme_name>/syntax-styles/string*
	Styling for single/double-quoted strings in text
	style
• comment	*/ui/theme-defs/<theme_name>/syntax-styles/comment*
	Styling for comments in source files
	style
• doc-directive	*/ui/theme-defs/<theme_name>/syntax-styles/doc-directive*
	Styling for documentation directives in source files
	style
• variable	*/ui/theme-defs/<theme_name>/syntax-styles/variable*
	Styling for variables in text
	style
• symbol	*/ui/theme-defs/<theme_name>/syntax-styles/symbol*
	Styling for symbols in source files
	style
• null	*/ui/theme-defs/<theme_name>/syntax-styles/null*
	Styling for nulls in source files
	style
• ascii-control	*/ui/theme-defs/<theme_name>/syntax-styles/ascii-control*
	Styling for ASCII control characters in source files
	style
• non-ascii	*/ui/theme-defs/<theme_name>/syntax-styles/non-ascii*
	Styling for non-ASCII characters in source files
	style
• number	*/ui/theme-defs/<theme_name>/syntax-styles/number*
	Styling for numbers in source files
	style
• type	*/ui/theme-defs/<theme_name>/syntax-styles/type*
	Styling for types in source files
	style
• function	*/ui/theme-defs/<theme_name>/syntax-styles/function*
	Styling for functions in source files
	style
• separators-references-accessors	*/ui/theme-defs/<theme_name>/syntax-styles/separators-references-accessors*

continues on next page

Table 3 – continued from previous page

	Styling for sigils in source files
	style
• re-special	*/ui/theme-defs/<theme_name>/syntax-styles/re-special*
	Styling for special characters in regular expressions
	style
• re-repeat	*/ui/theme-defs/<theme_name>/syntax-styles/re-repeat*
	Styling for repeats in regular expressions
	style
• diff-delete	*/ui/theme-defs/<theme_name>/syntax-styles/diff-delete*
	Styling for deleted lines in diffs
	style
• diff-add	*/ui/theme-defs/<theme_name>/syntax-styles/diff-add*
	Styling for added lines in diffs
	style
• diff-section	*/ui/theme-defs/<theme_name>/syntax-styles/diff-section*
	Styling for diffs
	style
• spectrogram-low	*/ui/theme-defs/<theme_name>/syntax-styles/spectrogram-low*
	Styling for the lower threshold values in the spectrogram view
	style
• spectrogram-medium	*/ui/theme-defs/<theme_name>/syntax-styles/spectrogram-medium*
	Styling for the medium threshold values in the spectrogram view
	style
• spectrogram-high	*/ui/theme-defs/<theme_name>/syntax-styles/spectrogram-high*
	Styling for the high threshold values in the spectrogram view
	style
• file	*/ui/theme-defs/<theme_name>/syntax-styles/file*
	Styling for file names in source files
	style
additionalProperties	False

/ui/theme-defs/<theme_name>/status-styles

Styles for the user-interface components.	
type	*object*
properties	
• text	*/ui/theme-defs/<theme_name>/status-styles/text*
	Styling for status bars
	style
• warn	*/ui/theme-defs/<theme_name>/status-styles/warn*
	Styling for warnings in status bars
	style
• alert	*/ui/theme-defs/<theme_name>/status-styles/alert*
	Styling for alerts in status bars

continues on next page

Table 4 – continued from previous page

	style
• active	*/ui/theme-defs/<theme_name>/status-styles/active*
	Styling for activity in status bars
	style
• inactive-warn	*/ui/theme-defs/<theme_name>/status-styles/inactive-warn*
	Styling for inactive warning status bars
	style
• inactive-alert	*/ui/theme-defs/<theme_name>/status-styles/inactive-alert*
	Styling for inactive alert status bars
	style
• inactive	*/ui/theme-defs/<theme_name>/status-styles/inactive*
	Styling for inactive status bars
	style
• title-hotkey	*/ui/theme-defs/<theme_name>/status-styles/title-hotkey*
	Styling for hotkey highlights in titles
	style
• title	*/ui/theme-defs/<theme_name>/status-styles/title*
	Styling for title sections of status bars
	style
• alert-title	*/ui/theme-defs/<theme_name>/status-styles/alert-title*
	Styling for title sections of status bars with alerts
	style
• disabled-title	*/ui/theme-defs/<theme_name>/status-styles/disabled-title*
	Styling for title sections of status bars
	style
• subtitle	*/ui/theme-defs/<theme_name>/status-styles/subtitle*
	Styling for subtitle sections of status bars
	style
• info	*/ui/theme-defs/<theme_name>/status-styles/info*
	Styling for informational messages in status bars
	style
• hotkey	*/ui/theme-defs/<theme_name>/status-styles/hotkey*
	Styling for hotkey highlights of status bars
	style
• suggestion	*/ui/theme-defs/<theme_name>/status-styles/suggestion*
	Styling for suggested values
	style
additionalProperties	False

/ui/theme-defs/<theme_name>/log-level-styles

Styles for each log message level.	
type	*object*
patternProperties	
• ^(trace | debug5 | debug4 | debug3 | debug2 | debug | info | stats | notice | warning | error | critical | fatal | invalid)$	*/ui/theme-defs/<theme_name>/log-level-styles/<level>*
	style
additionalProperties	False

style

type	*object*	
properties		
• text-align	*/text-align*	
	How to align text within a cell	
	type	*string*
	enum	start, center, end
• color	*/color*	
	The foreground color value for this style. The value can be the name of an xterm color, the hexa theme variable reference.	
	type	*string*
	examples	#fff
		Green
		$black
• background-color	*/background-color*	
	The background color value for this style. The value can be the name of an xterm color, the hex a theme variable reference.	
	type	*string*
	examples	#2d2a2e
		Green
• underline	*/underline*	
	Indicates that the text should be underlined.	
	type	*boolean*
• bold	*/bold*	
	Indicates that the text should be bolded.	
	type	*boolean*
• italic	*/italic*	
	Indicates that the text should be italicized.	
	type	*boolean*
• strike	*/strike*	
	Indicates that the text should be struck.	
	type	*boolean*
• nestable	*/nestable*	
	This highlight can be nested in another highlight.	
	type	*boolean*
additionalProperties	False	

8.3 Keymap Definitions

Keymaps in **lnav** map a key sequence to a command to execute. When a key is pressed, it is converted into a hex-encoded string that is looked up in the keymap. The `command` value associated with the entry in the keymap is then executed. Note that the "command" can be an **lnav** *command*, a *SQL statement/query*, or an **lnav** script. If an `alt-msg` value is included in the entry, the bottom-right section of the UI will be updated with the help text. See the default keymap for an example of how many of the *default hotkeys* are implemented.

> **Note**
>
> Not all functionality is available via commands or SQL at the moment. Also, some hotkeys are not implemented via keymaps.

8.3.1 Key Sequence Encoding

Key presses are converted into a string that is used to lookup an entry in the keymap. Function keys are encoded as an `f` followed by the key number. Other keys are encoded as UTF-8 bytes and formatted as an `x` followed by the hex-encoding in lowercase. For example, the encoding for the `£` key would be `xc2xa3`. To make it easier to discover the encoding for unassigned keys, **lnav** will print in the command prompt the `:config` command and JSON-Pointer for assigning a command to the key.

Fig. 1: Screenshot of the command prompt when an unassigned key is pressed.

> **Note**
>
> Since **lnav** is a terminal application, it can only receive keypresses that can be represented as characters or escape sequences. For example, it cannot handle the press of a modifier key.

8.3.2 Reference

/ui/keymap-defs/<keymap_name>

The keymap definitions	
type	*object*
patternProperties	
• ^((?:x[0-9a-f]{2}|f[0-9]{1,2})+	*/ui/keymap-defs/<keymap_name>/<key_seq>* Map of key codes to commands to execute. The field names are the keys to be mapped using as a hexadecimal representation of the UTF-8 encoding. Each byte of the UTF-8 should start with an 'x' followed by the hexadecimal representation of the byte.
type	*object*
properties	
• id	*/ui/keymap-defs/<keymap_name>/<key_seq>/id* The identifier that can be used to refer to this key
type	*string*
• command	*/ui/keymap-defs/<keymap_name>/<key_seq>/command* The command to execute for the given key sequence. Use a script to execute more complicated operations.
type	*string*
examples	:goto next hour
pattern	^$|^[:|;].*
• alt-msg	*/ui/keymap-defs/<keymap_name>/<key_seq>/alt-msg* The help message to display after the key is pressed.
type	*string*
additionalProperties	False
additionalProperties	False

8.4 Log Handling

The handling of logs is largely determined by the *log file formats*, this section covers options that are not specific to a particular format.

8.4.1 Timezone Conversion (v0.12.0+)

Log messages that have a numeric timezone, like `-03:00` or `Z` for UTC, will be converted to the local timezone as given by the *TZ* environment variable. For example, a timestamp ending in *-03:00* will be treated as three hours behind UTC and then adjusted to the local timezone.

This behavior can be disabled by setting the `/log/date-time/convert-zoned-to-local` configuration property to `false`.

8.4.2 Watch Expressions (v0.11.0+)

Watch expressions can be used to fire an event when a log message matches a condition. You can then install a listener for these events and trigger an action to be performed. For example, to automate filtering based on identifiers, a watch expression can match messages that mention the ID and then a trigger can install a filter for that ID. Creating a watch expression is done by adding an entry into the `/log/watch-expressions` configuration tree. For example, to create a watch named "dhcpdiscover" that matches DHCPDISCOVER messages from the `dhclient` daemon, you would run the following:

```
:config /log/watch-expressions/dhcpdiscover/expr :log_procname = 'dhclient' AND␣
↪startswith(:log_body, 'DHCPDISCOVER')
```

The watch expression can refer to column names in the log message by prefixing them with a colon. The expression is evaluated by passing the log message fields as bound parameters and not against a table. The easiest way to test out an expression is with the *:mark-expr expr* command, since it will behave similarly. After changing the configuration, you'll need to restart lnav for the effect to take place. You can then query the `lnav_events` table to see any generated `https://lnav.org/event-log-msg-detected-v1.schema.json` events from the logs that were loaded:

```
;SELECT * FROM lnav_events
```

From there, you can create a SQLite trigger on the `lnav_events` table that will examine the event contents and perform an action. See the *Events (v0.11.0+)* section for more information on handling events.

/log/watch-expressions/<watch_name>

A log message watch expression		
type	*object*	
properties		
• expr	*/log/watch-expressions/<watch_name>/expr*	
	The SQL expression to execute for each input line. If expression evaluates to true, a 'log message detected' event will be published.	
	type	*string*
• enabled	*/log/watch-expressions/<watch_name>/enabled*	
	Indicates whether or not this expression should be evaluated during log processing.	
	type	*boolean*
additional-Properties	False	

8.4.3 Annotations (v0.12.0+)

Annotations are content generated by a script for a given log message and displayed along with the message, like comments and tags. Since the script is run asynchronously, it can do complex analysis without delaying loading or interrupting the viewing experience. An annotation is defined by a condition and a handler in the **lnav** configuration. The condition is tested against a log message to determine if the annotation is applicable. If it is, the handler script will be executed for that log message when the user runs the *:annotate* command.

Conditions are SQLite expressions like the ones passed to *:filter-expr* where the expression is appended to `SELECT 1 WHERE`. The expression can use bound variables that correspond to the columns that would be in the format table and are prefixed by a colon (`:`). For example, the standard `log_opid` table column can be accessed by using `:log_opid`.

> **Note**
>
> The expression is executed with bound variables because it can be applied to log messages from multiple formats. Writing an expression that could handle different formats would be more challenging. In this approach, variables for log message fields that are not part of a format will evaluate to `NULL`.

Handlers are executable script files that should be co-located with the configuration file that defined the annotation. The handler will be executed and a JSON object with log message data fed in on the standard input. The handler should then generate the annotation content on the standard output. The output is treated as Markdown, so the content can be styled as desired.

/log/annotations/<annotation_name>

type	*object*	
properties		
• description	*/log/annotations/<annotation_name>/description*	
	A description of this annotation	
	type	*string*
• condition	*/log/annotations/<annotation_name>/condition*	
	The SQLite expression to execute for a log message that determines whether or not this annotation is applicable. The expression is evaluated the same way as a filter expression	
	type	*string*
	minLength	1
• handler	*/log/annotations/<annotation_name>/handler*	
	The script to execute to generate the annotation content. A JSON object with the log message content will be sent to the script on the standard input	
	type	*string*
	minLength	1
additionalProperties	False	

8.5 Demultiplexing (v0.12.3+)

Files that contain a mix of content from different sources, like the output of `docker compose logs`, can be automatically demultiplexed so that *lnav* can process them correctly.

Each line of the input file must include a unique identifier that can be used to determine which service the line belongs to. A demultiplexer is a regular expression that extracts the identifier, the log message, and an optional timestamp. Once extracted, lines are distributed to separate files based on the identifier.

Demultiplexing can be done on plain text files using a regular expression or JSON-lines files where the JSON contains certain properties.

Demultiplexers that are based on regular expressions are defined in the main configuration under the `/log/demux` path. The pattern for the demuxer has the following known capture names:

mux_id
: (required) Captures the unique identifier.

body
: (required) Captures the body of the log message that should be written to the file.

timestamp
: (optional) Captures the timestamp for the log message. If this is available and the log message does

not have it's own timestamp, this will be used instead.

If there are additional captures, they will be included in the file metadata that can be accessed by the `lnav_file_demux_metadata` view of the `lnav_file_metadata` table.

8.5.1 Example

```
{
    "$schema": "https://lnav.org/schemas/config-v1.schema.json",
    "log": {
        "demux": {
            "recv-with-pod": {
                "control-pattern": "^===== (?:START|END) =====$",
                "pattern": "^(?<timestamp>\\d{4}-\\d{2}-\\d{2}T\\d{2}:\\d{2}:\\d{2}\\.\\
↪d{3}(?:Z|[+\\-]\\d{2}:\\d{2})) source=[a-zA-Z0-9][a-zA-Z0-9_\\.\\-]* (?<body>.*)␣
↪kubernetes_host=(?<k8s_host>[a-zA-Z0-9][a-zA-Z0-9_\\.\\-]*) kubernetes_pod_name=(?<mux_
↪id>[a-zA-Z0-9][a-zA-Z0-9_\\.\\-]*)"
            }
        }
    }
}
```

Sample Input Document:

```
===== START =====
2024-12-12T08:00:00.123Z source=service-a This is a log message kubernetes_host=host-a␣
↪kubernetes_pod_name=pod-1
2024-12-12T08:01:00.456Z source=service-b Another log message kubernetes_host=host-b␣
↪kubernetes_pod_name=pod-2
===== END =====
```

Demuxed Output Files:

File: *pod-1*

```
2024-12-12T08:00:00.123Z This is a log message
```

File: *pod-2*

```
2024-12-12T08:01:00.456Z Another log message
```

8.5.2 JSON-Lines Demultiplexing (v0.13.0+)

Demultiplexers for JSON-lines files are defined in the main configuration under the `/log/demux-json` path. The configuration needs to specify the following three properties:

timestamp
: (optional) The timestamp for the log message. If this is available and the log message does not have it's own timestamp, this will be used instead.

mux_id
: (required) Captures the unique identifier for the source of the log message.

body
: (required) Captures the body of the log message that should be written to the file.

8.5.3 Behavior Details

Control Lines:

- Control lines match the control-pattern and are used to separate sections of the file. These lines are ignored for matching log patterns but are placed into an _out_of_frame_ sub-file.
- If the log file contains control lines at the beginning, only demultiplexers with a matching control-pattern will attempt to process the subsequent lines.

Demuxing Process:

- For each non-control line, lnav checks against all defined demultiplexers.
- The first demultiplexer whose pattern matches the line is applied.
- If a match occurs, the captured *mux_id* and *body* fields are required. If either is missing, the match is discarded.
- Matched lines are written to sub-files corresponding to their *mux_id* values.
- Sub-files created by demultiplexing are treated as standalone files for further processing. They will not undergo additional demuxing.

8.5.4 JSON Schema Reference

/log/demux/<name>

The definition of a demultiplexer	
type	*object*
properties	
• enabled	*/log/demux/<name>/enabled*
	Indicates whether this demuxer will be used at the demuxing stage (defaults to 'true')
	type *boolean*
• pattern	*/log/demux/<name>/pattern*
	A regular expression to match a line in a multiplexed file
	type *string*
• control-pattern	*/log/demux/<name>/control-pattern*
	A regular expression to match a control line in a multiplexed file
	type *string*
additionalProperties	False

/log/demux-json/<name>

The definition of a JSON demultiplexer		
type	*object*	
properties		
• enabled	*/log/demux-json/<name>/enabled*	
	Indicates whether this demuxer will be used at the demuxing stage (defaults to 'true')	
	type	*boolean*
• timestamp	*/log/demux-json/<name>/timestamp*	
	The pointer to the timestamp of the message	
	type	*string*
• mux_id	*/log/demux-json/<name>/mux_id*	
	The pointer to the ID for demultiplexing	
	type	*string*
• body	*/log/demux-json/<name>/body*	
	The pointer to the property that contains the log message	
	type	*string*
additionalProperties	False	

8.6 Tuning

The following configuration options can be used to tune the internals of **lnav** to your liking. The options can be changed using the `:config` command.

8.6.1 /tuning/archive-manager

Settings related to opening archive files		
type	*object*	
properties		
• min-free-space	*/tuning/archive-manager/min-free-space*	
	The minimum free space, in bytes, to maintain when unpacking archives	
	type	*integer*
	minimum	0
• cache-ttl	*/tuning/archive-manager/cache-ttl*	
	The time-to-live for unpacked archives, expressed as a duration (e.g. '3d' for three days)	
	type	*string*
	examples	3d
		12h
additionalProperties	False	

8.6.2 /tuning/clipboard

Settings related to the clipboard				
type	*object*			
properties				
• impls	*/tuning/clipboard/impls*			
	Clipboard implementations			
	type	*object*		
	patternProperties			
	• ^([\w\-]+)$	*/tuning/clipboard/impls/<clipboard_impl_name>*		
		Clipboard implementation		
		type	*object*	
		properties		
		• test	*/tuning/clipboard/impls/<clipboard_impl_name>/test*	
			The command that checks if a clipboard command is available	
			type	*string*
			examples	command -v pbcopy
		• general	*/tuning/clipboard/impls/<clipboard_impl_name>/general*	
			Commands to work with the general clipboard	
			clip-commands	
		• find	*/tuning/clipboard/impls/<clipboard_impl_name>/find*	
			Commands to work with the find clipboard	
			clip-commands	
		additionalProperties	False	
	additionalProperties	False		
additionalProperties	False			

8.6.3 /tuning/piper

Settings related to capturing piped data			
type	*object*		
properties			
• max-size	*/tuning/piper/max-size*		
	The maximum size of a capture file		
	type	*integer*	
	minimum	128	
• rotations	*/tuning/piper/rotations*		
	The number of rotated files to keep		
	type	*integer*	
	minimum	2	
• ttl	*/tuning/piper/ttl*		
	The time-to-live for captured data, expressed as a duration (e.g. '3d' for three days)		
	type	*string*	
	examples	3d	
		12h	
additionalProperties	False		

8.6.4 clip-commands

Container for the commands used to read from and write to the system clipboard	
type	*object*
properties	
• write	*/write*
	The command used to write to the clipboard
	type *string*
	examples pbcopy
• read	*/read*
	The command used to read from the clipboard
	type *string*
	examples pbpaste
additionalProperties	False

8.6.5 /tuning/file-vtab

Settings related to the lnav_file virtual-table	
type	*object*
properties	
• max-content-size	*/tuning/file-vtab/max-content-size*
	The maximum allowed file size for the content column
	type *integer*
	minimum 0
additionalProperties	False

8.6.6 /tuning/logfile

Settings related to log files	
type	*object*
properties	
• max-unrecognized-lines	*/tuning/logfile/max-unrecognized-lines*
	The maximum number of lines in a file to use when detecting the format
	type *integer*
	minimum 1
additionalProperties	False

8.6.7 /tuning/remote/ssh

Settings related to the ssh command used to contact remote machines	
type	*object*
properties	
• command	*/tuning/remote/ssh/command*
	The SSH command to execute
	type *string*
• transfer-command	*/tuning/remote/ssh/transfer-command*
	Command executed on the remote host when transferring the file
	type *string*

continues on next page

Table 6 – continued from previous page

• start-command		*/tuning/remote/ssh/start-command*		
		Command executed on the remote host to start the tailer		
	type	*string*		
• flags		*/tuning/remote/ssh/flags*		
		The flags to pass to the SSH command		
	type	*string*		
• options		*/tuning/remote/ssh/options*		
		The options to pass to the SSH command		
	type	*object*		
	patternProperties			
	• ^(\w+)$	*/tuning/remote/ssh/options/<option_name>*		
		Set an option to be passed to the SSH command		
		type	*string*	
	additionalProperties	False		
• config		*/tuning/remote/ssh/config*		
		The ssh_config options to pass to SSH with the -o option		
	type	*object*		
	patternProperties			
	• ^(\w+)$	*/tuning/remote/ssh/config/<config_name>*		
		Set an SSH configuration value		
		type	*string*	
	additionalProperties	False		
additionalProperties		False		

8.6.8 /tuning/url-scheme

Settings related to custom URL handling				
type	*object*			
patternProperties				
• ^([a-z][\w\-\+\.]+)$	*/tuning/url-scheme/<url_scheme>*			
	Definition of a custom URL scheme			
	type	*object*		
	properties			
	• handler	*/tuning/url-scheme/<url_scheme>/handler*		
		The name of the lnav script that can handle URLs with of this scheme. This should not include the '.lnav' suffix.		
		type	*string*	
		pattern	^[\w\-]+(?!\.lnav)$	
	additionalProperties	False		
additionalProperties	False			

CHAPTER

NINE

LOG FORMATS

9.1 Built-in Formats

Log files loaded into **lnav** are parsed based on formats defined in configuration files. Many formats are already built in to the **lnav** binary and you can define your own using a JSON file. When loading files, each format is checked to see if it can parse the first few lines in the file. Once a match is found, that format will be considered that files format and used to parse the remaining lines in the file. If no match is found, the file is considered to be plain text and can be viewed in the "text" view that is accessed with the **t** key.

The following log formats are built into **lnav**:

Name	Table Name	Description
Common Access Log	access_log	The default web access log format for servers like Apache.
Amazon ALB log	alb_log	Log format for Amazon Application Load Balancers
Generic Block	block_log	A generic format for logs, like cron, that have a date at the start of a block.
Bunyan log	bunyan_log	Bunyan JSON logging library for node.js
Caddy log format	caddy_log	
Candlepin log format	candlepin_log	Log format used by Candlepin registration system
Yum choose_repo Log	choose_repo_log	The log format for the yum choose_repo tool.
Cloudflare Access Log	cloudflare_json_log	Cloudflare Enterprise detailed logs of metadata
CloudVM Ram Log	cloudvm_ram_l	Periodic dumps of ram sizes
CUPS log format	cups_log	Log format used by the Common Unix Printing System
Dpkg Log	dpkg_log	The debian dpkg log.
ecs	ecs_log	Elastic Common Schema (ECS) defines a common set of fields for ingesting data into Elasticsearch including log records
Amazon ELB log	elb_log	Log format for Amazon Elastic Load Balancers
engine log	engine_log	The log format for the engine.log files from RHEV/oVirt
env_logger format	env_logger_log	Format file for Rust's env_logger crate
Common Error Log	error_log	The default web error log format for servers like Apache.
ESXi Syslog	esx_syslog_log	Format specific to the ESXi syslog
Fsck_hfs Log	fsck_hfs_log	Log for the fsck_hfs tool on Mac OS X.
GitHub Events Log	github_events_l	Format for the public GitHub timeline from gharchive.org
Glog	glog_log	The google glog format.
HAProxy HTTP Log Format	haproxy_log	The HAProxy log format
IntelliJ IDEA Log File	idea_log	Log file generated by IntelliJ IDEA-based IDEs (IntelliJ IDEA, PyCharm, WebStorm, etc.)
Java log format	java_log	Log format used by log4j and output by most java programs

continues on next page

Table 1 – continued from previous page

Name	Table Name	Description
journalctl JSON log format	journald_json_log	Logger format as created by systemd journalctl -o json
Katello log format	katello_log	Log format used by katello and foreman as used in Satellite 6.
Laravel	laravel_log	Laravel log format
lnav debug log	lnav_debug_log	Log format for lnav's own debug log
Android Logcat	logcat_log	Format for Android Logcat tested with the following formats verbs and adverbs: time,threadtime,usec,uid.
macOS unified log	macosuni_log	Format for ndjson logs produced by macOS
MongoDB log	mongodb_json_log	MongoDB structured JSON log format (4.4+)
MySQL Error Log	mysql_error_log	Format for MySQL error logs
MySQL General Log	mysql_gen_log	Format for the MySQL general log
MySQL Slow Query	mysql_slow_log	Format for the MySQL slow query log
Nextcloud server logs	nextcloud	Nextcloud JSON server logs audit.log, flow.log, and nextcloud.log
Nextflow log format	nextflow_log	Format file for nextflow.io logs
OpenAM Log	openam_log	The OpenAM identity provider.
OpenAM Debug Log	opendb_log	Debug logs for the OpenAM identity provider.
OpenStack log format	openstack_log	The log format for the OpenStack log files
OpenTelemetry Collector File Exporter	otel_collector_l	Format for OpenTelemetry Collector file exporter JSON logs
Open Telemetry Python	otlp_python_log	Format for Open Telemetry services in python
CUPS Page Log	page_log	The CUPS server log of printed pages.
Packet Capture	pcap_log	Internal format for pcap files
Pino log	pino_log	Pino JSON logging library for node.js
PostgreSQL	postgres_log	Format for PostgreSQL logs
Process State	procstate_log	Periodic dumps of process state
Proxifier	proxifier_log	Proxifier proxy client
Ruby on Rails	rails_log	Format for ruby on rails
Redis	redis_log	The Redis database
Robot Framework debug log	robot_fwk_log	Robot Framework debug log
Rust Tracing crate formatting. Expects a tracing subscribed like so *tracing_subscriber::fmt().json(* (or compatibile)	rust_tracing_log	
S3 Access Log	s3_log	S3 server access log format
simple_logger format	simple_rs_log	Format file for Rust's simple_logger crate
SnapLogic Server Log	snaplogic_log	The SnapLogic server log format.
spdlog C++ logs	spdlog_log	Format for the spdlog C++ logging library
SSSD log format	sssd_log	Log format used by the System Security Services Daemon
Strace	strace_log	The strace output format.
Syslog	syslog_log	The system logger format found on most posix systems.
TCF Log	tcf_log	Target Communication Framework log
TCSH History	tcsh_history	The tcsh history file format.
UniFi iptables log	unifi_iptables_l	The UniFi gateway iptables logger format (for /var/log/iptables).
UniFi log	unifi_log	The UniFi gateway messages logger format (for /var/log/messages).
Uwsgi Log	uwsgi_log	The uwsgi log format.

continues on next page

Table 1 – continued from previous page

Name	Table Name	Description
Vdsm Logs	vdsm_log	Vdsm log format
VMKernel Logs	vmk_log	The VMKernel's log format
VMware Logs	vmw_log	One of the log formats used in VMware's ESXi and vCenter software.
VMware vSphere log format	vmw_py_log	The log format for some VMware vSphere services
VMware Go Log	vmw_vc_svc_lo	Log files for go-based logs
VMWare PostgreSQL	vpostgres_log	Format for vpostgresql log files with format '%m %c %x %d %u %r %p %l'
web robot log	web_robot_log	
RHN server XMLRPC log format	xmlrpc_log	Generated by Satellite's XMLRPC component
Zap Console Log	zap_console_log	The Uber Zap log format
Zellij's logs	zellijFmt	Zellij's format file. Zellij is a terminal multiplexer
ZooKeeper log format	zookeeper_log	Log format for the ZooKeeper coordination service

9.1.1 XSV Formats

In addition to the above formats, the following self-describing formats are supported:

- The Bro Network Security Monitor TSV log format is supported in lnav versions v0.8.3+. The Bro log format is self-describing, so **lnav** will read the header to determine the shape of the file.
- The W3C Extended Log File Format is supported in lnav versions v0.10.0+. The W3C log format is self-describing, so **lnav** will read the header to determine the shape of the file.

9.1.2 JSON-lines

Logs encoded as JSON-lines can be parsed and pretty-printed in lnav by creating a log format file. The format file is a bit simpler to create since it doesn't require a regular expression to match plain text. Instead, the format defines the relevant fields and provides a `line-format` array that specifies how the fields in the JSON object should be displayed.

See the following formats that are built into lnav as examples:

- cloudflare_log.json
- github_events_log.json

9.1.3 logfmt

There is also basic support for the logfmt convention for formatting log messages. Files that use this format must have the entire line be key/value pairs. If the file you're using does not quite follow this formatting, but wraps logfmt data with another recognized format, you can use the *logfmt2json(str)* SQL function to convert the data into JSON for further analysis.

The following keys are recognized by lnav:

- `timestamp`, `time`, `ts`, `t`: The timestamp for the log message.
- `level`, `lvl`: The log level.
- `message`, `msg`: The body of the message.

Any other keys are available in the `fields` column of the `logfmt_log` table as a JSON object.

9.2 Defining a New Format

New log formats can be defined by placing JSON configuration files in subdirectories of the `/etc/lnav/formats` and `~/.lnav/formats/` directories. The directories and files can be named anything you like, but the files must have the '.json' suffix. A sample file containing the builtin configuration will be written to this directory when **lnav** starts up. You can consult that file when writing your own formats or if you need to modify existing ones. Format directories can also contain '.sql' and '.lnav' script files that can be used automate log file analysis.

9.2.1 Creating a Format Using Regex101.com (v0.11.0+)

For plain-text log files, the easiest way to create a log format definition is to create the regular expression that recognizes log messages using https://regex101.com . Simply copy a log line into the test string input box on the site and then start editing a PCRE2 regular expression. When building the regular expression, you'll want to use named captures for the structured parts of the log message. Any raw message text should be matched by a captured named "body". Once you have a regex that matches the whole log message, you can use **lnav**'s "management CLI" to create a skeleton format file. The skeleton will be populated with the regular expression from the site and the test string, along with any unit tests, will be added to the "samples" list. The "regex101 import" management command is used to create the skeleton and has the following form:

```
lnav -m regex101 import <regex101-url> <format-name> [<regex-name>]
```

If the import was successful, the path to the new format file should be printed out. The skeleton will most likely need some changes to make it fully functional. For example, the `kind` properties for captured values default to `string`, but you'll want to change them to the appropriate type.

9.2.2 Format File Reference

An **lnav** format file must contain a single JSON object, preferably with a `$schema` property that refers to the format-v1.schema, like so:

```
{
    "$schema": "https://lnav.org/schemas/format-v1.schema.json"
}
```

Each format to be defined in the file should be a separate field in the top-level object. The field name should be the symbolic name of the format and consist only of alphanumeric characters and underscores. This value will also be used as the SQL table name for the log. The value for each field should be another object with the following fields:

title
: The short and human-readable name for the format.

description
: A longer description of the format.

url
: A URL to the definition of the format.

file-pattern
: A regular expression used to match log file paths. Typically, every file format will be tried during the detection process. This field can be used to limit which files a format is applied to in case there is a potential for conflicts.

regex
: This object contains sub-objects that describe the message formats to match in a plain-text log file. Each `regex` MUST only match one type of log message. It must not match log messages that are matched by other regexes in this format. This uniqueness requirement is necessary because **lnav** will "lock-on" to a regex and use it to match against the next line in a file. So, if the regexes do

not uniquely match each type of log message, messages can be matched by the wrong regex. The "lock-on" behavior is needed to avoid the performance hit of having to try too many different regexes.

> **Note**
>
> If the format allows for multiline log entries, the regex must match also only the first line for automatic format detection to work correctly, i.e. everything after the first line must be optional.

> **Note**
>
> Log files that contain JSON messages should not specify this field.

pattern
The regular expression that should be used to match log messages. The PCRE2 library is used by **lnav** to do all regular expression matching.

json
True if each log line is JSON-encoded.

converter
An object that describes how an input file can be detected and then converted to a form that can be interpreted by **lnav**. For example, a PCAP file is in a binary format that cannot be handled natively by **lnav**. However, a PCAP file can be converted by `tshark` into JSON-lines that can be handled by **lnav**. So, this configuration describes how the input file format can be detected and converted. See *Automatic File Conversion* for more information.

header
An object that describes how to match the header of the input file.

expr
An object that contains SQLite expressions that can be used to check if the input file's header is of this type. The property name is the name of the expression and the value is the expression. The expression is evaluated with the following variables:

:header
The hex-encoded version of the header content.

:filepath
The path to the input file.

size
The minimum size of header that is needed to do the match.

command
The command to execute to convert the input file.

line-format
An array that specifies the text format for JSON-encoded log messages. Log files that are JSON-encoded will have each message converted from the raw JSON encoding into this format. Each element is either an object that defines which fields should be inserted into the final message string and or a string constant that should be inserted. For example, the following configuration will transform each log message object into a string that contains the timestamp, followed by a space, and then the message body:

```
[ { "field": "ts" }, " ", { "field": "msg" } ]
```

Note

Line-feeds at the end of a value are automatically stripped.

field
: The name or JSON-Pointer of the message field that should be inserted at this point in the message. The special `__timestamp__` field name can be used to insert a human-readable timestamp. The `__level__` field can be used to insert the level name as defined by lnav. The `__duration__` field can be used to insert a humanized duration value (e.g. "1m23s") when a duration field is defined for the format.

Tip

Use a JSON-Pointer to reference nested fields. For example, to include a "procname" property that is nested in a "details" object, you would write the field reference as `/details/procname`.

min-width
: The minimum width for the field. If the value for the field in a given log message is shorter, padding will be added as needed to meet the minimum-width requirement. (v0.8.2+)

max-width
: The maximum width for the field. If the value for the field in a given log message is longer, the overflow algorithm will be applied to try and shorten the field. (v0.8.2+)

auto-width
: Flag that indicates that the width of the field should automatically be set to the widest value seen. (v0.11.2)

align
: Specifies the alignment for the field, either "left" or "right". If "left", padding to meet the minimum-width will be added on the right. If "right", padding will be added on the left. (v0.8.2+)

overflow
: The algorithm used to shorten a field that is longer than "max-width". The following algorithms are supported:

  abbrev
  : Removes all but the first letter in dotted text. For example, "com.example.foo" would be shortened to "c.e.foo".

  truncate
  : Truncates any text past the maximum width.

  dot-dot
  : Cuts out the middle of the text and replaces it with two dots (i.e. '..').

  last-word
  : Removes all but the last word in text with dot, dash,

forward-slash, or colon separators. For example, "com.example.foo" would be shortened to "foo".

(v0.8.2+)

timestamp-format
The timestamp format to use when displaying the time for this log message. (v0.8.2+)

default-value
The default value to use if the field could not be found in the current log message. The built-in default is "-".

text-transform
Transform the text in the field. Supported options are: none, uppercase, lowercase, capitalize

prefix
Text to prepend to the value. If the value is empty, this prefix will not be added.

suffix
Text to append to the value. If the value is empty, this suffix will not be added.

timestamp-field
The name of the field that contains the log message timestamp. Internally, timestamps are stored with microsecond precision. Defaults to "timestamp".

timestamp-format
An array of timestamp formats using a subset of the strftime conversion specification. The following conversions are supported: %a, %b, %L, %M, %H, %I, %d, %e, %j, %k, %l, %m, %p, %y, %Y, %S, %s, %Z, %z. In addition, you can also use the following:

%L
Milliseconds as a decimal number (range 000 to 999).

%f
Microseconds as a decimal number (range 000000 to 999999).

%N
Nanoseconds as a decimal number (range 000000000 to 999999999).

%q
Seconds from the epoch as a hexidecimal number.

%i
Milliseconds from the epoch.

%6
Microseconds from the epoch.

%9
Nanoseconds from the epoch.

timestamp-divisor
For JSON logs with numeric timestamps, this value is used to divide the timestamp by to get the number of seconds and fractional seconds.

subsecond-field
(v0.11.1+) The path to the property in a JSON-lines log message that contains the sub-second time value

subsecond-units
(v0.11.1+) The units of the subsecond-field property value. The following values are supported:

milli
for milliseconds

micro
for microseconds

nano
for nanoseconds

timestamp-point-of-reference
(v0.14.0+) Specifies the relationship of the timestamp to the operation that the message refers to. This is used in conjunction with `duration-field` to determine time spans in the TIMELINE view. The following values are supported:

end
The timestamp indicates when the message was sent/logged. This is the default.

start
The timestamp indicates when the operation started. The operation's time span will extend from the timestamp to the timestamp plus the duration.

start-timestamp-field
The name of a field that contains the start time of the operation. When set, the `timestamp-field` is treated as the end time and the duration is computed as the difference between the two. The `timestamp-divisor` is applied to both fields. This is an alternative to using `duration-field` for logs that record separate start and end timestamps.

ordered-by-time
(v0.8.3+) Indicates that the order of messages in the file is time-based. Files that are not naturally ordered by time will be sorted in order to display them in the correct order. Note that this sorting can incur a performance penalty when tailing logs.

level-field
The name of the regex capture group that contains the log message level. Defaults to "level".

The following log level strings are recognized automatically (case-insensitive) and do not require a custom `level` mapping:

Level	Recognized strings
trace	`trace`, `verbose`
debug	`debug`
debug2	`debug2`
debug3	`debug3`
debug4	`debug4`
debug5	`debug5`
info	`info`, `system`
notice	`notice`, `note`, `log`
stats	`stats`
warning	`warn`, `warning`, `deprecation`
error	`err`, `error`, `fail`
critical	`critical`, `severe`, `alert`
fatal	`fatal`, `emergency`

Single-letter abbreviations are also recognized: **T** (trace), **D/V** (debug), **I** (info), **S** (stats), **N** (notice), **W** (warning), **E** (error), **C** (critical), **F** (fatal).

body-field
The name of the field that contains the main body of the message. Defaults to "body".

opid-field
The name of the field that contains the "operation ID" of the message. An "operation ID" establishes a thread of messages that might correspond to a particular operation/request/transaction. The user can press the 'o' or 'Shift+O' hotkeys to move forward/backward through the list of messages that have the same operation ID. Note: For JSON-encoded logs, the opid field can be a path (e.g. "foo/bar/opid") if the field is nested in an object and it MUST be included in the "line-format" for the 'o' hotkeys to work.

(v0.14.0+) For JSON-lines logs, the opid field can refer to a JSON array or object. The OPID will be computed by hashing the contents of the array or object and the description will be the container itself. For example, the `spans` array in a Rust tracing log message.

To construct an OPID from multiple fields, leave `opid-field` blank and create a single `opid/description` definition with a `format` array. The content of the format fields will be hashed to create the OPID. For example, the built-in `access_log` format uses `c_ip` and `cs_user_agent` as the OPID.

opid
This object contains further options related to OP IDs:

source
Specifies the source of the operation ID if `opid-field` is not set. The possible values are:

from-description
The description captured from the log message is hashed and used as the operation ID. This is the default if a description is set.

from-whole-msg
The log message line is hashed and used as the operation ID. This is the default if no descriptions are given.

description
This object contains definitions for how to construct a description of an operation. Each definition should contain a `format` array with objects that have the following fields:

field
The field in the log message to capture as part of the description.

extractor
An optional regular expression used to extract portions of the `field`.

prefix
A prefix to insert before this field in the description.

suffix
A suffix to insert after this field in the description.

thread-id-field
The name of the field that contains the identifier for a thread. Thread identifiers are tracked by lnav and can be accessed through the `all_thread_ids` table.

duration-field
The name of the field that contains the duration of an operation. If a duration is available, it will be used to calculate time spans in the TIMELINE view.

duration-divisor
: The value to divide a duration by to convert it to seconds. For example, if the duration field is in milliseconds, the divisor should be 1000.

src-file-field
: (v0.14.0+) The name of the field that contains the source file name where the log statement originated. This field is accessible in SQL queries as the `log_src_file` column.

src-line-field
: (v0.14.0+) The name of the field that contains the source line number where the log statement originated. This field is accessible in SQL queries as the `log_src_line` column.

src-location-field
: (v0.14.0+) The name of a field that contains both the source file and line number as a combined value (e.g. `file.c:42`). This is an alternative to using both `src-file-field` and `src-line-field` separately. The field will be parsed to populate the `log_src_file` and `log_src_line` SQL columns.

hide-extra
: A boolean for JSON logs that, when `true`, hides fields not defined in the `value` object.

level
: A mapping of error levels to regular expressions. During scanning the contents of the capture group specified by *level-field* will be checked against each of these regexes. Once a match is found, the log message level will set to the corresponding level. The available levels, in order of severity, are: **fatal**, **critical**, **error**, **warning**, **stats**, **info**, **debug**, **debug2-5**, **trace**. For JSON logs with exact numeric levels, the number for the corresponding level can be supplied. If the JSON log format uses numeric ranges instead of exact numbers, you can supply a pattern and the number found in the log will be converted to a string for pattern-matching.

> **Note**
>
> The regular expression is not anchored to the start of the string by default, so an expression like `1` will match `-1`. If you want to exactly match `1`, you would use `^1$` as the expression.

multiline
: If false, **lnav** will consider any log lines that do not match one of the message patterns to be in error when checking files with the '-C' option. This flag will not affect normal viewing operation. Default: true.

value
: This object contains the definitions for the values captured by the regexes.

  kind
  : The type of data that was captured **string**, **integer**, **float**, **json**, **quoted**, **timestamp**.

  collate
  : The name of the SQLite collation function for this value. The standard SQLite collation functions can be used as well as the ones defined by lnav, as described in *Collators*.

  identifier
  : A boolean that indicates whether or not this field represents an identifier and should be syntax colored.

  foreign-key
  : A boolean that indicates that this field is a key and should not be graphed. This should only need to be set for integer fields.

hidden
A boolean for log fields that indicates whether they should be displayed. The behavior is slightly different for JSON logs and text logs. For a JSON log, this property determines whether an extra line will be added with the key/value pair. For text logs, this property controls whether the value should be displayed by default or replaced with an ellipsis.

rewriter
A command to rewrite this field when pretty-printing log messages containing this value. The command must start with ':', ';', or '|' to signify whether it is a regular command, SQL query, or a script to be executed. The other fields in the line are accessible in SQL by using the ':' prefix. The text value of this field will then be replaced with the result of the command when pretty-printing. For example, the HTTP access log format will rewrite the status code field to include the textual version (e.g. 200 (OK)) using the following SQL query:

```
;SELECT :sc_status || ' (' || (
    SELECT message FROM http_status_codes
        WHERE status = :sc_status) || ') '
```

highlights
(v0.14.0+) This object contains definitions for patterns to be highlighted within this specific field, rather than across the whole log line. Each entry should have a name and a definition with the following fields:

pattern
The regular expression to match within the field value.

base-style
The style to apply to the entire matched text. This is an object with the following fields:

color
The foreground color. Colors can be specified using hexadecimal notation (e.g. `#aabbcc`) or using a color name.

background-color
The background color.

underline
If true, underline the text.

bold
If true, bold the text.

italic
If true, italicize the text.

strike
If true, strike through the text.

nestable
If true, this highlight can be applied to text contained within another highlight. Defaults to `true`.

captures
This object maps named capture groups in the pattern to individual styles. Each key should be the name of a capture group and the value is a style object (with the same fields as `base-style`).

For example, the following highlights Java package names within a `tag` field, with the final component in a different color:

```
"tag": {
    "kind": "string",
    "identifier": true,
    "highlights": {
        "package": {
            "pattern": "(?<pkg>([a-z]+\\\\.){2,})(?<cls>[a-
↪z]+)(?=[ '\\\"])",
            "base-style": {
                "color": "#97d1F6"
            },
            "captures": {
                "cls": {
                    "color": "#c0d1F6",
                    "bold": true
                }
            }
        }
    }
}
```

tags
This object contains the tags that should automatically be added to log messages.

pattern
The regular expression evaluated over a line in the log file as it is read in. If there is a match, the log message the line is a part of will have this tag added to it.

paths
This array contains objects that define restrictions on the file paths that the tags will be applied to. The objects in this array can contain:

glob
A glob pattern to check against the log files read by lnav.

partitions
This object contains a description of partitions that should automatically be created in the log view.

pattern
The regular expression evaluated over a line in the log file as it is read in. If there is a match, the log message the line is a part of will be used as the start of the partition. The name of the partition will be taken from any captures in the regex.

paths
This array contains objects that define restrictions on the file paths in which partitions will be created. The objects in this array can contain:

glob
A glob pattern to check against the log files read by lnav.

sample
A list of objects that contain sample log messages. All formats must include at least one sample and it must be matched by one of the included regexes. Each object must contain the following field:

line
The sample message.

level
The expected error level. An error will be raised if this level does not match the level parsed by lnav for this sample message.

highlights
This object contains the definitions for patterns to be highlighted in a log message. Each entry should have a name and a definition with the following fields:

pattern
The regular expression to match in the log message body.

color
The foreground color to use when highlighting the part of the message that matched the pattern. If no color is specified, one will be picked automatically. Colors can be specified using hexadecimal notation by starting with a hash (e.g. #aabbcc) or using a color name as found at http://jonasjacek.github.io/colors/.

background-color
The background color to use when highlighting the part of the message that matched the pattern. If no background color is specified, black will be used. The background color is only considered if a foreground color is specified.

underline
If true, underline the part of the message that matched the pattern.

blink
If true, blink the part of the message that matched the pattern.

nestable
If true, this highlight can be applied to text contained within another highlight. Defaults to `true`.

Example format:

```
{
    "$schema": "https://lnav.org/schemas/format-v1.schema.json",
    "example_log" : {
        "title" : "Example Log Format",
        "description" : "Log format used in the documentation example.",
        "url" : "http://example.com/log-format.html",
        "regex" : {
            "basic" : {
                "pattern" : "^(?<timestamp>\\d{4}-\\d{2}-\\d{2}T\\d{2}:\\d{2}:\\d{2}\\.\\
↪d{3}Z)>>(?<level>\\w+)>>(?<component>\\w+)>>(?<body>.*)$"
            }
        },
        "level-field" : "level",
        "level" : {
            "error" : "ERROR",
            "warning" : "WARNING"
        },
        "value" : {
            "component" : {
                "kind" : "string",
                "identifier" : true
            }
        },
```

(continues on next page)

(continued from previous page)

```
        "sample" : [
            {
                "line" : "2011-04-01T15:14:34.203Z>>ERROR>>core>>Shit's on fire yo!"
            }
        ]
    }
}
```

9.2.3 Patching an Existing Format

When loading log formats from files, **lnav** will overlay any new data over previously loaded data. This feature allows you to override existing value or append new ones to the format configurations. For example, you can separately add a new regex to the example log format given above by creating another file with the following contents:

```
{
    "$schema": "https://lnav.org/schemas/format-v1.schema.json",
    "example_log" : {
        "regex" : {
            "custom1" : {
                "pattern" : "^(?<timestamp>\\d{4}-\\d{2}-\\d{2}T\\d{2}:\\d{2}:\\d{2}\\.\\
↪d{3}Z)<<(?<level>\\w+)--(?<component>\\w+)>>(?<body>.*)$"
            }
        },
        "sample" : [
            {
                "line" : "2011-04-01T15:14:34.203Z<<ERROR--core>>Shit's on fire yo!"
            }
        ]
    }
}
```

This example overrides the default syslog_log error detection regex to **not** match the `errors=` string.

```
{
  "syslog_log": {
      "level": {
          "error": "(?:(?:(?<![a-zA-Z]))(?:(?i)error(?:s)?(?!=))(?:(?![a-zA-
↪Z]))|failed|failure)"
      }
  }
}
```

9.2.4 Installing Format Files

File formats are loaded from subdirectories in `/etc/lnav/formats` and `~/.lnav/formats/`. You can manually create these subdirectories and copy the format files into there. Or, you can pass the `-i` option to **lnav** to automatically install formats from the command-line. For example:

```
$ lnav -i myformat.json
info: installed: /home/example/.lnav/formats/installed/myformat_log.json
```

Format files installed using this method will be placed in the `installed` subdirectory and named based on the first format name found in the file.

The *-i* option can also be used to install `.sql` and `.lnav` script files. The SQL files are executed on startup to create any helper tables or views and the '.lnav' script files can be executed using the pipe hotkey `|`.

You can also install formats from git repositories by passing the repository's clone URL. A standard set of repositories is maintained at (https://github.com/tstack/lnav-config) and can be installed by passing 'extra' on the command line, like so:

```
lnav -i extra
```

These repositories can be updated by running **lnav** with the '-u' flag.

Format files can also be made executable by adding a shebang (#!) line to the top of the file, like so:

```
#! /usr/bin/env lnav -i
{
    "myformat_log" : ...
}
```

Executing the format file should then install it automatically:

```
$ chmod ugo+rx myformat.json
$ ./myformat.json
info: installed: /home/example/.lnav/formats/installed/myformat_log.json
```

9.3 Format Order When Scanning a File

When **lnav** loads a file, it tries each log format against the first 15,000 lines[1] of the file trying to find a match. When a match is found, that log format will be locked in and used for the rest of the lines in that file. Since there may be overlap between formats, **lnav** performs a test on startup to determine which formats match each others sample lines. Using this information it will create an ordering of the formats so that the more specific formats are tried before the more generic ones. For example, a format that matches certain syslog messages will match its own sample lines, but not the ones in the syslog samples. On the other hand, the syslog format will match its own samples and those in the more specific format. You can see the order of the format by enabling debugging and checking the **lnav** log file for the "Format order" message:

```
lnav -d /tmp/lnav.log
```

For JSON-lines log files, the log message must have the timestamp property specified in the format in order to match. If multiple formats match a message, the format that has the most matching `line-format` elements will win (referred to as "quality"). In the case of a tie, the format with the least number of required `line-format` elements missing ("strikes") wins.

9.4 Automatic File Conversion

File formats that are not naturally understood by **lnav** can be automatically detected and converted to a usable form using the `converter` property. For example, PCAP files can be detected and converted to a JSON-lines form using `tshark`. The conversion process works as follows:

1. The first 1024 bytes of the file are read, if available.
2. This header is converted into a hex string.
3. For each log format that has defined a `converter`, every "header expression" is evaluated to see if there is a match. The header expressions are SQLite expressions where the following variables are defined:

[1] The maximum number of lines to check can be configured. See the *Tuning* section for more details.

:header
A string containing the header as a hex string.

:filepath
The path to the file.

4. If a match is found, the converter script defined in the log format will be invoked and passed the format name and path to the file as arguments. The script should write the converted form of the input file on its standard output. Any errors should be written to the standard error.
5. The log format will be associated with the original file will be used to interpret the converted file.

CHAPTER

TEN

SESSIONS

Session information is stored automatically for the set of files that were passed in on the command-line and reloaded the next time **lnav** is executed. The information currently stored is:

- Position within the files being viewed.
- Active searches for each view.
- *Log filters*.
- *Highlights*.
- *Hidden files*.
- *Hidden fields*.

Bookmarks and log-time adjustments are stored separately on a per-file basis. Note that the bookmarks are associated with files based on the content of the first line of the file so that they are preserved even if the file has been moved from its current location.

Session data is stored in the `~/.lnav` directory.

CHAPTER

ELEVEN

COMMANDS

Commands provide access to some of the more advanced features in **lnav**, like *filtering* and *"search tables"*. You can activate the command prompt by pressing the `:` key. At the prompt, you can start typing in the desired command and/or double-tap `TAB` to activate auto-completion and show the available commands. To guide you in the usage of the commands, a help window will appear above the command prompt with an explanation of the command and its parameters (if it has any). For example, the screenshot below shows the help for the `:open` command:

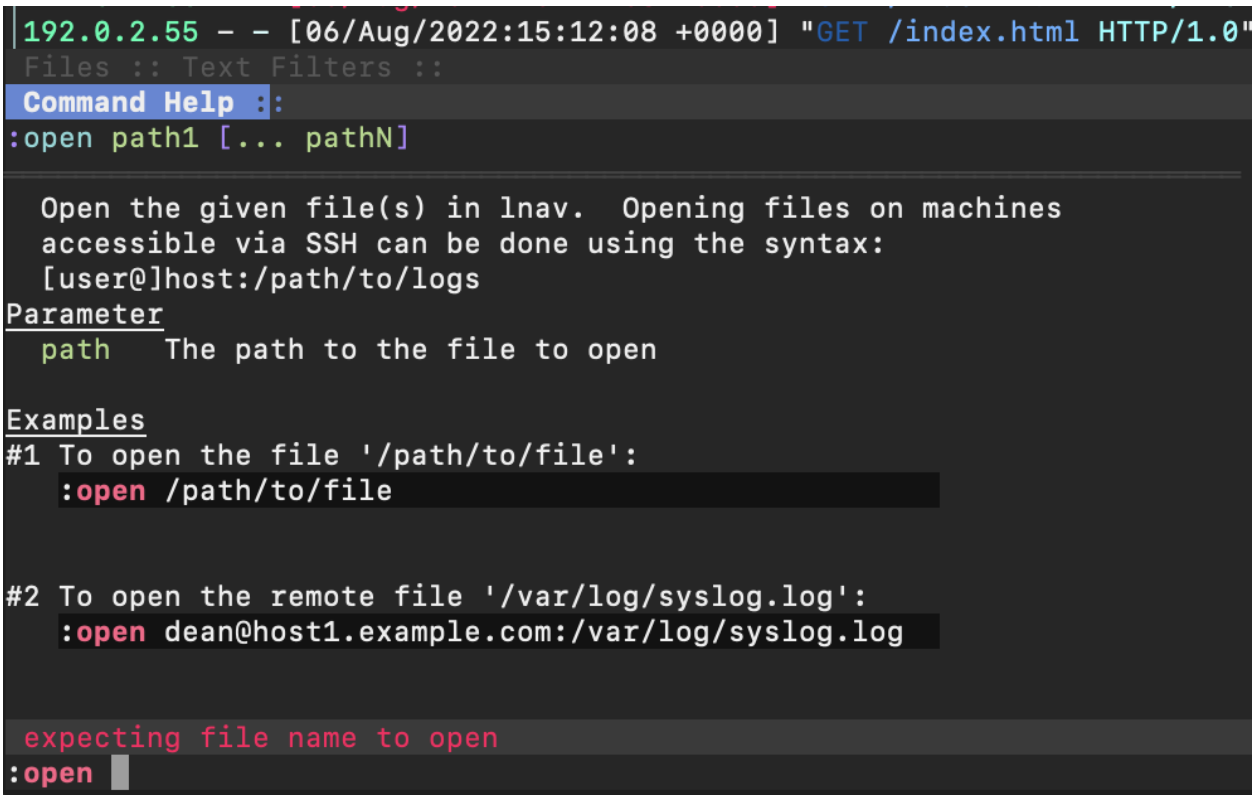

Fig. 1: Screenshot of the online help for the `:open` command.

In addition to online help, many commands provide a preview of the effects that the command will have. This preview will activate shortly after you have finished typing, but before you have pressed `Enter` to execute the command. For example, the `:open` command will show a preview of the first few lines of the file given as its argument:

The `:filter-out pattern` command is another instance where the preview behavior can help you craft the correct command-line. This command takes a PCRE2 regular expression that specifies the log messages that should be filtered out of the view. The preview for this command will highlight the portion of the log messages that match the expression

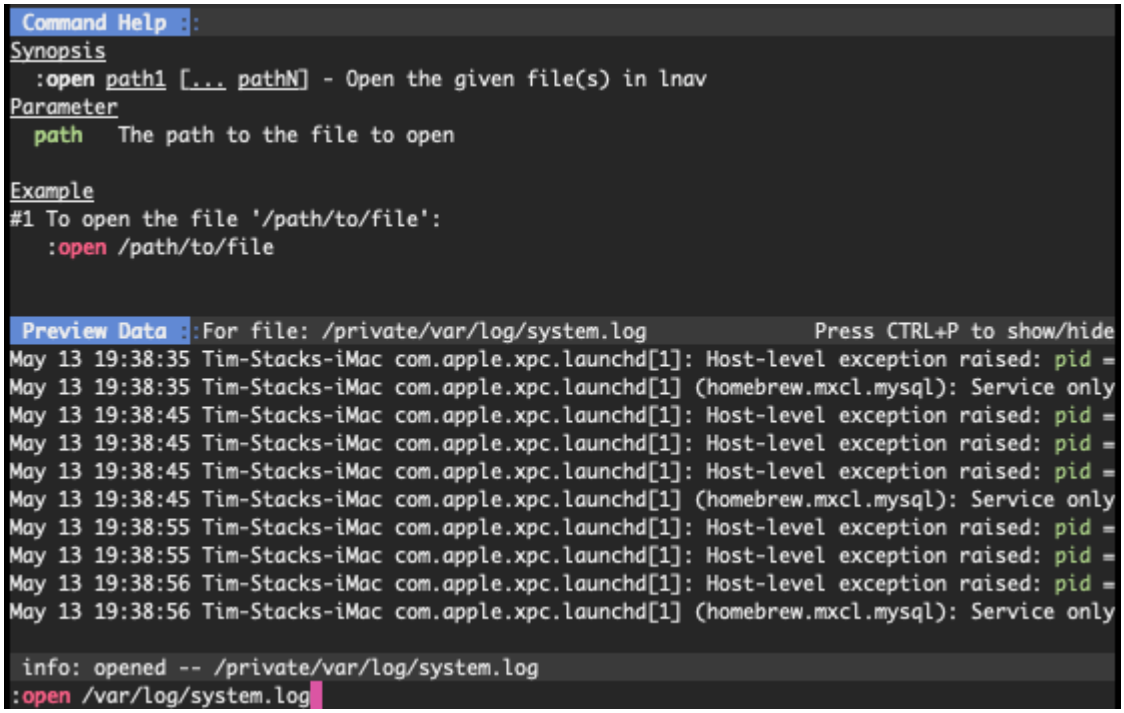

Fig. 2: Screenshot of the preview shown for the `:open` command.

in red. Thus, you can be certain that the regular expression is matching the log messages you are interested in before committing the filter. The following screenshot shows an example of this preview behavior for the string "launchd":

Any errors detected during preview will be shown in the status bar right above the command prompt. For example, an attempt to open an unknown file will show an error message in the status bar, like so:

Tip

Note that almost all commands support TAB-completion for their arguments. So, if you are in doubt as to what to type for an argument, you can double- tap the `TAB` key to get suggestions. For example, the TAB-completion for the `filter-in` command will suggest words that are currently displayed in the view.

Note

The following commands can be disabled by setting the `LNAVSECURE` environment variable before executing the **lnav** binary:

- `:cd`
- `:export-session-to`
- `:open`
- `:pipe-to`
- `:pipe-line-to`
- `:redirecto-to`
- `:sh`

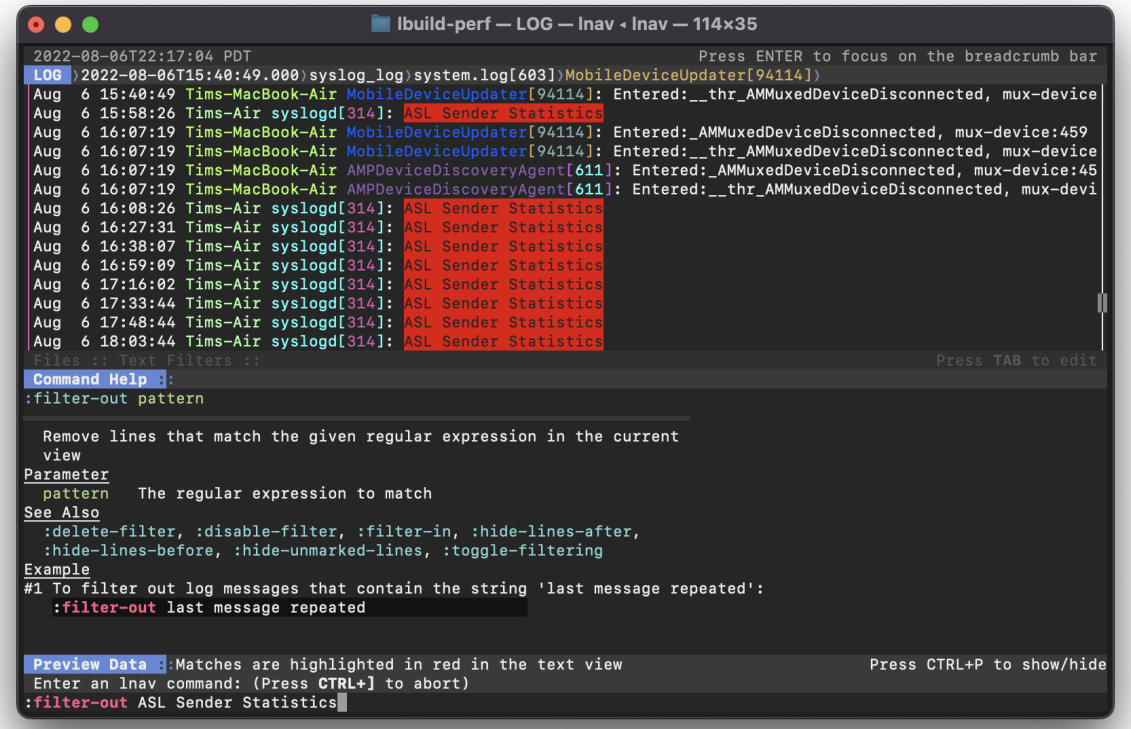

Fig. 3: Screenshot showing the preview for the `:filter-out` command.

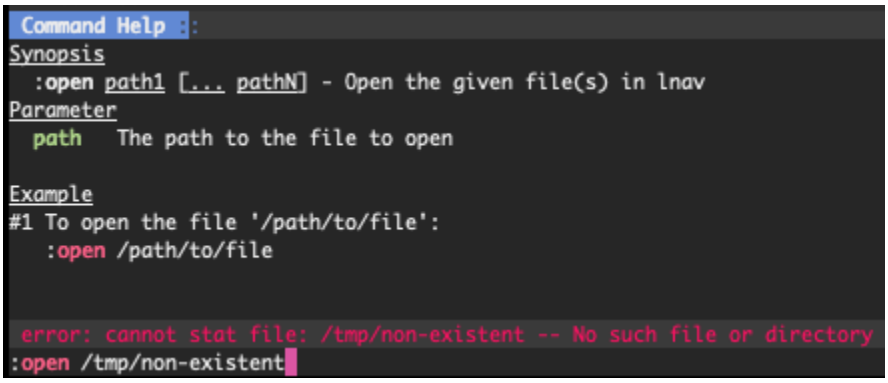

Fig. 4: Screenshot of the error shown when trying to open a non-existent file.

- `:write-*-to`

This makes it easier to run **lnav** in restricted environments without the risk of privilege escalation.

11.1 I/O Commands

11.1.1 Anonymization

Anonymization is the process of removing identifying information from content to make it safer for sharing with others. For example, an IP address can often be used to uniquely identify an entity. Substituting all instances of a particular IP with the same dummy value would remove the identifying data without losing statistical accuracy. **lnav** has built-in support for anonymization through the `--anonymize` flag on the `:write-*` collection of commands. While the anonymization process should catch most

IPv4 Addresses
: Are replaced with addresses in the `10.0.0.0/8` range.

IPv6 Addresses
: Are replaced with addresses in the `2001:db8::/32` range.

URL User Names
: Are replaced with a random animal name.

URL Passwords
: Are replaced with a hash of the input password.

URL Hosts
: Are replaced with a random name under the example.com domain.

URL Paths
: Are recursively examined for substitution.

URL Query Strings
: Are recursively examined for substitution.

URL Fragments
: Are recursively examined for substitution.

Paths
: Are recursively examined for substitution.

Credit Card Numbers
: Are replaced with a 16 digit hash of the input number.

MAC Addresses
: Are replaced with addresses in the `00:00:5E:00:53:00` range.

Hex Dumps
: Are replaced with a hash of the input replicated to the size of input.

Email User Names
: Are replaced with a random animal name.

Email Host Names
: Are replaced with a random name under the example.com domain.

Words
: Are replaced with a random word with a matching case style.

Quoted Strings
: Are recursively examined for substitution.

UUID
Are replaced with a hash of the input.

XML Attribute Values
Are recursively examined for substitution.

11.2 Reference

11.2.1 :add-source-path *path*

Add a path to the source code that generated log messages. Adding source allows lnav to more accurately extract values from log messages

Parameters

- **path** — The path to the source code to index

11.2.2 :adjust-log-time *timestamp*

Change the timestamps of the focused file to be relative to the given date

Parameters

- **timestamp*** — The new timestamp for the focused line in the view

Examples
To set the focused timestamp to a given date:

```
:adjust-log-time 2017-01-02T05:33:00
```

To set the focused timestamp back an hour:

```
:adjust-log-time -1h
```

11.2.3 :alt-msg *msg*

Display a message in the alternate command position

Parameters

- **msg*** — The message to display

Examples
To display 'Press t to switch to the text view' on the bottom right:

```
:alt-msg Press t to switch to the text view
```

See Also
:cd dir, *:echo [-n] msg*, *:eval command*, *:export-session-to path*, *:external-access-login [app]*, *:external-access port api-key*, *:rebuild*, *:redirect-to [path]*, *:sh –name=<name> cmdline*, *:write-csv-to [–anonymize] path*, *:write-json-cols-to [–anonymize] path*, *:write-json-to [–anonymize] path*, *:write-jsonlines-to [–all] [–anonymize] path*, *:write-raw-to [–view] [–anonymize] path*, *:write-screen-to [–anonymize] path*, *:write-table-to [–anonymize] path*, *:write-to [–anonymize] path*, *:write-view-to [–anonymize] path*

11.2.4 :annotate

Analyze the focused log message and attach annotations

See Also

:comment text, *:partition-name name*, *:tag tag*

11.2.5 :append-to *path*

Append marked lines in the current view to the given file

Parameters

- **path*** — The path to the file to append to

Examples

To append marked lines to the file /tmp/interesting-lines.txt:

```
:append-to /tmp/interesting-lines.txt
```

See Also

;.dump path table, *;.read path*, *;.save path*, *:echo [-n] msg*, *echoln(value)*, *:export-session-to path*, *:open [–since] [–until] path*, *:pipe-line-to shell-cmd*, *:pipe-to shell-cmd*, *:redirect-to [path]*, *:write-csv-to [–anonymize] path*, *:write-json-cols-to [–anonymize] path*, *:write-json-to [–anonymize] path*, *:write-jsonlines-to [–all] [–anonymize] path*, *:write-raw-to [–view] [–anonymize] path*, *:write-screen-to [–anonymize] path*, *:write-table-to [–anonymize] path*, *:write-to [–anonymize] path*, *:write-view-to [–anonymize] path*, *:xopen path*

11.2.6 :breakpoint *point*

Set a breakpoint for the given [<format>:]<file>:<line> tuples or the current line

Parameters

- **point** — The file and line number of the breakpoint. If the format is different from the currently focused one, the format name should be used as the prefix

Examples

To set a breakpoint for a log message at foo.cc:32:

```
:breakpoint foo.cc:32
```

11.2.7 :cd *dir*

Change the current directory

Parameters

- **dir*** — The new current directory

See Also

:alt-msg msg, *:echo [-n] msg*, *:eval command*, *:export-session-to path*, *:external-access-login [app]*, *:external-access port api-key*, *:rebuild*, *:redirect-to [path]*, *:sh –name=<name> cmdline*, *:write-csv-to [–anonymize] path*, *:write-json-cols-to [–anonymize] path*, *:write-json-to [–anonymize] path*, *:write-jsonlines-to [–all] [–anonymize] path*, *:write-raw-to [–view]*

[–anonymize] path, *:write-screen-to [–anonymize] path*, *:write-table-to [–anonymize] path*, *:write-to [–anonymize] path*, *:write-view-to [–anonymize] path*

11.2.8 :clear-adjusted-log-time

Clear the adjusted time for the focused line in the view

11.2.9 :clear-all-sticky-headers

Clear all sticky header bookmarks in the current view

See Also

:enable-word-wrap, *:hide-fields field-name*, *:hide-unmarked-lines*, *:mark*, *:next-mark type*, *:prev-mark type*, *:set-text-view-mode mode*, *:toggle-sticky-header*, *:toggle-sticky-header*

11.2.10 :clear-breakpoint *pattern*

Clear the breakpoints that match the given glob pattern

Parameters

- **pattern** — The glob pattern to use when matching the breakpoint definition of the format <format>:<file>:<line>

Examples

To clear all breakpoints:

```
:clear-breakpoint *
```

11.2.11 :clear-comment

Clear the comment attached to the focused log line

See Also

:annotate, *:comment text*, *:partition-name name*, *:tag tag*

11.2.12 :clear-file-timezone *pattern*

Clear the timezone setting for the focused file or the given glob pattern.

Parameters

- **pattern*** — The glob pattern to match against files that should no longer use this timezone

See Also

:set-file-timezone zone [pattern]

11.2.13 :clear-filter-expr

Clear the filter expression

See Also

:filter-expr expr, *:filter-in pattern*, *:filter-out pattern*, *:hide-lines-after date*, *:hide-lines-before date*, *:hide-unmarked-lines*, *:toggle-filtering*

11.2.14 :clear-highlight *pattern*

Remove a previously set highlight regular expression

Parameters

- **pattern*** — The regular expression previously used with :highlight

Examples

To clear the highlight with the pattern 'foobar':

```
:clear-highlight foobar
```

See Also

:clear-all-sticky-headers, *:enable-word-wrap*, *:hide-fields field-name*, *:highlight pattern*, *:set-text-view-mode mode*, *:toggle-sticky-header*

11.2.15 :clear-highlight-field *field*

Remove a field highlight

Parameters

- **field*** — The name of highlighted field

Examples

To clear the highlights for the 'sc_status' field:

```
:clear-highlight-field sc_status
```

See Also

:clear-all-sticky-headers, *:enable-word-wrap*, *:hide-fields field-name*, *:highlight-field [–color] [–bold] [–underline] [–italic] [–strike] [–blink] field [pattern]*, *:set-text-view-mode mode*, *:toggle-sticky-header*

11.2.16 :clear-mark-expr

Clear the mark expression

See Also

:clear-all-sticky-headers, *:hide-unmarked-lines*, *:mark-expr expr*, *:mark*, *:next-mark type*, *:prev-mark type*, *:toggle-sticky-header*

11.2.17 :clear-partition

Clear the partition the focused line is a part of

See Also

:annotate, *:comment text*, *:partition-name name*, *:tag tag*

11.2.18 :close *path*

Close the given file(s) or the focused file in the view

Parameters

- **path** — A path or glob pattern that specifies the files to close

See Also

:append-to path, *;.dump path table*, *;.read path*, *;.save path*, *:echo [-n] msg*, *echoln(value)*, *:export-session-to path*, *:open [–since] [–until] path*, *:pipe-line-to shell-cmd*, *:pipe-to shell-cmd*, *:redirect-to [path]*, *:write-csv-to [–anonymize] path*, *:write-json-cols-to [–anonymize] path*, *:write-json-to [–anonymize] path*, *:write-jsonlines-to [–all] [–anonymize] path*, *:write-raw-to [–view] [–anonymize] path*, *:write-screen-to [–anonymize] path*, *:write-table-to [–anonymize] path*, *:write-to [–anonymize] path*, *:write-view-to [–anonymize] path*, *:xopen path*

11.2.19 :comment *text*

Attach a comment to the focused log line. The comment will be displayed right below the log message it is associated with. The comment can contain Markdown directives for styling and linking.

Parameters

- **text*** — The comment text

Examples

To add the comment 'This is where it all went wrong' to the focused line:

```
:comment This is where it all went wrong
```

See Also

:annotate, *:clear-comment*, *:partition-name name*, *:tag tag*

11.2.20 :config *option [value]*

Read or write a configuration option

Parameters

- **option*** — The path to the option to read or write
- **value** — The value to write. If not given, the current value is returned

Examples

To read the configuration of the '/ui/clock-format' option:

```
:config /ui/clock-format
```

To set the '/ui/dim-text' option to 'false':

```
:config /ui/dim-text false
```

See Also

:reset-config option

11.2.21 :convert-time-to *zone*

Convert the focused timestamp to the given timezone

Parameters

- **zone*** — The timezone name

11.2.22 :create-logline-table *table-name*

Create an SQL table using the focused line of the log view as a template

Parameters

- **table-name*** — The name for the new table

Examples

To create a logline-style table named 'task_durations':

```
:create-logline-table task_durations
```

See Also

:create-search-table table-name [pattern], *:create-search-table table-name [pattern]*, *:write-csv-to [–anonymize] path*, *:write-json-cols-to [–anonymize] path*, *:write-json-to [–anonymize] path*, *:write-jsonlines-to [–all] [–anonymize] path*, *:write-raw-to [–view] [–anonymize] path*, *:write-screen-to [–anonymize] path*, *:write-table-to [–anonymize] path*, *:write-view-to [–anonymize] path*

11.2.23 :create-search-table *table-name [pattern]*

Create an SQL table based on a regex search

Parameters

- **table-name*** — The name of the table to create
- **pattern** — The regular expression used to capture the table columns. If not given, the current search pattern is used.

Examples

To create a table named 'task_durations' that matches log messages with the pattern 'duration=(?<duration>d+)':

```
:create-search-table task_durations duration=(?<duration>\d+)
```

See Also

:create-logline-table table-name, *:create-logline-table table-name*, *:delete-search-table table-name*, *:delete-search-table table-name*, *:write-csv-to [–anonymize] path*, *:write-json-cols-to [–anonymize] path*, *:write-json-to [–anonymize] path*, *:write-jsonlines-to [–all] [–anonymize] path*, *:write-raw-to [–view] [–anonymize] path*, *:write-screen-to [–anonymize] path*, *:write-table-to [–anonymize] path*, *:write-view-to [–anonymize] path*

11.2.24 :current-time

Print the current time in human-readable form and seconds since the epoch

11.2.25 :delete-filter *pattern*

Delete the filter created with ':filter-in' or ':filter-out'

Parameters

- **pattern*** — The regular expression to match

Examples

To delete the filter with the pattern 'last message repeated':

```
:delete-filter last message repeated
```

See Also

:filter-in pattern, *:filter-out pattern*, *:hide-lines-after date*, *:hide-lines-before date*, *:hide-unmarked-lines*, *:toggle-filtering*

11.2.26 :delete-logline-table *table-name*

Delete a table created with create-logline-table

Parameters

- **table-name*** — The name of the table to delete

Examples

To delete the logline-style table named 'task_durations':

```
:delete-logline-table task_durations
```

See Also

:create-logline-table table-name, *:create-logline-table table-name*, *:create-search-table table-name [pattern]*, *:create-search-table table-name [pattern]*, *:write-csv-to [–anonymize] path*, *:write-json-cols-to [–anonymize] path*, *:write-json-to [–anonymize] path*, *:write-jsonlines-to [–all] [–anonymize] path*, *:write-raw-to [–view] [–anonymize] path*, *:write-screen-to [–anonymize] path*, *:write-table-to [–anonymize] path*, *:write-view-to [–anonymize] path*

11.2.27 :delete-search-table *table-name*

Delete a search table

Parameters

- **table-name** — The name of the table to delete

Examples

To delete the search table named 'task_durations':

```
:delete-search-table task_durations
```

See Also

:create-logline-table table-name, *:create-logline-table table-name*, *:create-search-table table-name [pattern]*, *:create-search-table table-name [pattern]*, *:write-csv-to [–anonymize] path*, *:write-json-cols-to [–anonymize] path*, *:write-json-to [–anonymize] path*, *:write-jsonlines-to [–all] [–anonymize] path*, *:write-raw-to [–view] [–anonymize] path*, *:write-screen-to [–anonymize] path*, *:write-table-to [–anonymize] path*, *:write-view-to [–anonymize] path*

11.2.28 :delete-tags *tag*

Remove the given tags from all log lines

Parameters

- **tag** — The tags to delete

Examples

To remove the tags '#BUG123' and '#needs-review' from all log lines:

```
:delete-tags #BUG123 #needs-review
```

See Also

:annotate, *:comment text*, *:partition-name name*, *:tag tag*

11.2.29 :disable-breakpoint *point*

Disable a breakpoint for the given [<format>:]<file>:<line> tuples or the current line

Parameters

- **point** — The file and line number of the breakpoint

11.2.30 :disable-filter *pattern*

Disable a filter created with filter-in/filter-out

Parameters

- **pattern*** — The regular expression used in the filter command

Examples

To disable the filter with the pattern 'last message repeated':

```
:disable-filter last message repeated
```

See Also

:enable-filter pattern, *:filter-in pattern*, *:filter-out pattern*, *:hide-lines-after date*, *:hide-lines-before date*, *:hide-unmarked-lines*, *:toggle-filtering*

11.2.31 :disable-word-wrap

Disable word-wrapping for the current view

See Also

:clear-all-sticky-headers, *:enable-word-wrap*, *:hide-fields field-name*, *:set-text-view-mode mode*, *:toggle-sticky-header*

11.2.32 :echo *[-n] msg*

Echo the given message to the screen or, if :redirect-to has been called, to output file specified in the redirect. Variable substitution is performed on the message. Use a backslash to escape any special characters, like '$'

Parameters

- **-n** — Do not print a line-feed at the end of the output
- **msg*** — The message to display

Examples

To output 'Hello, World!':

```
:echo Hello, World!
```

See Also

:alt-msg msg, *:append-to path*, *:cd dir*, *;.dump path table*, *;.read path*, *;.save path*, *echoln(value)*, *:eval command*, *:export-session-to path*, *:export-session-to path*, *:external-access-login [app]*, *:external-access port api-key*, *:open [–since] [–until] path*, *:pipe-line-to shell-cmd*, *:pipe-to shell-cmd*, *:rebuild*, *:redirect-to [path]*, *:redirect-to [path]*, *:sh –name=<name> cmdline*, *:write-csv-to [–anonymize] path*, *:write-csv-to [–anonymize] path*, *:write-json-cols-to [–anonymize] path*, *:write-json-cols-to [–anonymize] path*, *:write-json-to [–anonymize] path*, *:write-json-to [–anonymize] path*, *:write-jsonlines-to [–all] [–anonymize] path*, *:write-jsonlines-to [–all] [–anonymize] path*, *:write-raw-to [–view] [–anonymize] path*, *:write-raw-to [–view] [–anonymize] path*, *:write-screen-to [–anonymize] path*, *:write-screen-to [–anonymize] path*, *:write-table-to [–anonymize] path*, *:write-table-to [–anonymize] path*, *:write-to [–anonymize] path*, *:write-to [–anonymize] path*, *:write-view-to [–anonymize] path*, *:write-view-to [–anonymize] path*, *:xopen path*

11.2.33 :enable-breakpoint *point*

Enable or create a breakpoint for the given [<format>:]<file>:<line> tuples or the current line

Parameters

- **point** — The file and line number of the breakpoint

11.2.34 :enable-filter *pattern*

Enable a previously created and disabled filter

Parameters

- **pattern*** — The regular expression used in the filter command

Examples

To enable the disabled filter with the pattern 'last message repeated':

```
:enable-filter last message repeated
```

See Also

:filter-in pattern, *:filter-out pattern*, *:hide-lines-after date*, *:hide-lines-before date*, *:hide-unmarked-lines*, *:toggle-filtering*

11.2.35 :enable-word-wrap

Enable word-wrapping for the current view

See Also

:clear-all-sticky-headers, *:disable-word-wrap*, *:hide-fields field-name*, *:set-text-view-mode mode*, *:toggle-sticky-header*

11.2.36 :eval *command*

Evaluate the given command/query after doing environment variable substitution

Parameters

- **command*** — The command or query to perform substitution on.

Examples

To substitute the table name from a variable:

```
:eval ;SELECT * FROM ${table}
```

See Also

:alt-msg msg, *:cd dir*, *:echo [-n] msg*, *:export-session-to path*, *:external-access-login [app]*, *:external-access port api-key*, *:rebuild*, *:redirect-to [path]*, *:sh –name=<name> cmdline*, *:write-csv-to [–anonymize] path*, *:write-json-cols-to [–anonymize] path*, *:write-json-to [–anonymize] path*, *:write-jsonlines-to [–all] [–anonymize] path*, *:write-raw-to [–view] [–anonymize] path*, *:write-screen-to [–anonymize] path*, *:write-table-to [–anonymize] path*, *:write-to [–anonymize] path*, *:write-view-to [–anonymize] path*

11.2.37 :export-session-to *path*

Export the current lnav state to an executable lnav script file that contains the commands needed to restore the current session

Parameters

- **path*** — The path to the file to write

See Also

:alt-msg msg, *:append-to path*, *:cd dir*, *;.dump path table*, *;.read path*, *;.save path*, *:echo [-n] msg*, *:echo [-n] msg*, *echoln(value)*, *:eval command*, *:external-access-login [app]*, *:external-access port api-key*, *:open [–since] [–until] path*, *:pipe-line-to shell-cmd*, *:pipe-to shell-cmd*, *:rebuild*, *:redirect-to [path]*, *:redirect-to [path]*, *:sh –name=<name> cmdline*, *:write-csv-to*

[–anonymize] path, *:write-csv-to [–anonymize] path*, *:write-json-cols-to [–anonymize] path*, *:write-json-cols-to [–anonymize] path*, *:write-json-to [–anonymize] path*, *:write-json-to [–anonymize] path*, *:write-jsonlines-to [–all] [–anonymize] path*, *:write-jsonlines-to [–all] [–anonymize] path*, *:write-raw-to [–view] [–anonymize] path*, *:write-raw-to [–view] [–anonymize] path*, *:write-screen-to [–anonymize] path*, *:write-screen-to [–anonymize] path*, *:write-table-to [–anonymize] path*, *:write-table-to [–anonymize] path*, *:write-to [–anonymize] path*, *:write-to [–anonymize] path*, *:write-view-to [–anonymize] path*, *:write-view-to [–anonymize] path*, *:xopen path*

11.2.38 :external-access *port api-key*

Open a port to give remote access to this lnav instance

Parameters

- **port*** — The port number to listen on
- **api-key*** — The API key

See Also

:alt-msg msg, *:cd dir*, *:echo [-n] msg*, *:eval command*, *:export-session-to path*, *:external-access-login [app]*, *:rebuild*, *:redirect-to [path]*, *:sh –name=<name> cmdline*, *:write-csv-to [–anonymize] path*, *:write-json-cols-to [–anonymize] path*, *:write-json-to [–anonymize] path*, *:write-jsonlines-to [–all] [–anonymize] path*, *:write-raw-to [–view] [–anonymize] path*, *:write-screen-to [–anonymize] path*, *:write-table-to [–anonymize] path*, *:write-to [–anonymize] path*, *:write-view-to [–anonymize] path*

11.2.39 :external-access-login *[app]*

Use the external-opener to open a URL that refers to lnav's external-access server

Parameters

- **app** — The app to launch

See Also

:alt-msg msg, *:cd dir*, *:echo [-n] msg*, *:eval command*, *:export-session-to path*, *:external-access port api-key*, *:rebuild*, *:redirect-to [path]*, *:sh –name=<name> cmdline*, *:write-csv-to [–anonymize] path*, *:write-json-cols-to [–anonymize] path*, *:write-json-to [–anonymize] path*, *:write-jsonlines-to [–all] [–anonymize] path*, *:write-raw-to [–view] [–anonymize] path*, *:write-screen-to [–anonymize] path*, *:write-table-to [–anonymize] path*, *:write-to [–anonymize] path*, *:write-view-to [–anonymize] path*

11.2.40 :filter-expr *expr*

Set the filter expression

Parameters

- **expr*** — The SQL expression to evaluate for each log message. The message values can be accessed using column names prefixed with a colon

Examples

To set a filter expression that matched syslog messages from 'syslogd':

```
:filter-expr :log_procname = 'syslogd'
```

To set a filter expression that matches log messages where 'id' is followed by a number and contains the string 'foo':

```
:filter-expr :log_body REGEXP 'id\d+' AND :log_body REGEXP 'foo'
```

See Also

:clear-filter-expr, *:filter-in pattern*, *:filter-out pattern*, *:hide-lines-after date*, *:hide-lines-before date*, *:hide-unmarked-lines*, *:toggle-filtering*

11.2.41 :filter-in *pattern*

Only show lines that match the given regular expression in the current view

Parameters

- **pattern*** — The regular expression to match

Examples

To filter out log messages that do not have the string 'dhclient':

```
:filter-in dhclient
```

See Also

:delete-filter pattern, *:disable-filter pattern*, *:filter-out pattern*, *:hide-lines-after date*, *:hide-lines-before date*, *:hide-unmarked-lines*, *:toggle-filtering*

11.2.42 :filter-out *pattern*

Remove lines that match the given regular expression in the current view

Parameters

- **pattern*** — The regular expression to match

Examples

To filter out log messages that contain the string 'last message repeated':

```
:filter-out last message repeated
```

See Also

:delete-filter pattern, *:disable-filter pattern*, *:filter-in pattern*, *:hide-lines-after date*, *:hide-lines-before date*, *:hide-unmarked-lines*, *:toggle-filtering*

11.2.43 :goto *line#|N%|timestamp|#anchor*

Go to the given location in the top view

Parameters

- **line#|N%|timestamp|#anchor*** — A line number, percent into the file, timestamp, or an anchor in a text file

Examples

To go to line 22:

```
:goto 22
```

To go to the line 75% of the way into the view:

```
:goto 75%
```

To go to the first message on the first day of 2017:

```
:goto 2017-01-01
```

To go to the Screenshots section:

```
:goto #screenshots
```

See Also

:next-location, *:next-mark type*, *:next-section*, *:prev-location*, *:prev-mark type*, *:prev-section*, *:relative-goto line-count|N%*

11.2.44 :help

Open the help text view

11.2.45 :hide-fields *field-name*

Hide log message fields by replacing them with an ellipsis

Parameters

- **field-name** — The name of the field to hide in the format for the focused log line. A qualified name can be used where the field name is prefixed by the format name and a dot to hide any field.

Examples

To hide the log_procname fields in all formats:

```
:hide-fields log_procname
```

To hide only the log_procname field in the syslog format:

```
:hide-fields syslog_log.log_procname
```

See Also

:clear-all-sticky-headers, *:enable-word-wrap*, *:set-text-view-mode mode*, *:show-fields field-name*, *:toggle-sticky-header*

11.2.46 :hide-file *path*

Hide the given file(s) and skip indexing until it is shown again. If no path is given, the current file in the view is hidden

Parameters

- **path** — A path or glob pattern that specifies the files to hide

---

11.2.47 :hide-in-timeline *row-type*

Hide rows of the given type(s) in the timeline view

Parameters

- **row-type** — The type of row to hide

Examples

To hide logfile and thread rows:

```
:hide-in-timeline logfile thread
```

See Also

:clear-all-sticky-headers, *:enable-word-wrap*, *:hide-fields field-name*, *:set-text-view-mode mode*, *:show-in-timeline row-type*, *:toggle-sticky-header*

11.2.48 :hide-lines-after *date*

Hide lines that come after the given date

Parameters

- **date*** — An absolute or relative date

Examples

To hide the lines after the focused line in the view:

```
:hide-lines-after here
```

To hide the lines after 6 AM today:

```
:hide-lines-after 6am
```

See Also

:filter-in pattern, *:filter-out pattern*, *:hide-lines-before date*, *:hide-unmarked-lines*, *:show-lines-before-and-after*, *:toggle-filtering*

11.2.49 :hide-lines-before *date*

Hide lines that come before the given date

Parameters

- **date*** — An absolute or relative date

Examples

To hide the lines before the focused line in the view:

```
:hide-lines-before here
```

To hide the log messages before 6 AM today:

```
:hide-lines-before 6am
```

See Also

:filter-in pattern, *:filter-out pattern*, *:hide-lines-after date*, *:hide-unmarked-lines*, *:show-lines-before-and-after*, *:toggle-filtering*

11.2.50 :hide-unmarked-lines

Hide lines that have not been bookmarked

See Also

:clear-all-sticky-headers, *:filter-in pattern*, *:filter-out pattern*, *:hide-lines-after date*, *:hide-lines-before date*, *:mark*, *:next-mark type*, *:prev-mark type*, *:toggle-filtering*, *:toggle-sticky-header*

11.2.51 :highlight *pattern*

Add coloring to log messages fragments that match the given regular expression

Parameters

- **pattern*** — The regular expression to match

Examples

To highlight numbers with three or more digits:

```
:highlight \d{3,}
```

See Also

:clear-all-sticky-headers, *:clear-highlight pattern*, *:enable-word-wrap*, *:hide-fields field-name*, *:set-text-view-mode mode*, *:toggle-sticky-header*

11.2.52 :highlight-field *[–color] [–bold] [–underline] [–italic] [–strike] [–blink] field [pattern]*

Highlight a field that matches the given pattern

Parameters

- **–color** — The foreground color to apply
- **–bold** — Make the text bold
- **–underline** — Underline the text
- **–italic** — Italicize the text
- **–strike** — Strikethrough the text

- **–blink** — Make the text blink
- **field*** — The name of the field to highlight
- **pattern** — The regular expression to match

Examples

To color status values that start with '2' green:

```
:highlight-field --color=green sc_status ^2.*
```

See Also

:clear-all-sticky-headers, *:clear-highlight-field field*, *:enable-word-wrap*, *:hide-fields field-name*, *:set-text-view-mode mode*, *:toggle-sticky-header*

11.2.53 :load-session

Load the latest session state

11.2.54 :mark

Toggle the bookmark state for the focused line in the current view

See Also

:clear-all-sticky-headers, *:hide-unmarked-lines*, *:next-mark type*, *:prev-mark type*, *:toggle-sticky-header*

11.2.55 :mark-expr *expr*

Set the bookmark expression

Parameters

- **expr*** — The SQL expression to evaluate for each log message. The message values can be accessed using column names prefixed with a colon

Examples

To mark lines from 'dhclient' that mention 'eth0':

```
:mark-expr :log_procname = 'dhclient' AND :log_body LIKE '%eth0%'
```

See Also

:clear-all-sticky-headers, *:clear-mark-expr*, *:hide-unmarked-lines*, *:mark*, *:next-mark type*, *:prev-mark type*, *:toggle-sticky-header*

11.2.56 :next-location

Move to the next position in the location history

See Also

:goto line#|N%|timestamp|#anchor, *:next-mark type*, *:next-section*, *:prev-location*, *:prev-mark type*, *:prev-section*, *:relative-goto line-count|N%*

11.2.57 :next-mark *type*

Move to the next bookmark of the given type in the current view

Parameters

- **type** — The type of bookmark – error, warning, search, user, file, meta

Examples

To go to the next error:

```
:next-mark error
```

See Also

:clear-all-sticky-headers, *:goto line#|N%|timestamp|#anchor*, *:hide-unmarked-lines*, *:mark*, *:next-location*, *:next-section*, *:prev-location*, *:prev-mark type*, *:prev-mark type*, *:prev-section*, *:relative-goto line-count|N%*, *:toggle-sticky-header*

11.2.58 :next-section

Move to the next section in the document

See Also

:goto line#|N%|timestamp|#anchor, *:next-location*, *:next-mark type*, *:prev-location*, *:prev-mark type*, *:prev-section*, *:relative-goto line-count|N%*

11.2.59 :open *[–since] [–until] path*

Open the given file(s) in lnav. Opening files on machines accessible via SSH can be done using the syntax: [user@]host:/path/to/logs

Parameters

- **–since** — The low cutoff time
- **–until** — The high cutoff time
- **path** — The path to the file to open

Examples

To open the file '/path/to/file':

```
:open /path/to/file
```

To open the remote file '/var/log/syslog.log':

```
:open dean@host1.example.com:/var/log/syslog.log
```

See Also

:append-to path, *:close path*, *;.dump path table*, *;.read path*, *;.save path*, *:echo [-n] msg*, *echoln(value)*, *:export-session-to path*, *:pipe-line-to shell-cmd*, *:pipe-to shell-cmd*, *:redirect-to [path]*, *:write-csv-to [–anonymize] path*, *:write-json-cols-to [–anonymize] path*, *:write-json-to [–anonymize] path*, *:write-jsonlines-to [–all] [–anonymize] path*, *:write-raw-to [–view] [–anonymize] path*, *:write-screen-to [–anonymize] path*, *:write-table-to [–anonymize] path*, *:write-to [–anonymize] path*, *:write-view-to [–anonymize] path*, *:xopen path*

11.2.60 :partition-name *name*

Mark the focused line in the log view as the start of a new partition with the given name

Parameters

- **name*** — The name for the new partition

Examples

To mark the focused line as the start of the partition named 'boot #1':

```
:partition-name boot #1
```

See Also

:annotate, *:clear-partition*, *:comment text*, *:tag tag*

11.2.61 :pipe-line-to *shell-cmd*

Pipe the focused line to the given shell command. Any fields defined by the format will be set as environment variables.

Parameters

- **shell-cmd*** — The shell command-line to execute

Examples

To write the focused line to 'sed' for processing:

```
:pipe-line-to sed -e 's/foo/bar/g'
```

See Also

:append-to path, *;.dump path table*, *;.read path*, *;.save path*, *:echo [-n] msg*, *echoln(value)*, *:export-session-to path*, *:open [–since] [–until] path*, *:pipe-to shell-cmd*, *:redirect-to [path]*, *:write-csv-to [–anonymize] path*, *:write-json-cols-to [–anonymize] path*, *:write-json-to [–anonymize] path*, *:write-jsonlines-to [–all] [–anonymize] path*, *:write-raw-to [–view] [–anonymize] path*, *:write-screen-to [–anonymize] path*, *:write-table-to [–anonymize] path*, *:write-to [–anonymize] path*, *:write-view-to [–anonymize] path*, *:xopen path*

11.2.62 :pipe-to *shell-cmd*

Pipe the marked lines to the given shell command

Parameters

- **shell-cmd*** — The shell command-line to execute

Examples

To write marked lines to 'sed' for processing:

```
:pipe-to sed -e s/foo/bar/g
```

See Also

:append-to path, *;.dump path table*, *;.read path*, *;.save path*, *:echo [-n] msg*, *echoln(value)*, *:export-session-to path*, *:open [–since] [–until] path*, *:pipe-line-to shell-cmd*, *:redirect-to [path]*, *:write-csv-to [–anonymize] path*, *:write-json-cols-to [–anonymize] path*, *:write-json-to [–anonymize] path*,

:write-jsonlines-to [–all] [–anonymize] path, *:write-raw-to [–view] [–anonymize] path*, *:write-screen-to [–anonymize] path*, *:write-table-to [–anonymize] path*, *:write-to [–anonymize] path*, *:write-view-to [–anonymize] path*, *:xopen path*

11.2.63 :prev-location

Move to the previous position in the location history

See Also

:goto line#|N%|timestamp|#anchor, *:next-location*, *:next-mark type*, *:next-section*, *:prev-mark type*, *:prev-section*, *:relative-goto line-count|N%*

11.2.64 :prev-mark *type*

Move to the previous bookmark of the given type in the current view

Parameters

- **type** — The type of bookmark – error, warning, search, user, file, meta

Examples

To go to the previous error:

```
:prev-mark error
```

See Also

:clear-all-sticky-headers, *:goto line#|N%|timestamp|#anchor*, *:hide-unmarked-lines*, *:mark*, *:next-location*, *:next-mark type*, *:next-mark type*, *:next-section*, *:prev-location*, *:prev-section*, *:relative-goto line-count|N%*, *:toggle-sticky-header*

11.2.65 :prev-section

Move to the previous section in the document

See Also

:goto line#|N%|timestamp|#anchor, *:next-location*, *:next-mark type*, *:next-section*, *:prev-location*, *:prev-mark type*, *:relative-goto line-count|N%*

11.2.66 :prompt *type [–alt] [prompt] [initial-value]*

Open the given prompt

Parameters

- **type*** — The type of prompt
- **–alt** — Perform the alternate action for this prompt by default
- **prompt** — The prompt to display
- **initial-value** — The initial value to fill in for the prompt

Examples

To open the command prompt with 'filter-in' already filled in:

```
:prompt command : 'filter-in '
```

To ask the user a question:

```
:prompt user 'Are you sure? '
```

11.2.67 :quit

Quit lnav

11.2.68 :rebuild

Forcefully rebuild file indexes

See Also

:alt-msg msg, *:cd dir*, *:echo [-n] msg*, *:eval command*, *:export-session-to path*, *:external-access-login [app]*, *:external-access port api-key*, *:redirect-to [path]*, *:sh –name=<name> cmdline*, *:write-csv-to [–anonymize] path*, *:write-json-cols-to [–anonymize] path*, *:write-json-to [–anonymize] path*, *:write-jsonlines-to [–all] [–anonymize] path*, *:write-raw-to [–view] [–anonymize] path*, *:write-screen-to [–anonymize] path*, *:write-table-to [–anonymize] path*, *:write-to [–anonymize] path*, *:write-view-to [–anonymize] path*

11.2.69 :redirect-to *[path]*

Redirect the output of commands that write to stdout to the given file

Parameters

- **path** — The path to the file to write. If not specified, the current redirect will be cleared

Examples

To write the output of lnav commands to the file /tmp/script-output.txt:

```
:redirect-to /tmp/script-output.txt
```

See Also

:alt-msg msg, *:append-to path*, *:cd dir*, *;.dump path table*, *;.read path*, *;.save path*, *:echo [-n] msg*, *:echo [-n] msg*, *echoln(value)*, *:eval command*, *:export-session-to path*, *:export-session-to path*, *:external-access-login [app]*, *:external-access port api-key*, *:open [–since] [–until] path*, *:pipe-line-to shell-cmd*, *:pipe-to shell-cmd*, *:rebuild*, *:sh –name=<name> cmdline*, *:write-csv-to [–anonymize] path*, *:write-csv-to [–anonymize] path*, *:write-json-cols-to [–anonymize] path*, *:write-json-cols-to [–anonymize] path*, *:write-json-to [–anonymize] path*, *:write-json-to [–anonymize] path*, *:write-jsonlines-to [–all] [–anonymize] path*, *:write-jsonlines-to [–all] [–anonymize] path*, *:write-raw-to [–view] [–anonymize] path*, *:write-raw-to [–view] [–anonymize] path*, *:write-screen-to [–anonymize] path*, *:write-screen-to [–anonymize] path*, *:write-table-to [–anonymize] path*, *:write-table-to [–anonymize] path*, *:write-to [–anonymize] path*, *:write-to [–anonymize] path*, *:write-view-to [–anonymize] path*, *:write-view-to [–anonymize] path*, *:xopen path*

11.2.70 :redraw

Do a full redraw of the screen

11.2.71 :relative-goto *line-count|N%*

Move the current view up or down by the given amount

Parameters

- **line-count|N%*** — The amount to move the view by.

Examples

To move 22 lines down in the view:

```
:relative-goto +22
```

To move 10 percent back in the view:

```
:relative-goto -10%
```

See Also

:goto line#|N%|timestamp|#anchor, *:next-location*, *:next-mark type*, *:next-section*, *:prev-location*, *:prev-mark type*, *:prev-section*

11.2.72 :reset-config *option*

Reset the configuration option to its default value

Parameters

- **option*** — The path to the option to reset

Examples

To reset the '/ui/clock-format' option back to the builtin default:

```
:reset-config /ui/clock-format
```

See Also

:config option [value]

11.2.73 :reset-session

Reset the session state, clearing all filters, highlights, and bookmarks

---

11.2.74 :save-session

Save the current state as a session

11.2.75 :session *lnav-command*

Add the given command to the session file (~/.lnav/session)

Parameters

- **lnav-command*** — The lnav command to save.

Examples

To add the command ':highlight foobar' to the session file:

```
:session :highlight foobar
```

11.2.76 :set-file-timezone *zone [pattern]*

Set the timezone to use for log messages that do not include a timezone. The timezone is applied to the focused file or the given glob pattern.

Parameters

- **zone*** — The timezone name
- **pattern** — The glob pattern to match against files that should use this timezone

See Also

:clear-file-timezone pattern

11.2.77 :set-min-log-level *log-level*

Set the minimum log level to display in the log view

Parameters

- **log-level*** — The new minimum log level

Examples

To set the minimum log level displayed to error:

```
:set-min-log-level error
```

11.2.78 :set-text-view-mode *mode*

Set the display mode for text files

Parameters

- **mode*** — The display mode

See Also

:clear-all-sticky-headers, *:enable-word-wrap*, *:hide-fields field-name*, *:toggle-sticky-header*

11.2.79 :sh *–name=<name> cmdline*

Execute the given command-line and display the captured output

Parameters

- **–name=<name>*** — The name to give to the captured output
- **cmdline*** — The command-line to execute.

See Also

:alt-msg msg, *:cd dir*, *:echo [-n] msg*, *:eval command*, *:export-session-to path*, *:external-access-login [app]*, *:external-access port api-key*, *:rebuild*, *:redirect-to [path]*, *:write-csv-to [–anonymize] path*, *:write-json-cols-to [–anonymize] path*, *:write-json-to [–anonymize] path*, *:write-jsonlines-to [–all] [–anonymize] path*, *:write-raw-to [–view] [–anonymize] path*, *:write-screen-to [–anonymize] path*, *:write-table-to [–anonymize] path*, *:write-to [–anonymize] path*, *:write-view-to [–anonymize] path*

11.2.80 :show-fields *field-name*

Show log message fields that were previously hidden

Parameters

- **field-name** — The name of the field to show

Examples

To show all the log_procname fields in all formats:

```
:show-fields log_procname
```

See Also

:clear-all-sticky-headers, *:enable-word-wrap*, *:hide-fields field-name*, *:set-text-view-mode mode*, *:toggle-sticky-header*

11.2.81 :show-file *path*

Show the given file(s) and resume indexing.

Parameters

- **path** — The path or glob pattern that specifies the files to show

---

11.2.82 :show-in-timeline *row-type*

Show rows of the given type(s) that were previously hidden in the timeline view

Parameters

- **row-type** — The type of row to show

Examples

To show logfile and thread rows:

```
:show-in-timeline logfile thread
```

See Also

:clear-all-sticky-headers, *:enable-word-wrap*, *:hide-fields field-name*, *:hide-in-timeline row-type*, *:set-text-view-mode mode*, *:toggle-sticky-header*

11.2.83 :show-lines-before-and-after

Show lines that were hidden by the 'hide-lines' commands

See Also

:filter-in pattern, *:filter-out pattern*, *:hide-lines-after date*, *:hide-lines-before date*, *:hide-unmarked-lines*, *:toggle-filtering*

11.2.84 :show-only-this-file

Show only the file for the focused line in the view

11.2.85 :show-unmarked-lines

Show lines that have not been bookmarked

See Also

:clear-all-sticky-headers, *:filter-in pattern*, *:filter-out pattern*, *:hide-lines-after date*, *:hide-lines-before date*, *:hide-unmarked-lines*, *:hide-unmarked-lines*, *:mark*, *:next-mark type*, *:prev-mark type*, *:toggle-filtering*, *:toggle-sticky-header*

11.2.86 :spectrogram *field-name*

Visualize the given message field or database column using a spectrogram

Parameters

- **field-name*** — The name of the numeric field to visualize.

Examples

To visualize the sc_bytes field in the access_log format:

```
:spectrogram sc_bytes
```

11.2.87 :summarize *column-name*

Execute a SQL query that computes the characteristics of the values in the given column

Parameters

- **column-name*** — The name of the column to analyze.

Examples

To get a summary of the sc_bytes column in the access_log table:

```
:summarize sc_bytes
```

11.2.88 :switch-to-view *view-name*

Switch to the given view

Parameters

- **view-name*** — The name of the view to switch to.

Examples

To switch to the 'schema' view:

```
:switch-to-view schema
```

11.2.89 :tag *tag*

Attach tags to the focused log line

Parameters

- **tag** — The tags to attach

Examples

To add the tags '#BUG123' and '#needs-review' to the focused line:

```
:tag #BUG123 #needs-review
```

See Also

:annotate, *:comment text*, *:delete-tags tag*, *:partition-name name*, *:untag tag*

11.2.90 :toggle-breakpoint

Toggle a breakpoint for the focused line in the LOG view

11.2.91 :toggle-filtering

Toggle the filtering flag for the current view

See Also

:filter-in pattern, *:filter-out pattern*, *:hide-lines-after date*, *:hide-lines-before date*, *:hide-unmarked-lines*

11.2.92 :toggle-sticky-header

Toggle the sticky header state for the focused line in the current view

See Also

:clear-all-sticky-headers, *:clear-all-sticky-headers*, *:enable-word-wrap*, *:hide-fields field-name*, *:hide-unmarked-lines*, *:mark*, *:next-mark type*, *:prev-mark type*, *:set-text-view-mode mode*

11.2.93 :toggle-view *view-name*

Switch to the given view or, if it is already displayed, switch to the previous view

Parameters

- **view-name*** — The name of the view to toggle the display of.

Examples

To switch to the 'schema' view if it is not displayed or switch back to the previous view:

```
:toggle-view schema
```

11.2.94 :unix-time *seconds*

Convert epoch time to a human-readable form

Parameters

- **seconds*** — The epoch timestamp to convert

Examples

To convert the epoch time 1490191111:

```
:unix-time 1490191111
```

11.2.95 :untag *tag*

Detach tags from the focused log line

Parameters

- **tag** — The tags to detach

Examples

To remove the tags '#BUG123' and '#needs-review' from the focused line:

```
:untag #BUG123 #needs-review
```

See Also

:annotate, *:comment text*, *:partition-name name*, *:tag tag*

11.2.96 :write-table-to *[–anonymize] path*

Write SQL results to the given file in a tabular format

Parameters

- **–anonymize** — Anonymize the table contents
- **path*** — The path to the file to write

Examples

To write SQL results as text to /tmp/table.txt:

```
:write-table-to /tmp/table.txt
```

See Also

:alt-msg msg, *:append-to path*, *:cd dir*, *:create-logline-table table-name*, *:create-search-table table-name [pattern]*, *;.dump path table*, *;.read path*, *;.save path*, *:echo [-n] msg*, *:echo [-n] msg*, *echoln(value)*, *:eval command*, *:export-session-to path*, *:export-session-to path*, *:external-access-login [app]*, *:external-access port api-key*, *:open [–since] [–until] path*, *:pipe-line-to shell-cmd*, *:pipe-to shell-cmd*, *:rebuild*, *:redirect-to [path]*, *:redirect-to [path]*, *:sh –name=<name> cmd-line*, *:write-csv-to [–anonymize] path*, *:write-csv-to [–anonymize] path*, *:write-csv-to [–anonymize] path*, *:write-json-cols-to [–anonymize] path*, *:write-json-cols-to [–anonymize] path*, *:write-json-cols-to [–anonymize] path*, *:write-json-to [–anonymize] path*, *:write-json-to [–anonymize] path*, *:write-json-to [–anonymize] path*, *:write-jsonlines-to [–all] [–anonymize] path*, *:write-jsonlines-to [–all] [–anonymize] path*, *:write-jsonlines-to [–all] [–anonymize] path*, *:write-raw-to [–view] [–anonymize] path*, *:write-raw-to [–view] [–anonymize] path*, *:write-raw-to [–view] [–anonymize] path*, *:write-screen-to [–anonymize] path*, *:write-screen-to [–anonymize] path*, *:write-screen-to [–anonymize] path*, *:write-to [–anonymize] path*, *:write-to [–anonymize] path*, *:write-view-to [–anonymize] path*, *:write-view-to [–anonymize] path*, *:write-view-to [–anonymize] path*, *:xopen path*

11.2.97 :write-csv-to *[–anonymize] path*

Write SQL results to the given file in CSV format

Parameters

- **–anonymize** — Anonymize the row contents
- **path*** — The path to the file to write

Examples

To write SQL results as CSV to /tmp/table.csv:

```
:write-csv-to /tmp/table.csv
```

See Also

:alt-msg msg, *:append-to path*, *:cd dir*, *:create-logline-table table-name*, *:create-search-table table-name [pattern]*, *;.dump path table*, *;.read path*, *;.save path*, *:echo [-n] msg*, *:echo [-n] msg*, *echoln(value)*, *:eval command*, *:export-session-to path*, *:export-session-to path*, *:external-access-login [app]*, *:external-access port api-key*, *:open [–since] [–until] path*, *:pipe-line-to shell-cmd*, *:pipe-to shell-cmd*, *:rebuild*, *:redirect-to [path]*, *:redirect-to [path]*, *:sh –name=<name> cmd-line*, *:write-json-cols-to [–anonymize] path*, *:write-json-cols-to [–anonymize] path*, *:write-json-cols-to [–anonymize] path*, *:write-json-to [–anonymize] path*, *:write-json-to [–anonymize] path*, *:write-json-to [–anonymize] path*, *:write-jsonlines-to [–all] [–anonymize] path*, *:write-jsonlines-to [–all] [–anonymize] path*, *:write-jsonlines-to [–all] [–anonymize] path*, *:write-raw-to [–view] [–anonymize] path*, *:write-raw-to [–view] [–anonymize] path*, *:write-raw-to [–view] [–anonymize] path*, *:write-screen-to [–anonymize] path*, *:write-screen-to [–anonymize] path*, *:write-screen-to [–anonymize] path*, *:write-table-to [–anonymize] path*, *:write-table-to [–anonymize] path*, *:write-table-to [–anonymize] path*, *:write-to [–anonymize] path*, *:write-to [–anonymize] path*, *:write-view-to [–anonymize] path*, *:write-view-to [–anonymize] path*, *:write-view-to [–anonymize] path*, *:xopen path*

11.2.98 :write-debug-log-to *path*

Write lnav's internal debug log to the given path. This can be useful if the *-d* flag was not passed on the command line

Parameters

- **path*** — The destination path for the debug log

11.2.99 :write-json-cols-to *[–anonymize] path*

Write SQL results to the given file in a column-oriented JSON format. In addition, columns that contain JSON values will be flattened to their own columns. For example, a column containing values shaped like *{"a": 1, "b": 2}* will be split into two separate columns named 'a' and 'b'. This format can be useful for feeding into charting libraries.

Parameters

- **–anonymize** — Anonymize the JSON values
- **path*** — The path to the file to write

Examples

To write SQL results as JSON to /tmp/table.json:

```
:write-json-cols-to /tmp/table.json
```

See Also

:alt-msg msg, *:append-to path*, *:cd dir*, *:create-logline-table table-name*, *:create-search-table table-name [pattern]*, *;.dump path table*, *;.read path*, *;.save path*, *:echo [-n] msg*, *:echo [-n] msg*, *echoln(value)*, *:eval command*, *:export-session-to path*, *:export-session-to path*, *:external-access-login [app]*, *:external-access port api-key*, *:open [–since] [–until] path*, *:pipe-line-to shell-cmd*, *:pipe-to shell-cmd*, *:rebuild*, *:redirect-to [path]*, *:redirect-to [path]*, *:sh –name=<name> cmdline*, *:write-csv-to [–anonymize] path*, *:write-csv-to [–anonymize] path*, *:write-csv-to [–anonymize] path*, *:write-json-to [–anonymize] path*, *:write-json-to [–anonymize] path*, *:write-json-to [–anonymize] path*, *:write-jsonlines-to [–all] [–anonymize] path*, *:write-jsonlines-to [–all] [–anonymize] path*, *:write-jsonlines-to [–all] [–anonymize] path*, *:write-raw-to [–view] [–anonymize] path*, *:write-raw-to [–view] [–anonymize] path*, *:write-raw-to [–view] [–anonymize] path*, *:write-screen-to [–anonymize] path*, *:write-screen-to [–anonymize] path*, *:write-screen-to [–anonymize] path*, *:write-table-to [–anonymize] path*, *:write-table-to [–anonymize] path*, *:write-table-to [–anonymize] path*, *:write-to [–anonymize] path*, *:write-to [–anonymize] path*, *:write-view-to [–anonymize] path*, *:write-view-to [–anonymize] path*, *:write-view-to [–anonymize] path*, *:xopen path*

11.2.100 :write-json-to *[–anonymize] path*

Write SQL results to the given file in JSON format

Parameters

- **–anonymize** — Anonymize the JSON values
- **path*** — The path to the file to write

Examples

To write SQL results as JSON to /tmp/table.json:

```
:write-json-to /tmp/table.json
```

See Also

:alt-msg msg, *:append-to path*, *:cd dir*, *:create-logline-table table-name*, *:create-search-table table-name [pattern]*, *;.dump path table*, *;.read path*, *;.save path*, *:echo [-n] msg*, *:echo [-n] msg*, *echoln(value)*, *:eval command*, *:export-session-to path*, *:export-session-to path*, *:external-access-login [app]*, *:external-access port api-key*, *:open [–since] [–until] path*, *:pipe-line-to shell-cmd*, *:pipe-to shell-cmd*, *:rebuild*, *:redirect-to [path]*, *:redirect-to [path]*, *:sh –name=<name> cmd-line*, *:write-csv-to [–anonymize] path*, *:write-csv-to [–anonymize] path*, *:write-csv-to [–anonymize] path*, *:write-json-cols-to [–anonymize] path*, *:write-json-cols-to [–anonymize] path*, *:write-json-cols-to [–anonymize] path*, *:write-jsonlines-to [–all] [–anonymize] path*, *:write-jsonlines-to [–all] [–anonymize] path*, *:write-jsonlines-to [–all] [–anonymize] path*, *:write-raw-to [–view] [–anonymize] path*, *:write-raw-to [–view] [–anonymize] path*, *:write-raw-to [–view] [–anonymize] path*, *:write-screen-to [–anonymize] path*, *:write-screen-to [–anonymize] path*, *:write-screen-to [–anonymize] path*, *:write-table-to [–anonymize] path*, *:write-table-to [–anonymize] path*, *:write-table-to [–anonymize] path*, *:write-to [–anonymize] path*, *:write-to [–anonymize] path*, *:write-view-to [–anonymize] path*, *:write-view-to [–anonymize] path*, *:write-view-to [–anonymize] path*, *:xopen path*

11.2.101 :write-jsonlines-to *[–all] [–anonymize] path*

Write SQL results or log lines to the given file in JSON Lines format

Parameters

- **–all** — Write all visible log lines instead of only marked lines
- **–anonymize** — Anonymize the JSON values
- **path*** — The path to the file to write

Examples

To write SQL results as JSON Lines to /tmp/table.json:

```
:write-jsonlines-to /tmp/table.json
```

See Also

:alt-msg msg, *:append-to path*, *:cd dir*, *:create-logline-table table-name*, *:create-search-table table-name [pattern]*, *;.dump path table*, *;.read path*, *;.save path*, *:echo [-n] msg*, *:echo [-n] msg*, *echoln(value)*, *:eval command*, *:export-session-to path*, *:export-session-to path*, *:external-access-login [app]*, *:external-access port api-key*, *:open [–since] [–until] path*, *:pipe-line-to shell-cmd*, *:pipe-to shell-cmd*, *:rebuild*, *:redirect-to [path]*, *:redirect-to [path]*, *:sh –name=<name> cmd-line*, *:write-csv-to [–anonymize] path*, *:write-csv-to [–anonymize] path*, *:write-csv-to [–anonymize] path*, *:write-json-cols-to [–anonymize] path*, *:write-json-cols-to [–anonymize] path*, *:write-json-cols-to [–anonymize] path*, *:write-json-to [–anonymize] path*, *:write-json-to [–anonymize] path*, *:write-json-to [–anonymize] path*, *:write-raw-to [–view] [–anonymize] path*, *:write-raw-to [–view] [–anonymize] path*, *:write-raw-to [–view] [–anonymize] path*, *:write-screen-to [–anonymize] path*, *:write-screen-to [–anonymize] path*, *:write-screen-to [–anonymize] path*, *:write-table-to [–anonymize] path*, *:write-table-to [–anonymize] path*, *:write-table-to [–anonymize] path*, *:write-to [–anonymize] path*, *:write-to [–anonymize] path*, *:write-view-to [–anonymize] path*, *:write-view-to [–anonymize] path*, *:write-view-to [–anonymize] path*, *:xopen path*

11.2.102 :write-raw-to *[–view] [–anonymize] path*

In the log view, write the original log file content of the marked messages to the file. In the DB view, the contents of the cells are written to the output file.

Parameters

- **–view** — The view to use as the source of data
- **–anonymize** — Anonymize the lines
- **path*** — The path to the file to write

Examples

To write the marked lines in the log view to /tmp/table.txt:

```
:write-raw-to /tmp/table.txt
```

See Also

:alt-msg msg, *:append-to path*, *:cd dir*, *:create-logline-table table-name*, *:create-search-table table-name [pattern]*, *;.dump path table*, *;.read path*, *;.save path*, *:echo [-n] msg*, *:echo [-n] msg*, *echoln(value)*, *:eval command*, *:export-session-to path*, *:export-session-to path*, *:external-access-login [app]*, *:external-access port api-key*, *:open [–since] [–until] path*, *:pipe-line-to shell-cmd*, *:pipe-to shell-cmd*, *:rebuild*, *:redirect-to [path]*, *:redirect-to [path]*, *:sh –name=<name> cmdline*, *:write-csv-to [–anonymize] path*, *:write-csv-to [–anonymize] path*, *:write-csv-to [–anonymize] path*, *:write-json-cols-to [–anonymize] path*, *:write-json-cols-to [–anonymize] path*, *:write-json-cols-to [–anonymize] path*, *:write-json-to [–anonymize] path*, *:write-json-to [–anonymize] path*, *:write-json-to [–anonymize] path*, *:write-jsonlines-to [–all] [–anonymize] path*, *:write-jsonlines-to [–all] [–anonymize] path*, *:write-jsonlines-to [–all] [–anonymize] path*, *:write-screen-to [–anonymize] path*, *:write-screen-to [–anonymize] path*, *:write-screen-to [–anonymize] path*, *:write-table-to [–anonymize] path*, *:write-table-to [–anonymize] path*, *:write-table-to [–anonymize] path*, *:write-to [–anonymize] path*, *:write-to [–anonymize] path*, *:write-view-to [–anonymize] path*, *:write-view-to [–anonymize] path*, *:write-view-to [–anonymize] path*, *:xopen path*

11.2.103 :write-screen-to *[–anonymize] path*

Write the displayed text or SQL results to the given file without any formatting

Parameters

- **–anonymize** — Anonymize the lines
- **path*** — The path to the file to write

Examples

To write only the displayed text to /tmp/table.txt:

```
:write-screen-to /tmp/table.txt
```

See Also

:alt-msg msg, *:append-to path*, *:cd dir*, *:create-logline-table table-name*, *:create-search-table table-name [pattern]*, *;.dump path table*, *;.read path*, *;.save path*, *:echo [-n] msg*, *:echo [-n] msg*, *echoln(value)*, *:eval command*, *:export-session-to path*, *:export-session-to path*, *:external-access-login [app]*, *:external-access port api-key*, *:open [–since] [–until] path*, *:pipe-line-to shell-cmd*, *:pipe-to shell-cmd*, *:rebuild*, *:redirect-to [path]*, *:redirect-to [path]*, *:sh –name=<name> cmdline*, *:write-csv-to [–anonymize] path*, *:write-csv-to [–anonymize] path*, *:write-csv-to [–anonymize] path*, *:write-json-cols-to [–anonymize] path*, *:write-json-cols-to [–anonymize] path*, *:write-json-cols-to [–anonymize] path*, *:write-json-to [–anonymize] path*, *:write-json-to [–anonymize]*

path, *:write-json-to [–anonymize] path*, *:write-jsonlines-to [–all] [–anonymize] path*, *:write-jsonlines-to [–all] [–anonymize] path*, *:write-jsonlines-to [–all] [–anonymize] path*, *:write-raw-to [–view] [–anonymize] path*, *:write-raw-to [–view] [–anonymize] path*, *:write-raw-to [–view] [–anonymize] path*, *:write-table-to [–anonymize] path*, *:write-table-to [–anonymize] path*, *:write-table-to [–anonymize] path*, *:write-to [–anonymize] path*, *:write-to [–anonymize] path*, *:write-view-to [–anonymize] path*, *:write-view-to [–anonymize] path*, *:write-view-to [–anonymize] path*, *:xopen path*

11.2.104 :write-to *[–anonymize] path*

Overwrite the given file with any marked lines in the current view

Parameters

- **–anonymize** — Anonymize the lines
- **path*** — The path to the file to write

Examples

To write marked lines to the file /tmp/interesting-lines.txt:

```
:write-to /tmp/interesting-lines.txt
```

See Also

:alt-msg msg, *:append-to path*, *:cd dir*, *;.dump path table*, *;.read path*, *;.save path*, *:echo [-n] msg*, *:echo [-n] msg*, *echoln(value)*, *:eval command*, *:export-session-to path*, *:export-session-to path*, *:external-access-login [app]*, *:external-access port api-key*, *:open [–since] [–until] path*, *:pipe-line-to shell-cmd*, *:pipe-to shell-cmd*, *:rebuild*, *:redirect-to [path]*, *:redirect-to [path]*, *:sh –name=<name> cmdline*, *:write-csv-to [–anonymize] path*, *:write-csv-to [–anonymize] path*, *:write-json-cols-to [–anonymize] path*, *:write-json-cols-to [–anonymize] path*, *:write-json-to [–anonymize] path*, *:write-json-to [–anonymize] path*, *:write-jsonlines-to [–all] [–anonymize] path*, *:write-jsonlines-to [–all] [–anonymize] path*, *:write-raw-to [–view] [–anonymize] path*, *:write-raw-to [–view] [–anonymize] path*, *:write-screen-to [–anonymize] path*, *:write-screen-to [–anonymize] path*, *:write-table-to [–anonymize] path*, *:write-table-to [–anonymize] path*, *:write-view-to [–anonymize] path*, *:write-view-to [–anonymize] path*, *:xopen path*

11.2.105 :write-view-to *[–anonymize] path*

Write the text in the top view to the given file without any formatting

Parameters

- **–anonymize** — Anonymize the lines
- **path*** — The path to the file to write

Examples

To write the top view to /tmp/table.txt:

```
:write-view-to /tmp/table.txt
```

See Also

:alt-msg msg, *:append-to path*, *:cd dir*, *:create-logline-table table-name*, *:create-search-table table-name [pattern]*, *;.dump path table*, *;.read path*, *;.save path*, *:echo [-n] msg*, *:echo [-n] msg*,

echoln(value), *:eval command*, *:export-session-to path*, *:export-session-to path*, *:external-access-login [app]*, *:external-access port api-key*, *:open [–since] [–until] path*, *:pipe-line-to shell-cmd*, *:pipe-to shell-cmd*, *:rebuild*, *:redirect-to [path]*, *:redirect-to [path]*, *:sh –name=<name> cmd-line*, *:write-csv-to [–anonymize] path*, *:write-csv-to [–anonymize] path*, *:write-csv-to [–anonymize] path*, *:write-json-cols-to [–anonymize] path*, *:write-json-cols-to [–anonymize] path*, *:write-json-cols-to [–anonymize] path*, *:write-json-to [–anonymize] path*, *:write-json-to [–anonymize] path*, *:write-json-to [–anonymize] path*, *:write-jsonlines-to [–all] [–anonymize] path*, *:write-jsonlines-to [–all] [–anonymize] path*, *:write-jsonlines-to [–all] [–anonymize] path*, *:write-raw-to [–view] [–anonymize] path*, *:write-raw-to [–view] [–anonymize] path*, *:write-raw-to [–view] [–anonymize] path*, *:write-screen-to [–anonymize] path*, *:write-screen-to [–anonymize] path*, *:write-screen-to [–anonymize] path*, *:write-table-to [–anonymize] path*, *:write-table-to [–anonymize] path*, *:write-table-to [–anonymize] path*, *:write-to [–anonymize] path*, *:write-to [–anonymize] path*, *:xopen path*

11.2.106 :xopen *path*

Use an external command to open the given file(s)

Parameters

- **path** — The path to the file to open

Examples

To open the file '/path/to/file':

```
:xopen /path/to/file
```

See Also

:append-to path, *;.dump path table*, *;.read path*, *;.save path*, *:echo [-n] msg*, *echoln(value)*, *:export-session-to path*, *:open [–since] [–until] path*, *:pipe-line-to shell-cmd*, *:pipe-to shell-cmd*, *:redirect-to [path]*, *:write-csv-to [–anonymize] path*, *:write-json-cols-to [–anonymize] path*, *:write-json-to [–anonymize] path*, *:write-jsonlines-to [–all] [–anonymize] path*, *:write-raw-to [–view] [–anonymize] path*, *:write-screen-to [–anonymize] path*, *:write-table-to [–anonymize] path*, *:write-to [–anonymize] path*, *:write-view-to [–anonymize] path*

11.2.107 :zoom-to *zoom-level*

Zoom the histogram view to the given level

Parameters

- **zoom-level*** — The zoom level

Examples

To set the zoom level to '1-week':

```
:zoom-to 1-week
```

CHAPTER

TWELVE

SQLITE INTERFACE

Log analysis in **lnav** can be done using the SQLite interface. Log messages can be accessed via virtual tables that are created for each file format. The tables have the same name as the log format and each message is its own row in the table. For example, given the following log message from an Apache access log:

```
127.0.0.1 - frank [10/Oct/2000:13:55:36 -0700] "GET /apache_pb.gif HTTP/1.0" 200 2326
```

These columns would be available for its row in the `access_log` table:

log_	log_	log_	log_	log_	log_	log_	log_	log_	c_ip	cs_r	cs_r	cs_ι	cs_ι	cs_ι	cs_ι	cs_ν	sc_b	sc_status
0	<NU	2000 10-10 13:5	0	info	1	<NU	<NU	<NU	127.(	GET	<NU	<NU	/apac	<NU	frank	HTT	2326	200

Note

Some columns are hidden by default to reduce the amount of noise in results, but they can still be accessed when explicitly used. The hidden columns are: `log_path`, `log_text`, `log_body`, and `log_raw_text`.

You can activate the SQL prompt by pressing the `;` key. At the prompt, you can start typing in the desired SQL statement and/or double-tap `TAB` to activate auto-completion. A help window will appear above the prompt to guide you in the usage of SQL keywords and functions.

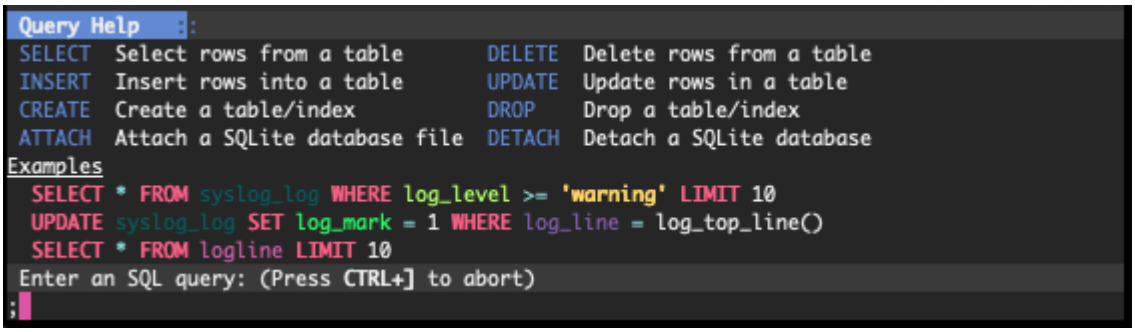

Fig. 1: Screenshot of the online help for the SQL prompt.

A simple query to perform on an Apache access log might be to get the average and maximum number of bytes returned by the server, grouped by IP address:

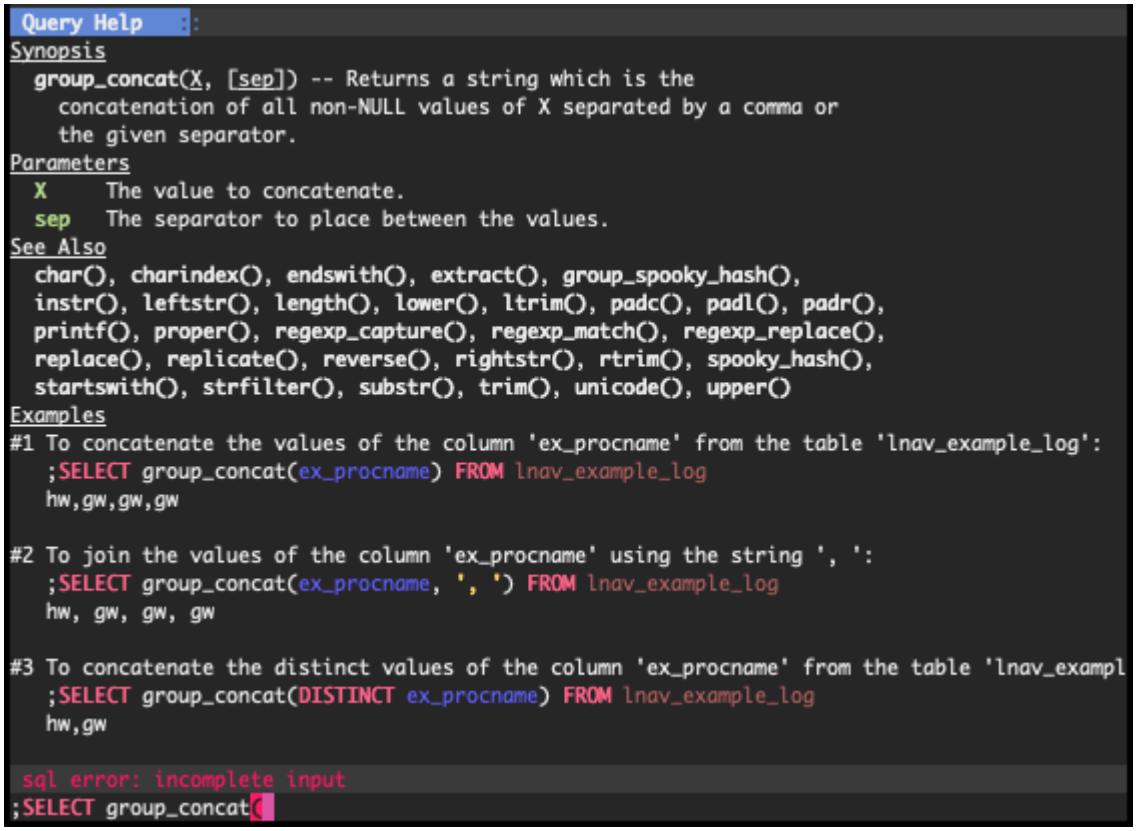

Fig. 2: Screenshot of the online help for the `group_concat()` function.

```
;SELECT c_ip, avg(sc_bytes), max(sc_bytes) FROM access_log GROUP BY c_ip
```

Note

For reference, the PRQL query would look like this:

```
from access_log | stats.by c_ip {average sc_bytes, max sc_bytes}
```

After pressing `Enter`, SQLite will execute the query using **lnav**'s virtual table implementation to extract the data directly from the log files. Once the query has finished, the main window will switch to the DB view to show the results. Press `q` to return to the log view and press `v` to return to the log view. If the SQL results contain a `log_line` column, you can press to `Shift` + `V` to switch between the DB view and the log

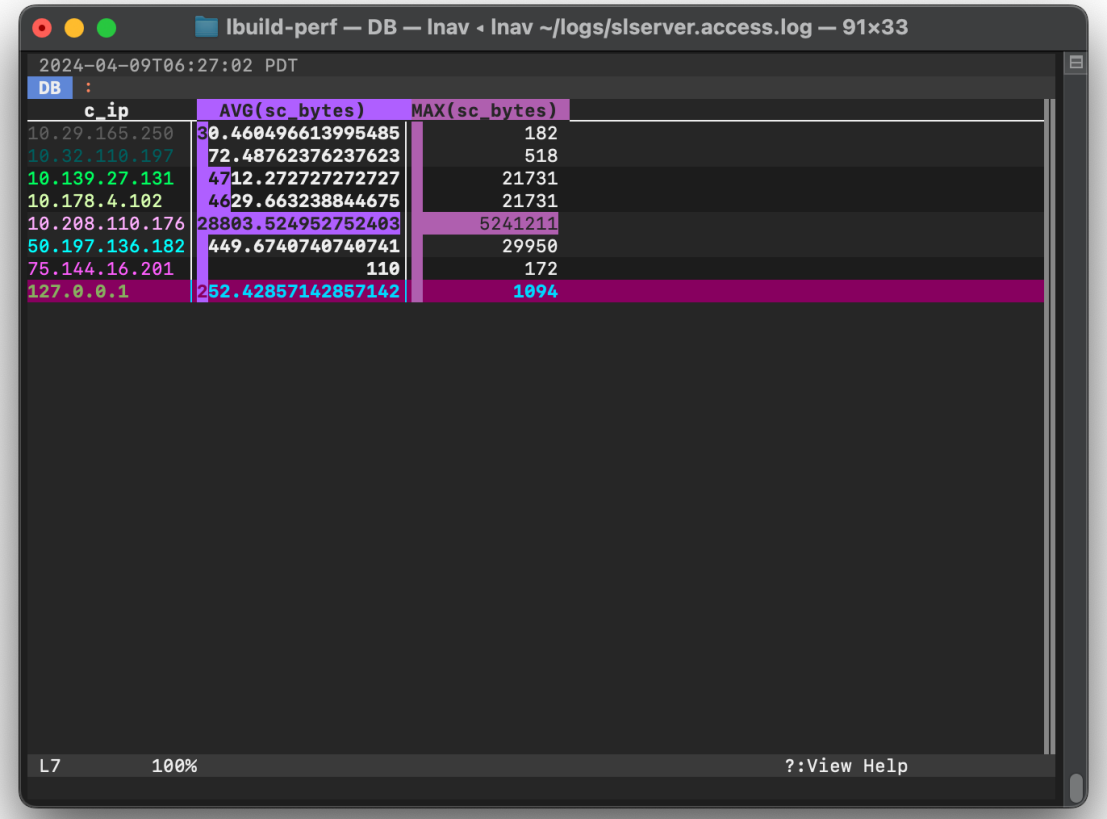

Fig. 3: Screenshot of the SQL results view.

12.1 DB View

The DB view has the following display features:

- Column headers stick to the top of the view when scrolling.
- Numeric columns contain a bar chart of the values.
- Pressing `p` opens an overlay with the columns and the values from the focused row in a vertical orientation for easier reading. Columns with JSON objects/arrays are pretty-printed with bar-charts for numeric values as well. The display will show the value and JSON-Pointer path that can be passed to the *jget* function.
- With the overlay open, pressing `CTRL-]` will focus into it. Then, you can select a column and copy its contents by pressing `c` or hide/show it by pressing the space bar. You can also hide/show a column by clicking on the diamond on the left side.
- Table cells can be styled by adding a `__lnav_style__` column to the query. This column must be a JSON object with the key *columns* that contains the the column names to be styled and the *style configuration*. For example, to apply semantic coloring to the `cs_uri_stem` column you would use the following JSON:

```
{
    "columns": {
        "cs_uri_stem": {
           "color": "semantic()"
        }
    }
}
```

12.1.1 style

type	*object*	
properties		
– text-align	*/text-align*	
	How to align text within a cell	
	type	*string*
	enum	start, center, end
– color	*/color*	
	The foreground color value for this style. The value can be the name of an xterm color, the h or a theme variable reference.	
	type	*string*
	examples	#fff
		Green
		$black
– background-color	*/background-color*	
	The background color value for this style. The value can be the name of an xterm color, the value, or a theme variable reference.	
	type	*string*
	examples	#2d2a2e
		Green
– underline	*/underline*	
	Indicates that the text should be underlined.	
	type	*boolean*
– bold	*/bold*	
	Indicates that the text should be bolded.	

continu

Table 1 – continued from previous page

	type	*boolean*
– italic	*/italic*	
	Indicates that the text should be italicized.	
	type	*boolean*
– strike	*/strike*	
	Indicates that the text should be struck.	
	type	*boolean*
– nestable	*/nestable*	
	This highlight can be nested in another highlight.	
	type	*boolean*
additionalProperties	False	

12.2 PRQL Support (v0.12.1+)

PRQL is an alternative database query language that compiles to SQLite. You can enter PRQL in the database query prompt and lnav will switch accordingly. A major advantage of using PRQL is that lnav can show previews of the results of the pipeline stages and provide better tab completion options.

A PRQL query starts with the `from` keyword that specifies the table to use as a data source. The next stage of a pipeline is started by entering a pipe symbol (`|`) followed by a PRQL transform. As you build the query in the prompt, lnav will display any relevant help and preview for the current and previous stages of the pipeline.

The following is a screenshot of lnav viewing a web access log with a query in progress:

The top half of the window is the usual log message view. Below that is the online help panel showing the documentation for the `stats.count_by` PRQL function. lnav will show the help for what is currently under the cursor. The next panel shows the preview data for the pipeline stage that precedes the stage where the cursor is. In this case, the results of `from access_log`, which is the contents of the access log table. The second preview window shows the result of the pipeline stage where the cursor is located.

12.3 Log Tables

Each log format has its own database table that can be used to access log messages that match that format. The table name is the same as the format name, for example, the `syslog_log` format will have a table that is also named `syslog_log`. There is also an `all_logs` table that provides access to all messages from all formats.

Note

Only the displayed log messages are reflected in the SQLite interface. Any log messages that have been filtered out are not accessible.

The columns in the log tables are made up of several builtins along with the values captured by the log format specification. Use the `.schema` command in the SQL prompt to examine a dump of the current database schema.

Note

Unless explicitly stated, builtin columns are read-only and cannot be changed by an `UPDATE`.

The following columns are builtin and included in a `SELECT *`:

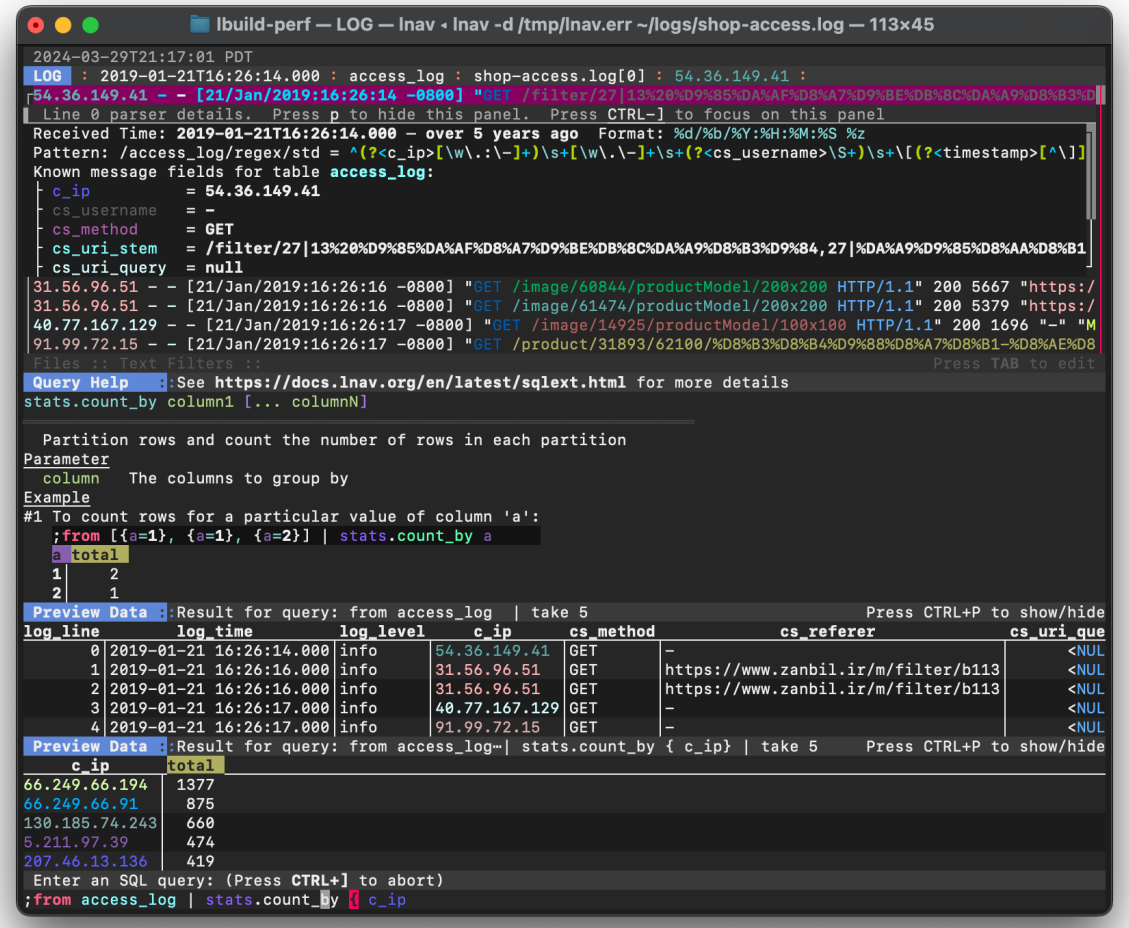

Fig. 4: Screenshot of a PRQL query in progress

log_line
The line number for the message in the log view.

log_time
The adjusted timestamp for the log message. This time can differ from the log message's time stamp if it arrived out-of-order and the log format expects log files to be time-ordered.

log_level
The log message level.

log_part
The partition the message is in. This column can be changed by an `UPDATE` or the *:parition-name* command.

log_actual_time
The log messages original timestamp in the file.

log_idle_msecs
The difference in time between this messages and the previous. The unit of time is milliseconds.

log_mark
True if the log message was marked by the user. This column can be changed by an `UPDATE`.

log_comment
The comment for the message. This column can be changed by an `UPDATE` or the *:comment* command.

log_tags
A JSON list of tags for the message. This column can be changed by an `UPDATE` or the *:tag* command.

log_annotations
A JSON object of annotations for this message. This column is populated by the *:annotate* command.

log_filters
A JSON list of filter IDs that matched this message

The following columns are builtin and are hidden, so they will *not* be included in a `SELECT *`:

log_opid
The OP ID as captured from the log message or as set by an `UPDATE`. Setting the OP ID allows operations to be visualized in the *timeline* view.

log_user_opid
The OP ID as set by the user.

log_format
The name of the format that parsed this log message.

log_format_regex
The name of the format's regex that matched this message.

log_time_msecs
The adjusted timestamp for the log message as the number of milliseconds from the epoch. This column can be more efficient to use for time-related operations, like *timeslice()*.

log_path
The path to the log file this message is from.

log_text
The full text of the log message.

log_body
The body of the log message.

log_raw_text
The raw text of this message from the log file. In this case of JSON and CSV logs, this will be the exact line of JSON-Line and CSV text from the file.

log_line_hash
A hash of the first line of the log message.

log_line_link
The permalink for the log message.

12.4 Extensions

To make it easier to analyze log data from within **lnav**, there are several built-in extensions that provide extra functions and collators beyond those provided by SQLite. The majority of the functions are from the extensions-functions.c file available from the sqlite.org web site.

> **Tip**
>
> You can include a SQLite database file on the command-line and use **lnav**'s interface to perform queries. The database will be attached with a name based on the database file name.

12.5 Commands

A SQL command is an internal macro implemented by lnav.

- `.schema` - Open the schema view. This view contains a dump of the schema for the internal tables and any tables in attached databases.
- `.read` - Execute the SQL statements in the given file.
- `.dump` - Write a file containing SQL statements that can be used to recreate a table.
- `.save` - Write the contents of the main DB to a SQLite database file. This command is useful if you have created some tables during your analysis and would like to preserve them. The command is implemented using the SQLite Backup API, so it can be run in the background.
- `.msgformats` - Executes a canned query that groups and counts log messages by the format of their message bodies. This command can be useful for quickly finding out the types of messages that are most common in a log file.

12.6 Variables

The following variables are available in SQL statements:

- `$LINES` - The number of lines in the terminal window.
- `$COLS` - The number of columns in the terminal window.

12.7 Environment

Environment variables can be accessed in queries using the usual syntax of `$VAR_NAME`. For example, to read the value of the "USER" variable, you can write:

```
;SELECT $USER
```

12.8 Collators

- **naturalcase** - Compare strings "naturally" so that number values in the string are compared based on their numeric value and not their character values. For example, "foo10" would be considered greater than "foo2".
- **naturalnocase** - The same as naturalcase, but case-insensitive.
- **ipaddress** - Compare IPv4/IPv6 addresses.
- **loglevel** - Compare log levels.
- **measure_with_units** - Compare numbers with unit suffixes. The currently supported suffixes are:
  - Sizes with an E/P/T/G/M/K prefix.
  - Seconds with an f/p/n/u/m prefix.
  - Durations of the form `HH:MM:SS` or `HH:MM:SS`

12.9 Reference

The following is a reference of the SQL syntax and functions that are available:

12.9.1 SELECT *[filter]* *result-column* FROM *table* WHERE *[cond]* GROUP BY *grouping-expr* ORDER BY *ordering-term* LIMIT *limit-expr*

Query the database and return zero or more rows of data.

Parameters

- **filter** — Additional processing of rows
- **result-column** — The expression used to generate a result for this column.
- **table** — The table(s) to query for data
- **cond** — The conditions used to select the rows to return.
- **grouping-expr** — The expression to use when grouping rows.
- **ordering-term** — The values to use when ordering the result set.
- **limit-expr** — The maximum number of rows to return.

Examples

To select all of the columns from the table 'lnav_example_log':

```
;SELECT * FROM lnav_example_log
 log_line  log_part    log_time     log_actual_time log_idle_msecs log_
↪level  log_mark  log_comment log_tags log_filters ex_procname ex_
↪duration log_time_msecs log_path    log_text        log_body
         0 <NULL>   2017-02:06.100 2017-02:06.100              0 info    ␣
↪          0 <NULL>      <NULL>   <NULL>      hw                     2 ␣
```

(continues on next page)

(continued from previous page)

```
↪1486094706000 /tmp/log 2017-02| World! Hello, World!
         1 <NULL>   2017-02|:06.200 2017-02|:06.200            100 error   ␣
↪           0 <NULL>       <NULL>   <NULL>      gw                   4 ␣
↪1486094706000 /tmp/log 2017-02| World! Goodbye, World!
         2 new      2017-02|:06.200 2017-02|:06.200        1200000 warn    ␣
↪           0 <NULL>       <NULL>   <NULL>      gw                   1 ␣
↪1486095906000 /tmp/log 2017-02| World! Goodbye, World!
         3 new      2017-02|:06.200 2017-02|:06.200        1800000 debug   ␣
↪           0 <NULL>       <NULL>   <NULL>      gw                  10 ␣
↪1486097706000 /tmp/log 2017-02| World! Goodbye, World!
```

12.9.2 expr *[NOT]* BETWEEN *low* AND *hi*

Test if an expression is between two values.

Parameters

- **low*** — The low point
- **hi*** — The high point

Examples

To check if 3 is between 5 and 10:

```
;SELECT 3 BETWEEN 5 AND 10
0
```

To check if 10 is between 5 and 10:

```
;SELECT 10 BETWEEN 5 AND 10
1
```

12.9.3 ATTACH DATABASE *filename* AS *schema-name*

Attach a database file to the current connection.

Parameters

- **filename*** — The path to the database file.
- **schema-name*** — The prefix for tables in this database.

Examples

To attach the database file '/tmp/customers.db' with the name customers:

```
;ATTACH DATABASE '/tmp/customers.db' AS customers
```

12.9.4 CREATE *[TEMP]* VIEW *[IF NOT EXISTS] [schema-name.] view-name* AS *select-stmt*

Assign a name to a SELECT statement

Parameters

- **IF NOT EXISTS** — Do not create the view if it already exists
- **schema-name.** — The database to create the view in
- **view-name*** — The name of the view
- **select-stmt*** — The SELECT statement the view represents

---

12.9.5 CREATE *[TEMP]* TABLE *[IF NOT EXISTS] [schema-name.] table-name* AS *select-stmt*

Create a table

12.9.6 WITH RECURSIVE *cte-table-name* AS *select-stmt*

Create a temporary view that exists only for the duration of a SQL statement.

Parameters

- **cte-table-name*** — The name for the temporary table.
- **select-stmt*** — The SELECT statement used to populate the temporary table.

12.9.7 CAST(*expr* AS *type-name*)

Convert the value of the given expression to a different storage class specified by type-name.

Parameters

- **expr*** — The value to convert.
- **type-name*** — The name of the type to convert to.

Examples

To cast the value 1.23 as an integer:

```
;SELECT CAST(1.23 AS INTEGER)
1
```

12.9.8 ordering-term COLLATE *[collation-name] [direction] [null-handling]*

The values to use in ordering result rows

Parameters

- **direction** — The direction, ASCending or DESCending

12.9.9 CASE *[base-expr]* WHEN *cmp-expr* ELSE *[else-expr]* END

Evaluate a series of expressions in order until one evaluates to true and then return it's result. Similar to an IF-THEN-ELSE construct in other languages.

Parameters

- **base-expr** — The base expression that is used for comparison in the branches
- **cmp-expr** — The expression to test if this branch should be taken
  - **then-expr*** — The result for this branch.
- **else-expr** — The result of this CASE if no branches matched.

Examples

To evaluate the number one and return the string 'one':

```
;SELECT CASE 1 WHEN 0 THEN 'zero' WHEN 1 THEN 'one' END
one
```

12.9.10 expr COLLATE *collation-name*

Assign a collating sequence to the expression.

Parameters

- **collation-name*** — The name of the collator.

Examples

To change the collation method for string comparisons:

```
;SELECT ('a2' < 'a10'), ('a2' < 'a10' COLLATE naturalnocase)
('a2' < 'a10') ('a2' <nocase)
             0               1
```

12.9.11 DETACH DATABASE *schema-name*

Detach a database from the current connection.

Parameters

- **schema-name*** — The prefix for tables in this database.

Examples

To detach the database named 'customers':

```
;DETACH DATABASE customers
```

12.9.12 DELETE FROM *table-name* WHERE *[cond]*

Delete rows from a table

Parameters

- **table-name*** — The name of the table
- **cond** — The conditions used to delete the rows.

12.9.13 expr IS *expr*

Test the distinctness of an expression

Examples

To check if 10 is between 5 and 10:

```
;SELECT 10 BETWEEN 5 AND 10
1
```

12.9.14 DROP INDEX *[IF EXISTS] [schema-name.] index-name*

Drop an index

12.9.15 DROP TABLE *[IF EXISTS] [schema-name.] table-name*

Drop a table

12.9.16 DROP VIEW *[IF EXISTS] [schema-name.] view-name*

Drop a view

12.9.17 DROP TRIGGER *[IF EXISTS] [schema-name.] trigger-name*

Drop a trigger

12.9.18 expr *[NOT]* LIKE *pattern* ESCAPE *[escape]*

Match an expression against a text pattern.

Parameters

- **pattern*** — The pattern to match against.
- **escape** — Character used to escape a % or _ in the pattern

Examples

To check if a value matches the pattern 'Hello, %!':

```
;SELECT 'Hello, World!' LIKE 'Hello, %!'
1
```

12.9.19 select-stmt

Execute a query and return 0 if no rows match or 1 otherwise

12.9.20 FILTER WHERE *expr*

Condition for rows to include in the aggregate

12.9.21 expr *[NOT]* GLOB *pattern*

Match an expression against a glob pattern.

Parameters

- **pattern*** — The glob pattern to match against.

Examples

To check if a value matches the pattern '*.log':

```
;SELECT 'foobar.log' GLOB '*.log'
1
```

12.9.22 INSERT INTO *[schema-name.] table-name column-name* VALUES *expr*

Insert rows into a table

Examples

To insert the pair containing 'MSG' and 'HELLO, WORLD!' into the 'environ' table:

```
;INSERT INTO environ VALUES ('MSG', 'HELLO, WORLD!')
```

12.9.23 expr *[nullness]*

Check an expression against NULL

Examples

To check if a value is not NULL:

```
;SELECT 'abc' NOT NULL
1
```

12.9.24 expr *[NOT]* REGEXP *pattern*

Match an expression against a regular expression.

Parameters

- **pattern*** — The regular expression to match against.

Examples

To check if a value matches the pattern 'file-d+':

```
;SELECT 'file-23' REGEXP 'file-\d+'
1
```

12.9.25 select-stmt

Execute a query and return 1 if no rows match or 0 otherwise

12.9.26 OVER(*[base-window-name]* PARTITION BY *expr* ORDER BY *expr*, *[frame-spec]*)

Executes the preceding function over a window

Parameters

- **base-window-name** — The name of the window definition
- **expr** — The values to use for partitioning
- **expr** — The values used to order the rows in the window
- **frame-spec** — Determines which output rows are read by an aggregate window function

---

12.9.27 OVER *window-name*

Executes the preceding function over a window

Parameters

- **window-name*** — The name of the window definition

12.9.28 UPDATE *table* SET *column-name* WHERE *[cond]*

Modify a subset of values in zero or more rows of the given table

Parameters

- **table*** — The table to update
- **column-name** — The columns in the table to update.
  - **expr*** — The values to place into the column.
- **cond** — The condition used to determine whether a row should be updated.

Examples

To mark the syslog message at line 40:

```
;UPDATE syslog_log SET log_mark = 1 WHERE log_line = 40
```

12.9.29 abs(*x*)

Return the absolute value of the argument

Parameters

- **x*** — The number to convert

Examples

To get the absolute value of -1:

```
;SELECT abs(-1)
1
```

See Also

acos(num), *acosh(num)*, *asin(num)*, *asinh(num)*, *atan2(y, x)*, *atan(num)*, *atanh(num)*, *atn2(y, x)*, *avg(X)*, *ceil(num)*, *degrees(radians)*, *exp(x)*, *floor(num)*, *log10(x)*, *log(x)*, *max(X)*, *min(X)*, *pi()*, *power(base, exp)*, *radians(degrees)*, *round(num, [digits])*, *sign(num)*, *square(num)*, *sum(X)*, *total(X)*

12.9.30 acos(*num*)

Returns the arccosine of a number, in radians

Parameters

- **num*** — A cosine value that is between -1 and 1

Examples

To get the arccosine of 0.2:

```
;SELECT printf('%.3f', acos(0.2))
1.369
```

See Also

abs(x), *acosh(num)*, *asin(num)*, *asinh(num)*, *atan2(y, x)*, *atan(num)*, *atanh(num)*, *atn2(y, x)*, *avg(X)*, *ceil(num)*, *degrees(radians)*, *exp(x)*, *floor(num)*, *log10(x)*, *log(x)*, *max(X)*, *min(X)*, *pi()*, *power(base, exp)*, *radians(degrees)*, *round(num, [digits])*, *sign(num)*, *square(num)*, *sum(X)*, *total(X)*

12.9.31 acosh(*num*)

Returns the hyperbolic arccosine of a number

Parameters

- **num*** — A number that is one or more

Examples

To get the hyperbolic arccosine of 1.2:

```
;SELECT acosh(1.2)
0.6223625037147786
```

See Also

abs(x), *acos(num)*, *asin(num)*, *asinh(num)*, *atan2(y, x)*, *atan(num)*, *atanh(num)*, *atn2(y, x)*, *avg(X)*, *ceil(num)*, *degrees(radians)*, *exp(x)*, *floor(num)*, *log10(x)*, *log(x)*, *max(X)*, *min(X)*, *pi()*, *power(base, exp)*, *radians(degrees)*, *round(num, [digits])*, *sign(num)*, *square(num)*, *sum(X)*, *total(X)*

12.9.32 anonymize(*value*)

Replace identifying information with random values.

PRQL Name: text.anonymize

Parameters

- **value*** — The text to anonymize

Examples

To anonymize an IP address:

```
;SELECT anonymize('Hello, 192.168.1.2')
Aback, 10.0.0.1
```

See Also

char(X), *charindex(needle, haystack, [start])*, *concat_ws(sep, X)*, *concat(X)*, *decode(value, algorithm)*, *encode(value, algorithm)*, *endswith(str, suffix)*, *extract(str)*, *fuzzy_match(pattern, str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *gunzip(b)*, *gzip(value)*, *humanize_duration(secs)*, *humanize_file_size(value)*, *humanize_id(id)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *logfmt2json(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *parse_url(url)*, *pretty_print(str)*, *printf(format, X)*, *proper(str)*, *regexp_capture_into_json(string, pattern, [options])*, *regexp_capture(string, pattern)*, *regexp_match(re, str)*, *regexp_replace(str, re, repl)*, *replace(str, old, replacement)*, *replicate(str, N)*, *reverse(str)*, *rightstr(str, N)*, *rtrim(str, [chars])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *timezone(tz, ts)*, *trim(str, [chars])*, *unhex(X)*, *unicode(X)*, *unparse_url(obj)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.9.33 asin(*num*)

Returns the arcsine of a number, in radians

Parameters

- **num*** — A sine value that is between -1 and 1

Examples

To get the arcsine of 0.2:

```
;SELECT asin(0.2)
0.2013579207903308
```

See Also

abs(x), *acos(num)*, *acosh(num)*, *asinh(num)*, *atan2(y, x)*, *atan(num)*, *atanh(num)*, *atn2(y, x)*, *avg(X)*, *ceil(num)*, *degrees(radians)*, *exp(x)*, *floor(num)*, *log10(x)*, *log(x)*, *max(X)*, *min(X)*, *pi()*, *power(base, exp)*, *radians(degrees)*, *round(num, [digits])*, *sign(num)*, *square(num)*, *sum(X)*, *total(X)*

12.9.34 asinh(*num*)

Returns the hyperbolic arcsine of a number

Parameters

- **num*** — The number

Examples

To get the hyperbolic arcsine of 0.2:

```
;SELECT asinh(0.2)
0.19869011034924142
```

See Also

abs(x), *acos(num)*, *acosh(num)*, *asin(num)*, *atan2(y, x)*, *atan(num)*, *atanh(num)*, *atn2(y, x)*, *avg(X)*, *ceil(num)*, *degrees(radians)*, *exp(x)*, *floor(num)*, *log10(x)*, *log(x)*, *max(X)*, *min(X)*, *pi()*, *power(base, exp)*, *radians(degrees)*, *round(num, [digits])*, *sign(num)*, *square(num)*, *sum(X)*, *total(X)*

12.9.35 atan(*num*)

Returns the arctangent of a number, in radians

Parameters

- **num*** — The number

Examples

To get the arctangent of 0.2:

```
;SELECT atan(0.2)
0.19739555984988078
```

See Also

abs(x), *acos(num)*, *acosh(num)*, *asin(num)*, *asinh(num)*, *atan2(y, x)*, *atanh(num)*, *atn2(y, x)*, *avg(X)*, *ceil(num)*, *degrees(radians)*, *exp(x)*, *floor(num)*, *log10(x)*, *log(x)*, *max(X)*, *min(X)*, *pi()*, *power(base, exp)*, *radians(degrees)*, *round(num, [digits])*, *sign(num)*, *square(num)*, *sum(X)*, *total(X)*

12.9.36 atan2(*y*, *x*)

Returns the angle in the plane between the positive X axis and the ray from (0, 0) to the point (x, y)

Parameters

- **y*** — The y coordinate of the point
- **x*** — The x coordinate of the point

Examples

To get the angle, in degrees, for the point at (5, 5):

```
;SELECT degrees(atan2(5, 5))
45
```

See Also

abs(x), *acos(num)*, *acosh(num)*, *asin(num)*, *asinh(num)*, *atan(num)*, *atanh(num)*, *atn2(y, x)*, *avg(X)*, *ceil(num)*, *degrees(radians)*, *exp(x)*, *floor(num)*, *log10(x)*, *log(x)*, *max(X)*, *min(X)*, *pi()*, *power(base, exp)*, *radians(degrees)*, *round(num, [digits])*, *sign(num)*, *square(num)*, *sum(X)*, *total(X)*

12.9.37 atanh(*num*)

Returns the hyperbolic arctangent of a number

Parameters

- **num*** — The number

Examples

To get the hyperbolic arctangent of 0.2:

```
;SELECT atanh(0.2)
0.2027325540540822
```

See Also

abs(x), *acos(num)*, *acosh(num)*, *asin(num)*, *asinh(num)*, *atan2(y, x)*, *atan(num)*, *atn2(y, x)*, *avg(X)*, *ceil(num)*, *degrees(radians)*, *exp(x)*, *floor(num)*, *log10(x)*, *log(x)*, *max(X)*, *min(X)*, *pi()*, *power(base, exp)*, *radians(degrees)*, *round(num, [digits])*, *sign(num)*, *square(num)*, *sum(X)*, *total(X)*

12.9.38 atn2(*y*, *x*)

Returns the angle in the plane between the positive X axis and the ray from (0, 0) to the point (x, y)

Parameters

- **y*** — The y coordinate of the point
- **x*** — The x coordinate of the point

Examples

To get the angle, in degrees, for the point at (5, 5):

```
;SELECT degrees(atn2(5, 5))
45
```

See Also

abs(x), *acos(num)*, *acosh(num)*, *asin(num)*, *asinh(num)*, *atan2(y, x)*, *atan(num)*, *atanh(num)*, *avg(X)*, *ceil(num)*, *degrees(radians)*, *exp(x)*, *floor(num)*, *log10(x)*, *log(x)*, *max(X)*, *min(X)*, *pi()*, *power(base, exp)*, *radians(degrees)*, *round(num, [digits])*, *sign(num)*, *square(num)*, *sum(X)*, *total(X)*

12.9.39 avg(*X*)

Returns the average value of all non-NULL numbers within a group.

Parameters

- **X*** — The value to compute the average of.

Examples

To get the average of the column 'ex_duration' from the table 'lnav_example_log':

```
;SELECT avg(ex_duration) FROM lnav_example_log
4.25
```

To get the average of the column 'ex_duration' from the table 'lnav_example_log' when grouped by 'ex_procname':

```
;SELECT ex_procname, avg(ex_duration) FROM lnav_example_log GROUP BY ex_
↪procname
ex_procname avg(ex_ration)
gw                        5
hw                        2
```

See Also

abs(x), *acos(num)*, *acosh(num)*, *asin(num)*, *asinh(num)*, *atan2(y, x)*, *atan(num)*, *atanh(num)*, *atn2(y, x)*, *ceil(num)*, *degrees(radians)*, *exp(x)*, *floor(num)*, *log10(x)*, *log(x)*, *max(X)*, *min(X)*, *pi()*, *power(base, exp)*, *radians(degrees)*, *round(num, [digits])*, *sign(num)*, *square(num)*, *sum(X)*, *total(X)*

12.9.40 basename(*path*)

Extract the base portion of a pathname.

PRQL Name: fs.basename

Parameters

- **path*** — The path

Examples

To get the base of a plain file name:

```
;SELECT basename('foobar')
foobar
```

To get the base of a path:

```
;SELECT basename('foo/bar')
bar
```

To get the base of a directory:

```
;SELECT basename('foo/bar/')
bar
```

To get the base of an empty string:

```
;SELECT basename('')
.
```

To get the base of a Windows path:

```
;SELECT basename('foo\bar')
bar
```

To get the base of the root directory:

```
;SELECT basename('/')
/
```

To get the base of a path:

```
;from [{p='foo/bar'}] | select { fs.basename p }
bar
```

See Also

dirname(path), *joinpath(path)*, *readlink(path)*, *realpath(path)*

12.9.41 ceil(*num*)

Returns the smallest integer that is not less than the argument

Parameters

- **num*** — The number to raise to the ceiling

Examples

To get the ceiling of 1.23:

```
;SELECT ceil(1.23)
2
```

See Also

abs(x), *acos(num)*, *acosh(num)*, *asin(num)*, *asinh(num)*, *atan2(y, x)*, *atan(num)*, *atanh(num)*, *atn2(y, x)*, *avg(X)*, *degrees(radians)*, *exp(x)*, *floor(num)*, *log10(x)*, *log(x)*, *max(X)*, *min(X)*, *pi()*, *power(base, exp)*, *radians(degrees)*, *round(num, [digits])*, *sign(num)*, *square(num)*, *sum(X)*, *total(X)*

12.9.42 changes()

The number of database rows that were changed, inserted, or deleted by the most recent statement.

12.9.43 char(*X*)

Returns a string composed of characters having the given unicode code point values

Parameters

- **X** — The unicode code point values

Examples

To get a string with the code points 0x48 and 0x49:

```
;SELECT char(0x48, 0x49)
HI
```

See Also

anonymize(value), *charindex(needle, haystack, [start])*, *concat_ws(sep, X)*, *concat(X)*, *decode(value, algorithm)*, *encode(value, algorithm)*, *endswith(str, suffix)*, *extract(str)*, *fuzzy_match(pattern, str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *gunzip(b)*, *gzip(value)*,

humanize_duration(secs), *humanize_file_size(value)*, *humanize_id(id)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *logfmt2json(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *parse_url(url)*, *pretty_print(str)*, *printf(format, X)*, *proper(str)*, *regexp_capture_into_json(string, pattern, [options])*, *regexp_capture(string, pattern)*, *regexp_match(re, str)*, *regexp_replace(str, re, repl)*, *replace(str, old, replacement)*, *replicate(str, N)*, *reverse(str)*, *rightstr(str, N)*, *rtrim(str, [chars])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *timezone(tz, ts)*, *trim(str, [chars])*, *unhex(X)*, *unicode(X)*, *unparse_url(obj)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.9.44 charindex(*needle*, *haystack*, *[start]*)

Finds the first occurrence of the needle within the haystack and returns the number of prior characters plus 1, or 0 if Y is nowhere found within X

Parameters

- **needle*** — The string to look for in the haystack
- **haystack*** — The string to search within
- **start** — The one-based index within the haystack to start the search

Examples

To search for the string 'abc' within 'abcabc' and starting at position 2:

```
;SELECT charindex('abc', 'abcabc', 2)
4
```

To search for the string 'abc' within 'abcdef' and starting at position 2:

```
;SELECT charindex('abc', 'abcdef', 2)
0
```

See Also

anonymize(value), *char(X)*, *concat_ws(sep, X)*, *concat(X)*, *decode(value, algorithm)*, *encode(value, algorithm)*, *endswith(str, suffix)*, *extract(str)*, *fuzzy_match(pattern, str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *gunzip(b)*, *gzip(value)*, *humanize_duration(secs)*, *humanize_file_size(value)*, *humanize_id(id)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *logfmt2json(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *parse_url(url)*, *pretty_print(str)*, *printf(format, X)*, *proper(str)*, *regexp_capture_into_json(string, pattern, [options])*, *regexp_capture(string, pattern)*, *regexp_match(re, str)*, *regexp_replace(str, re, repl)*, *replace(str, old, replacement)*, *replicate(str, N)*, *reverse(str)*, *rightstr(str, N)*, *rtrim(str, [chars])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *timezone(tz, ts)*, *trim(str, [chars])*, *unhex(X)*, *unicode(X)*, *unparse_url(obj)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.9.45 coalesce(*X*, *Y*)

Returns a copy of its first non-NULL argument, or NULL if all arguments are NULL

Parameters

- **X*** — A value to check for NULL-ness
- **Y** — A value to check for NULL-ness

Examples

To get the first non-null value from three parameters:

```
;SELECT coalesce(null, 0, null)
0
```

12.9.46 concat(*X*)

Returns a string that is the concatenation of all non-NULL arguments

Parameters

- **X** — The values to concatenate together

Examples

To concatenate a label and number:

```
;SELECT concat('Size: ', 1234)
Size: 1234
```

See Also

anonymize(value), *char(X)*, *charindex(needle, haystack, [start])*, *concat_ws(sep, X)*, *decode(value, algorithm)*, *encode(value, algorithm)*, *endswith(str, suffix)*, *extract(str)*, *fuzzy_match(pattern, str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *gunzip(b)*, *gzip(value)*, *humanize_duration(secs)*, *humanize_file_size(value)*, *humanize_id(id)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *logfmt2json(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *parse_url(url)*, *pretty_print(str)*, *printf(format, X)*, *proper(str)*, *regexp_capture_into_json(string, pattern, [options])*, *regexp_capture(string, pattern)*, *regexp_match(re, str)*, *regexp_replace(str, re, repl)*, *replace(str, old, replacement)*, *replicate(str, N)*, *reverse(str)*, *rightstr(str, N)*, *rtrim(str, [chars])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *timezone(tz, ts)*, *trim(str, [chars])*, *unhex(X)*, *unicode(X)*, *unparse_url(obj)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.9.47 concat_ws(*sep*, *X*)

Returns a string that is the concatenation of all non-NULL arguments separated by the first argument

Parameters

- **sep*** — The separator
- **X** — The values to concatenate together

Examples

To separate numbers with a comma:

```
;SELECT concat_ws(',', 1, 2, 3, 4)
1,2,3,4
```

See Also

anonymize(value), *char(X)*, *charindex(needle, haystack, [start])*, *concat(X)*, *decode(value, algorithm)*, *encode(value, algorithm)*, *endswith(str, suffix)*, *extract(str)*, *fuzzy_match(pattern, str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *gunzip(b)*, *gzip(value)*, *humanize_duration(secs)*, *humanize_file_size(value)*, *humanize_id(id)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*,

logfmt2json(str), *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *parse_url(url)*, *pretty_print(str)*, *printf(format, X)*, *proper(str)*, *regexp_capture_into_json(string, pattern, [options])*, *regexp_capture(string, pattern)*, *regexp_match(re, str)*, *regexp_replace(str, re, repl)*, *replace(str, old, replacement)*, *replicate(str, N)*, *reverse(str)*, *rightstr(str, N)*, *rtrim(str, [chars])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *timezone(tz, ts)*, *trim(str, [chars])*, *unhex(X)*, *unicode(X)*, *unparse_url(obj)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.9.48 count(*X*)

If the argument is '*', the total number of rows in the group is returned. Otherwise, the number of times the argument is non-NULL.

Parameters

- **X*** — The value to count.

Examples

To get the count of the non-NULL rows of 'lnav_example_log':

```
;SELECT count(*) FROM lnav_example_log
4
```

To get the count of the non-NULL values of 'log_part' from 'lnav_example_log':

```
;SELECT count(log_part) FROM lnav_example_log
2
```

12.9.49 cume_dist()

Returns the cumulative distribution

See Also

dense_rank(), *first_value(expr)*, *lag(expr, [offset], [default])*, *last_value(expr)*, *lead(expr, [offset], [default])*, *nth_value(expr, N)*, *ntile(groups)*, *percent_rank()*, *rank()*, *row_number()*

12.9.50 date(*timestring*, *modifier*)

Returns the date in this format: YYYY-MM-DD.

Parameters

- **timestring*** — The string to convert to a date.
- **modifier** — A transformation that is applied to the value to the left.

Examples

To get the date portion of the timestamp '2017-01-02T03:04:05':

```
;SELECT date('2017-01-02T03:04:05')
2017-01-02
```

To get the date portion of the timestamp '2017-01-02T03:04:05' plus one day:

```
;SELECT date('2017-01-02T03:04:05', '+1 day')
2017-01-03
```

To get the date portion of the epoch timestamp 1491341842:

```
;SELECT date(1491341842, 'unixepoch')
2017-04-04
```

See Also

datetime(timestring, modifier), *humanize_duration(secs)*, *julianday(timestring, modifier)*, *strftime(format, timestring, modifier)*, *time(timestring, modifier)*, *timediff(time1, time2)*, *timeslice(time, slice)*, *timezone(tz, ts)*

12.9.51 datetime(*timestring*, *modifier*)

Returns the date and time in this format: YYYY-MM-DD HH:MM:SS.

Parameters

- **timestring*** — The string to convert to a date with time.
- **modifier** — A transformation that is applied to the value to the left.

Examples

To get the date and time portion of the timestamp '2017-01-02T03:04:05':

```
;SELECT datetime('2017-01-02T03:04:05')
2017-01-02 03:04:05
```

To get the date and time portion of the timestamp '2017-01-02T03:04:05' plus one minute:

```
;SELECT datetime('2017-01-02T03:04:05', '+1 minute')
2017-01-02 03:05:05
```

To get the date and time portion of the epoch timestamp 1491341842:

```
;SELECT datetime(1491341842, 'unixepoch')
2017-04-04 21:37:22
```

See Also

date(timestring, modifier), *humanize_duration(secs)*, *julianday(timestring, modifier)*, *strftime(format, timestring, modifier)*, *time(timestring, modifier)*, *timediff(time1, time2)*, *timeslice(time, slice)*, *timezone(tz, ts)*

12.9.52 decode(*value*, *algorithm*)

Decode the value using the given algorithm

Parameters

- **value*** — The value to decode
- **algorithm*** — One of the following encoding algorithms: base64, hex, uri

Examples

To decode the URI-encoded string '%63%75%72%6c':

```
;SELECT decode('%63%75%72%6c', 'uri')
curl
```

See Also

anonymize(value), *char(X)*, *charindex(needle, haystack, [start])*, *concat_ws(sep, X)*, *concat(X)*, *encode(value, algorithm)*, *endswith(str, suffix)*, *extract(str)*, *fuzzy_match(pattern, str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *gunzip(b)*, *gzip(value)*, *humanize_duration(secs)*, *humanize_file_size(value)*, *humanize_id(id)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *logfmt2json(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *parse_url(url)*, *pretty_print(str)*, *printf(format, X)*, *proper(str)*, *regexp_capture_into_json(string, pattern, [options])*, *regexp_capture(string, pattern)*, *regexp_match(re, str)*, *regexp_replace(str, re, repl)*, *replace(str, old, replacement)*, *replicate(str, N)*, *reverse(str)*, *rightstr(str, N)*, *rtrim(str, [chars])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *timezone(tz, ts)*, *trim(str, [chars])*, *unhex(X)*, *unicode(X)*, *unparse_url(obj)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.9.53 degrees(*radians*)

Converts radians to degrees

Parameters

- **radians*** — The radians value to convert to degrees

Examples

To convert PI to degrees:

```
;SELECT degrees(pi())
180
```

See Also

abs(x), *acos(num)*, *acosh(num)*, *asin(num)*, *asinh(num)*, *atan2(y, x)*, *atan(num)*, *atanh(num)*, *atn2(y, x)*, *avg(X)*, *ceil(num)*, *exp(x)*, *floor(num)*, *log10(x)*, *log(x)*, *max(X)*, *min(X)*, *pi()*, *power(base, exp)*, *radians(degrees)*, *round(num, [digits])*, *sign(num)*, *square(num)*, *sum(X)*, *total(X)*

12.9.54 dense_rank()

Returns the row_number() of the first peer in each group without gaps

See Also

cume_dist(), *first_value(expr)*, *lag(expr, [offset], [default])*, *last_value(expr)*, *lead(expr, [offset], [default])*, *nth_value(expr, N)*, *ntile(groups)*, *percent_rank()*, *rank()*, *row_number()*

12.9.55 dirname(*path*)

Extract the directory portion of a pathname.

PRQL Name: fs.dirname

Parameters

- **path*** — The path

Examples

To get the directory of a relative file path:

```
;SELECT dirname('foo/bar')
foo
```

To get the directory of an absolute file path:

```
;SELECT dirname('/foo/bar')
/foo
```

To get the directory of a file in the root directory:

```
;SELECT dirname('/bar')
/
```

To get the directory of a Windows path:

```
;SELECT dirname('foo\bar')
foo
```

To get the directory of an empty path:

```
;SELECT dirname('')
.
```

See Also

basename(path), *joinpath(path)*, *readlink(path)*, *realpath(path)*

12.9.56 echoln(*value*)

Echo the argument to the current output file and return it

Parameters

- **value*** — The value to write to the current output file

See Also

:append-to path, *;.dump path table*, *;.read path*, *;.save path*, *:echo [-n] msg*, *:export-session-to path*, *:open [–since] [–until] path*, *:pipe-line-to shell-cmd*, *:pipe-to shell-cmd*, *:redirect-to [path]*, *:write-csv-to [–anonymize] path*, *:write-json-cols-to [–anonymize] path*, *:write-json-to [–anonymize] path*, *:write-jsonlines-to [–all] [–anonymize] path*, *:write-raw-to [–view] [–anonymize] path*, *:write-screen-to [–anonymize] path*, *:write-table-to [–anonymize] path*, *:write-to [–anonymize] path*, *:write-view-to [–anonymize] path*, *:xopen path*

12.9.57 encode(*value*, *algorithm*)

Encode the value using the given algorithm

Parameters

- **value*** — The value to encode

- **algorithm*** — One of the following encoding algorithms: base64, hex, uri, html

Examples

To base64-encode 'Hello, World!':

```
;SELECT encode('Hello, World!', 'base64')
SGVsbG8sIFdvcmxkIQ==
```

To hex-encode 'Hello, World!':

```
;SELECT encode('Hello, World!', 'hex')
48656c6c6f2c20576f726c6421
```

To URI-encode 'Hello, World!':

```
;SELECT encode('Hello, World!', 'uri')
Hello%2C%20World%21
```

See Also

anonymize(value), *char(X)*, *charindex(needle, haystack, [start])*, *concat_ws(sep, X)*, *concat(X)*, *decode(value, algorithm)*, *endswith(str, suffix)*, *extract(str)*, *fuzzy_match(pattern, str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *gunzip(b)*, *gzip(value)*, *humanize_duration(secs)*, *humanize_file_size(value)*, *humanize_id(id)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *logfmt2json(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *parse_url(url)*, *pretty_print(str)*, *printf(format, X)*, *proper(str)*, *regexp_capture_into_json(string, pattern, [options])*, *regexp_capture(string, pattern)*, *regexp_match(re, str)*, *regexp_replace(str, re, repl)*, *replace(str, old, replacement)*, *replicate(str, N)*, *reverse(str)*, *rightstr(str, N)*, *rtrim(str, [chars])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *timezone(tz, ts)*, *trim(str, [chars])*, *unhex(X)*, *unicode(X)*, *unparse_url(obj)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.9.58 endswith(*str*, *suffix*)

Test if a string ends with the given suffix

Parameters

- **str*** — The string to test
- **suffix*** — The suffix to check in the string

Examples

To test if the string 'notbad.jpg' ends with '.jpg':

```
;SELECT endswith('notbad.jpg', '.jpg')
1
```

To test if the string 'notbad.png' starts with '.jpg':

```
;SELECT endswith('notbad.png', '.jpg')
0
```

See Also

anonymize(value), *char(X)*, *charindex(needle, haystack, [start])*, *concat_ws(sep, X)*, *concat(X)*, *decode(value, algorithm)*, *encode(value, algorithm)*, *extract(str)*, *fuzzy_match(pattern, str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *gunzip(b)*, *gzip(value)*, *humanize_duration(secs)*,

humanize_file_size(value), *humanize_id(id)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *logfmt2json(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *parse_url(url)*, *pretty_print(str)*, *printf(format, X)*, *proper(str)*, *regexp_capture_into_json(string, pattern, [options])*, *regexp_capture(string, pattern)*, *regexp_match(re, str)*, *regexp_replace(str, re, repl)*, *replace(str, old, replacement)*, *replicate(str, N)*, *reverse(str)*, *rightstr(str, N)*, *rtrim(str, [chars])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *timezone(tz, ts)*, *trim(str, [chars])*, *unhex(X)*, *unicode(X)*, *unparse_url(obj)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.9.59 exp(*x*)

Returns the value of e raised to the power of x

Parameters

- **x*** — The exponent

Examples

To raise e to 2:

```
;SELECT exp(2)
7.38905609893065
```

See Also

abs(x), *acos(num)*, *acosh(num)*, *asin(num)*, *asinh(num)*, *atan2(y, x)*, *atan(num)*, *atanh(num)*, *atn2(y, x)*, *avg(X)*, *ceil(num)*, *degrees(radians)*, *floor(num)*, *log10(x)*, *log(x)*, *max(X)*, *min(X)*, *pi()*, *power(base, exp)*, *radians(degrees)*, *round(num, [digits])*, *sign(num)*, *square(num)*, *sum(X)*, *total(X)*

12.9.60 extract(*str*)

Automatically Parse and extract data from a string

PRQL Name: text.discover

Parameters

- **str*** — The string to parse

Examples

To extract key/value pairs from a string:

```
;SELECT extract('foo=1 bar=2 name="Rolo Tomassi"')
{"foo":1,"bar":2,"name":"Rolo Tomassi"}
```

To extract columnar data from a string:

```
;SELECT extract('1.0 abc 2.0')
{"col_0":1.0,"col_1":2.0}
```

See Also

anonymize(value), *char(X)*, *charindex(needle, haystack, [start])*, *concat_ws(sep, X)*, *concat(X)*, *decode(value, algorithm)*, *encode(value, algorithm)*, *endswith(str, suffix)*, *fuzzy_match(pattern, str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *gunzip(b)*, *gzip(value)*, *humanize_duration(secs)*, *humanize_file_size(value)*, *humanize_id(id)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *logfmt2json(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*,

parse_url(url), *pretty_print(str)*, *printf(format, X)*, *proper(str)*, *regexp_capture_into_json(string, pattern, [options])*, *regexp_capture(string, pattern)*, *regexp_match(re, str)*, *regexp_replace(str, re, repl)*, *replace(str, old, replacement)*, *replicate(str, N)*, *reverse(str)*, *rightstr(str, N)*, *rtrim(str, [chars])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *timezone(tz, ts)*, *trim(str, [chars])*, *unhex(X)*, *unicode(X)*, *unparse_url(obj)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.9.61 first_value(*expr*)

Returns the result of evaluating the expression against the first row in the window frame.

Parameters

- **expr*** — The expression to execute over the first row

See Also

cume_dist(), *dense_rank()*, *lag(expr, [offset], [default])*, *last_value(expr)*, *lead(expr, [offset], [default])*, *nth_value(expr, N)*, *ntile(groups)*, *percent_rank()*, *rank()*, *row_number()*

12.9.62 floor(*num*)

Returns the largest integer that is not greater than the argument

Parameters

- **num*** — The number to lower to the floor

Examples

To get the floor of 1.23:

```
;SELECT floor(1.23)
1
```

See Also

abs(x), *acos(num)*, *acosh(num)*, *asin(num)*, *asinh(num)*, *atan2(y, x)*, *atan(num)*, *atanh(num)*, *atn2(y, x)*, *avg(X)*, *ceil(num)*, *degrees(radians)*, *exp(x)*, *log10(x)*, *log(x)*, *max(X)*, *min(X)*, *pi()*, *power(base, exp)*, *radians(degrees)*, *round(num, [digits])*, *sign(num)*, *square(num)*, *sum(X)*, *total(X)*

12.9.63 fstat(*pattern*)

A table-valued function for getting information about file paths/globs

Parameters

- **pattern*** — The file path or glob pattern to query.

Examples

To read a file and raise an error if there is a problem:

```
;SELECT ifnull(data, raise_error('cannot read: ' || st_name, error)) FROM␣
↪fstat('/non-existent')
  error: cannot read: non-existent
 reason: No such file or directory
```

(continues on next page)

(continued from previous page)

```
--> fstat:1
| SELECT ifnull(data, raise_error('cannot read: ' || st_name, error))␣
↪FROM fstat('/non-existent')
```

12.9.64 fuzzy_match(*pattern*, *str*)

Perform a fuzzy match of a pattern against a string and return a score or NULL if the pattern was not matched

Parameters

- **pattern*** — The pattern to look for in the string
- **str*** — The string to match against

Examples

To match the pattern 'fo' against 'filter-out':

```
;SELECT fuzzy_match('fo', 'filter-out')
137
```

See Also

anonymize(value), *char(X)*, *charindex(needle, haystack, [start])*, *concat_ws(sep, X)*, *concat(X)*, *decode(value, algorithm)*, *encode(value, algorithm)*, *endswith(str, suffix)*, *extract(str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *gunzip(b)*, *gzip(value)*, *humanize_duration(secs)*, *humanize_file_size(value)*, *humanize_id(id)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *logfmt2json(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *parse_url(url)*, *pretty_print(str)*, *printf(format, X)*, *proper(str)*, *regexp_capture_into_json(string, pattern, [options])*, *regexp_capture(string, pattern)*, *regexp_match(re, str)*, *regexp_replace(str, re, repl)*, *replace(str, old, replacement)*, *replicate(str, N)*, *reverse(str)*, *rightstr(str, N)*, *rtrim(str, [chars])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *timezone(tz, ts)*, *trim(str, [chars])*, *unhex(X)*, *unicode(X)*, *unparse_url(obj)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.9.65 generate_series(*start*, *stop*, *[step]*)

A table-valued-function that returns the whole numbers between a lower and upper bound, inclusive

Parameters

- **start*** — The starting point of the series
- **stop*** — The stopping point of the series
- **step** — The increment between each value

Examples

To generate the numbers in the range [10, 14]:

```
;SELECT value FROM generate_series(10, 14)
  value
        10
        11
```

(continues on next page)

(continued from previous page)

```
        12
        13
        14
```

To generate every other number in the range [10, 14]:

```
;SELECT value FROM generate_series(10, 14, 2)
  value
        10
        12
        14
```

To count down from five to 1:

```
;SELECT value FROM generate_series(1, 5, -1)
  value
         5
         4
         3
         2
         1
```

12.9.66 gethostbyaddr(*hostname*)

Get the hostname for the given IP address

PRQL Name: net.gethostbyaddr

Parameters

- **hostname*** — The IP address to lookup.

Examples

To get the hostname for the IP '127.0.0.1':

```
;SELECT gethostbyaddr('127.0.0.1')
localhost
```

See Also

gethostbyname(hostname)

12.9.67 gethostbyname(*hostname*)

Get the IP address for the given hostname

PRQL Name: net.gethostbyname

Parameters

- **hostname*** — The DNS hostname to lookup.

Examples

To get the IP address for 'localhost':

```
;SELECT gethostbyname('localhost')
127.0.0.1
```

See Also

gethostbyaddr(hostname)

12.9.68 glob(*pattern*, *str*)

Match a string against Unix glob pattern

Parameters

- **pattern*** — The glob pattern
- **str*** — The string to match

Examples

To test if the string 'abc' matches the glob 'a*':

```
;SELECT glob('a*', 'abc')
1
```

12.9.69 group_concat(*X*, *[sep]*)

Returns a string which is the concatenation of all non-NULL values of X separated by a comma or the given separator.

Parameters

- **X*** — The value to concatenate.
- **sep** — The separator to place between the values.

Examples

To concatenate the values of the column 'ex_procname' from the table 'lnav_example_log':

```
;SELECT group_concat(ex_procname) FROM lnav_example_log
hw,gw,gw,gw
```

To join the values of the column 'ex_procname' using the string ', ':

```
;SELECT group_concat(ex_procname, ', ') FROM lnav_example_log
hw, gw, gw, gw
```

To concatenate the distinct values of the column 'ex_procname' from the table 'lnav_example_log':

```
;SELECT group_concat(DISTINCT ex_procname) FROM lnav_example_log
hw,gw
```

See Also

anonymize(value), *char(X)*, *charindex(needle, haystack, [start])*, *concat_ws(sep, X)*, *concat(X)*, *decode(value, algorithm)*, *encode(value, algorithm)*, *endswith(str, suffix)*, *extract(str)*, *fuzzy_match(pattern, str)*, *group_spooky_hash(str)*, *gunzip(b)*, *gzip(value)*, *humanize_duration(secs)*, *humanize_file_size(value)*, *humanize_id(id)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *logfmt2json(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*,

padl(str, len), *padr(str, len)*, *parse_url(url)*, *pretty_print(str)*, *printf(format, X)*, *proper(str)*, *regexp_capture_into_json(string, pattern, [options])*, *regexp_capture(string, pattern)*, *regexp_match(re, str)*, *regexp_replace(str, re, repl)*, *replace(str, old, replacement)*, *replicate(str, N)*, *reverse(str)*, *rightstr(str, N)*, *rtrim(str, [chars])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *timezone(tz, ts)*, *trim(str, [chars])*, *unhex(X)*, *unicode(X)*, *unparse_url(obj)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.9.70 group_spooky_hash(*str*)

Compute the hash value for the given arguments

Parameters

- **str** — The string to hash

Examples

To produce a hash of all of the values of 'column1':

```
;SELECT group_spooky_hash(column1) FROM (VALUES ('abc'), ('123'))
4e7a190aead058cb123c94290f29c34a
```

See Also

anonymize(value), *char(X)*, *charindex(needle, haystack, [start])*, *concat_ws(sep, X)*, *concat(X)*, *decode(value, algorithm)*, *encode(value, algorithm)*, *endswith(str, suffix)*, *extract(str)*, *fuzzy_match(pattern, str)*, *group_concat(X, [sep])*, *gunzip(b)*, *gzip(value)*, *humanize_duration(secs)*, *humanize_file_size(value)*, *humanize_id(id)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *logfmt2json(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *parse_url(url)*, *pretty_print(str)*, *printf(format, X)*, *proper(str)*, *regexp_capture_into_json(string, pattern, [options])*, *regexp_capture(string, pattern)*, *regexp_match(re, str)*, *regexp_replace(str, re, repl)*, *replace(str, old, replacement)*, *replicate(str, N)*, *reverse(str)*, *rightstr(str, N)*, *rtrim(str, [chars])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *timezone(tz, ts)*, *trim(str, [chars])*, *unhex(X)*, *unicode(X)*, *unparse_url(obj)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.9.71 gunzip(*b*)

Decompress a gzip file

Parameters

- **b** — The blob to decompress

See Also

anonymize(value), *char(X)*, *charindex(needle, haystack, [start])*, *concat_ws(sep, X)*, *concat(X)*, *decode(value, algorithm)*, *encode(value, algorithm)*, *endswith(str, suffix)*, *extract(str)*, *fuzzy_match(pattern, str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *gzip(value)*, *humanize_duration(secs)*, *humanize_file_size(value)*, *humanize_id(id)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *logfmt2json(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *parse_url(url)*, *pretty_print(str)*, *printf(format, X)*, *proper(str)*, *regexp_capture_into_json(string, pattern, [options])*, *regexp_capture(string, pattern)*, *regexp_match(re, str)*, *regexp_replace(str, re, repl)*, *replace(str, old, replacement)*, *replicate(str, N)*, *reverse(str)*, *rightstr(str, N)*, *rtrim(str, [chars])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *timezone(tz, ts)*, *trim(str, [chars])*, *unhex(X)*, *unicode(X)*, *unparse_url(obj)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.9.72 gzip(*value*)

Compress a string into a gzip file

Parameters

- **value** — The value to compress

See Also

anonymize(value), *char(X)*, *charindex(needle, haystack, [start])*, *concat_ws(sep, X)*, *concat(X)*, *decode(value, algorithm)*, *encode(value, algorithm)*, *endswith(str, suffix)*, *extract(str)*, *fuzzy_match(pattern, str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *gunzip(b)*, *humanize_duration(secs)*, *humanize_file_size(value)*, *humanize_id(id)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *logfmt2json(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *parse_url(url)*, *pretty_print(str)*, *printf(format, X)*, *proper(str)*, *regexp_capture_into_json(string, pattern, [options])*, *regexp_capture(string, pattern)*, *regexp_match(re, str)*, *regexp_replace(str, re, repl)*, *replace(str, old, replacement)*, *replicate(str, N)*, *reverse(str)*, *rightstr(str, N)*, *rtrim(str, [chars])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *timezone(tz, ts)*, *trim(str, [chars])*, *unhex(X)*, *unicode(X)*, *unparse_url(obj)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.9.73 hex(*X*)

Returns a string which is the upper-case hexadecimal rendering of the content of its argument.

Parameters

- **X*** — The blob to convert to hexadecimal

Examples

To get the hexadecimal rendering of the string 'abc':

```
;SELECT hex('abc')
616263
```

12.9.74 humanize_duration(*secs*)

Format the given seconds value as an abbreviated duration string

PRQL Name: humanize.duration

Parameters

- **secs*** — The duration in seconds

Examples

To format a duration:

```
;SELECT humanize_duration(15 * 60)
15m00s
```

To format a sub-second value:

```
;SELECT humanize_duration(1.5)
1s500ms
```

See Also

anonymize(value), *char(X)*, *charindex(needle, haystack, [start])*, *concat_ws(sep, X)*, *concat(X)*, *date(timestring, modifier)*, *datetime(timestring, modifier)*, *decode(value, algorithm)*, *encode(value, algorithm)*, *endswith(str, suffix)*, *extract(str)*, *fuzzy_match(pattern, str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *gunzip(b)*, *gzip(value)*, *humanize_file_size(value)*, *humanize_id(id)*, *instr(haystack, needle)*, *julianday(timestring, modifier)*, *leftstr(str, N)*, *length(str)*, *logfmt2json(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *parse_url(url)*, *pretty_print(str)*, *printf(format, X)*, *proper(str)*, *regexp_capture_into_json(string, pattern, [options])*, *regexp_capture(string, pattern)*, *regexp_match(re, str)*, *regexp_replace(str, re, repl)*, *replace(str, old, replacement)*, *replicate(str, N)*, *reverse(str)*, *rightstr(str, N)*, *rtrim(str, [chars])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *strftime(format, timestring, modifier)*, *substr(str, start, [size])*, *time(timestring, modifier)*, *timediff(time1, time2)*, *timeslice(time, slice)*, *timezone(tz, ts)*, *timezone(tz, ts)*, *trim(str, [chars])*, *unhex(X)*, *unicode(X)*, *unparse_url(obj)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.9.75 humanize_file_size(*value*)

Format the given file size as a human-friendly string

PRQL Name: humanize.file_size

Parameters

- **value*** — The file size to format

Examples

To format an amount:

```
;SELECT humanize_file_size(10 * 1024 * 1024)
10.0MB
```

See Also

anonymize(value), *char(X)*, *charindex(needle, haystack, [start])*, *concat_ws(sep, X)*, *concat(X)*, *decode(value, algorithm)*, *encode(value, algorithm)*, *endswith(str, suffix)*, *extract(str)*, *fuzzy_match(pattern, str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *gunzip(b)*, *gzip(value)*, *humanize_duration(secs)*, *humanize_id(id)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *logfmt2json(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *parse_url(url)*, *pretty_print(str)*, *printf(format, X)*, *proper(str)*, *regexp_capture_into_json(string, pattern, [options])*, *regexp_capture(string, pattern)*, *regexp_match(re, str)*, *regexp_replace(str, re, repl)*, *replace(str, old, replacement)*, *replicate(str, N)*, *reverse(str)*, *rightstr(str, N)*, *rtrim(str, [chars])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *timezone(tz, ts)*, *trim(str, [chars])*, *unhex(X)*, *unicode(X)*, *unparse_url(obj)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.9.76 humanize_id(*id*)

Colorize the given ID using ANSI escape codes.

PRQL Name: humanize.id

Parameters

- **id*** — The identifier to color

Examples

To colorize the ID 'cluster1':

```
;SELECT humanize_id('cluster1')
cluster1
```

See Also

anonymize(value), *char(X)*, *charindex(needle, haystack, [start])*, *concat_ws(sep, X)*, *concat(X)*, *decode(value, algorithm)*, *encode(value, algorithm)*, *endswith(str, suffix)*, *extract(str)*, *fuzzy_match(pattern, str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *gunzip(b)*, *gzip(value)*, *humanize_duration(secs)*, *humanize_file_size(value)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *logfmt2json(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *parse_url(url)*, *pretty_print(str)*, *printf(format, X)*, *proper(str)*, *regexp_capture_into_json(string, pattern, [options])*, *regexp_capture(string, pattern)*, *regexp_match(re, str)*, *regexp_replace(str, re, repl)*, *replace(str, old, replacement)*, *replicate(str, N)*, *reverse(str)*, *rightstr(str, N)*, *rtrim(str, [chars])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *timezone(tz, ts)*, *trim(str, [chars])*, *unhex(X)*, *unicode(X)*, *unparse_url(obj)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.9.77 ifnull(*X*, *Y*)

Returns a copy of its first non-NULL argument, or NULL if both arguments are NULL

Parameters

- **X*** — A value to check for NULL-ness
- **Y*** — A value to check for NULL-ness

Examples

To get the first non-null value between null and zero:

```
;SELECT ifnull(null, 0)
0
```

12.9.78 instr(*haystack*, *needle*)

Finds the first occurrence of the needle within the haystack and returns the number of prior characters plus 1, or 0 if the needle was not found

Parameters

- **haystack*** — The string to search within
- **needle*** — The string to look for in the haystack

Examples

To test get the position of 'b' in the string 'abc':

```
;SELECT instr('abc', 'b')
2
```

See Also

anonymize(value), *char(X)*, *charindex(needle, haystack, [start])*, *concat_ws(sep, X)*, *concat(X)*, *decode(value, algorithm)*, *encode(value, algorithm)*, *endswith(str, suffix)*, *extract(str)*,

fuzzy_match(pattern, str), *group_concat(X, [sep])*, *group_spooky_hash(str)*, *gunzip(b)*, *gzip(value)*, *humanize_duration(secs)*, *humanize_file_size(value)*, *humanize_id(id)*, *leftstr(str, N)*, *length(str)*, *logfmt2json(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *parse_url(url)*, *pretty_print(str)*, *printf(format, X)*, *proper(str)*, *regexp_capture_into_json(string, pattern, [options])*, *regexp_capture(string, pattern)*, *regexp_match(re, str)*, *regexp_replace(str, re, repl)*, *replace(str, old, replacement)*, *replicate(str, N)*, *reverse(str)*, *rightstr(str, N)*, *rtrim(str, [chars])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *timezone(tz, ts)*, *trim(str, [chars])*, *unhex(X)*, *unicode(X)*, *unparse_url(obj)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.9.79 jget(*json*, *ptr*, *[default]*)

Get the value from a JSON object using a JSON-Pointer.

PRQL Name: json.get

Parameters

- **json*** — The JSON object to query.
- **ptr*** — The JSON-Pointer to lookup in the object.
- **default** — The default value if the value was not found

Examples

To get the root of a JSON value:

```
;SELECT jget('1', '')
1
```

To get the property named 'b' in a JSON object:

```
;SELECT jget('{ "a": 1, "b": 2 }', '/b')
2
```

To get the 'msg' property and return a default if it does not exist:

```
;SELECT jget(null, '/msg', 'Hello')
Hello
```

See Also

json_array_length(X, [P]), *json_array(X)*, *json_concat(json, value)*, *json_contains(json, value)*, *json_each(X, [P])*, *json_extract(X, P)*, *json_group_array(value)*, *json_group_object(name, value)*, *json_insert(X, P, Y)*, *json_object(N, V)*, *json_quote(X)*, *json_remove(X, P)*, *json_replace(X, P, Y)*, *json_set(X, P, Y)*, *json_tree(X, [P])*, *json_type(X, [P])*, *json_valid(X)*, *json(X)*, *yaml_to_json(yaml)*

12.9.80 joinpath(*path*)

Join components of a path together.

PRQL Name: fs.join

Parameters

- **path** — One or more path components to join together. If an argument starts with a forward or backward slash, it will be considered an absolute path and any preceding elements will be ignored.

Examples

To join a directory and file name into a relative path:

```
;SELECT joinpath('foo', 'bar')
foo/bar
```

To join an empty component with other names into a relative path:

```
;SELECT joinpath('', 'foo', 'bar')
foo/bar
```

To create an absolute path with two path components:

```
;SELECT joinpath('/', 'foo', 'bar')
/foo/bar
```

To create an absolute path from a path component that starts with a forward slash:

```
;SELECT joinpath('/', 'foo', '/bar')
/bar
```

See Also

basename(path), *dirname(path)*, *readlink(path)*, *realpath(path)*

12.9.81 json(*X*)

Verifies that its argument is valid JSON and returns a minified version or throws an error.

Parameters

- **X*** — The string to interpret as JSON.

See Also

jget(json, ptr, [default]), *json_array_length(X, [P])*, *json_array(X)*, *json_concat(json, value)*, *json_contains(json, value)*, *json_each(X, [P])*, *json_extract(X, P)*, *json_group_array(value)*, *json_group_object(name, value)*, *json_insert(X, P, Y)*, *json_object(N, V)*, *json_quote(X)*, *json_remove(X, P)*, *json_replace(X, P, Y)*, *json_set(X, P, Y)*, *json_tree(X, [P])*, *json_type(X, [P])*, *json_valid(X)*, *yaml_to_json(yaml)*

12.9.82 json_array(*X*)

Constructs a JSON array from its arguments.

Parameters

- **X** — The values of the JSON array

Examples

To create an array of all types:

```
;SELECT json_array(NULL, 1, 2.1, 'three', json_array(4), json_object('five
↪', 'six'))
[null,1,2.1,"three",[4],{"five":"six"}]
```

To create an empty array:

```
;SELECT json_array()
[]
```

See Also

jget(json, ptr, [default]), *json_array_length(X, [P])*, *json_concat(json, value)*, *json_contains(json, value)*, *json_each(X, [P])*, *json_extract(X, P)*, *json_group_array(value)*, *json_group_object(name, value)*, *json_insert(X, P, Y)*, *json_object(N, V)*, *json_quote(X)*, *json_remove(X, P)*, *json_replace(X, P, Y)*, *json_set(X, P, Y)*, *json_tree(X, [P])*, *json_type(X, [P])*, *json_valid(X)*, *json(X)*, *yaml_to_json(yaml)*

12.9.83 json_array_length(*X*, *[P]*)

Returns the length of a JSON array.

Parameters

- **X*** — The JSON object.
- **P** — The path to the array in 'X'.

Examples

To get the length of an array:

```
;SELECT json_array_length('[1, 2, 3]')
3
```

To get the length of a nested array:

```
;SELECT json_array_length('{"arr": [1, 2, 3]}', '$.arr')
3
```

See Also

jget(json, ptr, [default]), *json_array(X)*, *json_concat(json, value)*, *json_contains(json, value)*, *json_each(X, [P])*, *json_extract(X, P)*, *json_group_array(value)*, *json_group_object(name, value)*, *json_insert(X, P, Y)*, *json_object(N, V)*, *json_quote(X)*, *json_remove(X, P)*, *json_replace(X, P, Y)*, *json_set(X, P, Y)*, *json_tree(X, [P])*, *json_type(X, [P])*, *json_valid(X)*, *json(X)*, *yaml_to_json(yaml)*

12.9.84 json_concat(*json*, *value*)

Returns an array with the given values concatenated onto the end. If the initial value is null, the result will be an array with the given elements. If the initial value is an array, the result will be an array with the given values at the end. If the initial value is not null or an array, the result will be an array with two elements: the initial value and the given value.

PRQL Name: json.concat

Parameters

- **json*** — The initial JSON value.
- **value** — The value(s) to add to the end of the array.

Examples

To append the number 4 to null:

```
;SELECT json_concat(NULL, 4)
[4]
```

To append 4 and 5 to the array [1, 2, 3]:

```
;SELECT json_concat('[1, 2, 3]', 4, 5)
[1,2,3,4,5]
```

To concatenate two arrays together:

```
;SELECT json_concat('[1, 2, 3]', json('[4, 5]'))
[1,2,3,4,5]
```

See Also

jget(json, ptr, [default]), *json_array_length(X, [P])*, *json_array(X)*, *json_contains(json, value)*, *json_each(X, [P])*, *json_extract(X, P)*, *json_group_array(value)*, *json_group_object(name, value)*, *json_insert(X, P, Y)*, *json_object(N, V)*, *json_quote(X)*, *json_remove(X, P)*, *json_replace(X, P, Y)*, *json_set(X, P, Y)*, *json_tree(X, [P])*, *json_type(X, [P])*, *json_valid(X)*, *json(X)*, *yaml_to_json(yaml)*

12.9.85 json_contains(*json*, *value*)

Check if a JSON value contains the given element.

PRQL Name: json.contains

Parameters

- **json*** — The JSON value to query.
- **value*** — The value to look for in the first argument

Examples

To test if a JSON array contains the number 4:

```
;SELECT json_contains('[1, 2, 3]', 4)
0
```

To test if a JSON array contains the string 'def':

```
;SELECT json_contains('["abc", "def"]', 'def')
1
```

See Also

jget(json, ptr, [default]), *json_array_length(X, [P])*, *json_array(X)*, *json_concat(json, value)*, *json_each(X, [P])*, *json_extract(X, P)*, *json_group_array(value)*, *json_group_object(name, value)*, *json_insert(X, P, Y)*, *json_object(N, V)*, *json_quote(X)*, *json_remove(X, P)*, *json_replace(X, P, Y)*, *json_set(X, P, Y)*, *json_tree(X, [P])*, *json_type(X, [P])*, *json_valid(X)*, *json(X)*, *yaml_to_json(yaml)*

12.9.86 json_each(*X*, *[P]*)

A table-valued-function that returns the children of the top-level JSON value

Parameters

- **X*** — The JSON value to query
- **P** — The path to the value to query

Examples

To iterate over an array:

```
;SELECT * FROM json_each('[null,1,"two",{"three":4.5}]')
key     value      type    atom  id parent fullkey path
  0        <NULL> null    <NULL>  2 <NULL> $[0]    $
  1             1 integer      1  3 <NULL> $[1]    $
  2           two text       two  5 <NULL> $[2]    $
  3 {"three":4.5} object  <NULL>  9 <NULL> $[3]    $
```

See Also

jget(json, ptr, [default]), *json_array_length(X, [P])*, *json_array(X)*, *json_concat(json, value)*, *json_contains(json, value)*, *json_extract(X, P)*, *json_group_array(value)*, *json_group_object(name, value)*, *json_insert(X, P, Y)*, *json_object(N, V)*, *json_quote(X)*, *json_remove(X, P)*, *json_replace(X, P, Y)*, *json_set(X, P, Y)*, *json_tree(X, [P])*, *json_type(X, [P])*, *json_valid(X)*, *json(X)*, *yaml_to_json(yaml)*

12.9.87 json_extract(*X*, *P*)

Returns the value(s) from the given JSON at the given path(s).

Parameters

- **X*** — The JSON value.
- **P** — The path to extract.

Examples

To get a number:

```
;SELECT json_extract('{"num": 1}', '$.num')
1
```

To get two numbers:

```
;SELECT json_extract('{"num": 1, "val": 2}', '$.num', '$.val')
[1,2]
```

To get an object:

```
;SELECT json_extract('{"obj": {"sub": 1}}', '$.obj')
{"sub":1}
```

See Also

jget(json, ptr, [default]), *json_array_length(X, [P])*, *json_array(X)*, *json_concat(json, value)*, *json_contains(json, value)*, *json_each(X, [P])*, *json_group_array(value)*, *json_group_object(name, value)*, *json_insert(X, P, Y)*, *json_object(N, V)*, *json_quote(X)*, *json_remove(X, P)*, *json_replace(X,*

P, Y), *json_set(X, P, Y)*, *json_tree(X, [P])*, *json_type(X, [P])*, *json_valid(X)*, *json(X)*, *yaml_to_json(yaml)*

12.9.88 json_group_array(*value*)

Collect the given values from a query into a JSON array

PRQL Name: json.group_array

Parameters

- **value** — The values to append to the array

Examples

To create an array from arguments:

```
;SELECT json_group_array('one', 2, 3.4)
["one",2,3.3999999999999999112]
```

To create an array from a column of values:

```
;SELECT json_group_array(column1) FROM (VALUES (1), (2), (3))
[1,2,3]
```

See Also

jget(json, ptr, [default]), *json_array_length(X, [P])*, *json_array(X)*, *json_concat(json, value)*, *json_contains(json, value)*, *json_each(X, [P])*, *json_extract(X, P)*, *json_group_object(name, value)*, *json_insert(X, P, Y)*, *json_object(N, V)*, *json_quote(X)*, *json_remove(X, P)*, *json_replace(X, P, Y)*, *json_set(X, P, Y)*, *json_tree(X, [P])*, *json_type(X, [P])*, *json_valid(X)*, *json(X)*, *yaml_to_json(yaml)*

12.9.89 json_group_object(*name*, *value*)

Collect the given values from a query into a JSON object

PRQL Name: json.group_object

Parameters

- **name*** — The property name for the value
- **value** — The value to add to the object

Examples

To create an object from arguments:

```
;SELECT json_group_object('a', 1, 'b', 2)
{"a":1,"b":2}
```

To create an object from a pair of columns:

```
;SELECT json_group_object(column1, column2) FROM (VALUES ('a', 1), ('b',␣
↪2))
{"a":1,"b":2}
```

See Also

jget(json, ptr, [default]), *json_array_length(X, [P])*, *json_array(X)*, *json_concat(json, value)*, *json_contains(json, value)*, *json_each(X, [P])*, *json_extract(X, P)*, *json_group_array(value)*, *json_insert(X, P, Y)*, *json_object(N, V)*, *json_quote(X)*, *json_remove(X, P)*, *json_replace(X, P, Y)*, *json_set(X, P, Y)*, *json_tree(X, [P])*, *json_type(X, [P])*, *json_valid(X)*, *json(X)*, *yaml_to_json(yaml)*

12.9.90 json_insert(*X*, *P*, *Y*)

Inserts values into a JSON object/array at the given locations, if it does not already exist

Parameters

- **X*** — The JSON value to update
- **P*** — The path to the insertion point. A '#' array index means append the value
- **Y*** — The value to insert

Examples

To append to an array:

```
;SELECT json_insert('[1, 2]', '$[#]', 3)
[1,2,3]
```

To update an object:

```
;SELECT json_insert('{"a": 1}', '$.b', 2)
{"a":1,"b":2}
```

To ensure a value is set:

```
;SELECT json_insert('{"a": 1}', '$.a', 2)
{"a":1}
```

To update multiple values:

```
;SELECT json_insert('{"a": 1}', '$.b', 2, '$.c', 3)
{"a":1,"b":2,"c":3}
```

See Also

jget(json, ptr, [default]), *json_array_length(X, [P])*, *json_array(X)*, *json_concat(json, value)*, *json_contains(json, value)*, *json_each(X, [P])*, *json_extract(X, P)*, *json_group_array(value)*, *json_group_object(name, value)*, *json_object(N, V)*, *json_quote(X)*, *json_remove(X, P)*, *json_replace(X, P, Y)*, *json_set(X, P, Y)*, *json_tree(X, [P])*, *json_type(X, [P])*, *json_valid(X)*, *json(X)*, *yaml_to_json(yaml)*

12.9.91 json_object(*N*, *V*)

Create a JSON object from the given arguments

Parameters

- **N*** — The property name
- **V*** — The property value

Examples

To create an object:

```
;SELECT json_object('a', 1, 'b', 'c')
{"a":1,"b":"c"}
```

To create an empty object:

```
;SELECT json_object()
{}
```

See Also

jget(json, ptr, [default]), *json_array_length(X, [P])*, *json_array(X)*, *json_concat(json, value)*, *json_contains(json, value)*, *json_each(X, [P])*, *json_extract(X, P)*, *json_group_array(value)*, *json_group_object(name, value)*, *json_insert(X, P, Y)*, *json_quote(X)*, *json_remove(X, P)*, *json_replace(X, P, Y)*, *json_set(X, P, Y)*, *json_tree(X, [P])*, *json_type(X, [P])*, *json_valid(X)*, *json(X)*, *yaml_to_json(yaml)*

12.9.92 json_object_count_of(*value*)

Count the number of times the argument has been seen

PRQL Name: json.object_count_of

Parameters

- **value*** — The value to count in the object

Examples

To count the number of message for each level:

```
;SELECT json_object_count_of(log_level) FROM lnav_example_log
{"info":1,"error":1,"debug":1,"warn":1}
```

12.9.93 json_object_sum_of(*id*, *value*)

Sum values for an associated identifier

PRQL Name: json.object_sum_of

Parameters

- **id*** — The identifier to track
- **value*** — The value to sum

12.9.94 json_quote(*X*)

Returns the JSON representation of the given value, if it is not already JSON

Parameters

- **X*** — The value to convert

Examples

To convert a string:

```
;SELECT json_quote('Hello, World!')
"Hello, World!"
```

To pass through an existing JSON value:

```
;SELECT json_quote(json('"Hello, World!"'))
"Hello, World!"
```

See Also

jget(json, ptr, [default]), *json_array_length(X, [P])*, *json_array(X)*, *json_concat(json, value)*, *json_contains(json, value)*, *json_each(X, [P])*, *json_extract(X, P)*, *json_group_array(value)*, *json_group_object(name, value)*, *json_insert(X, P, Y)*, *json_object(N, V)*, *json_remove(X, P)*, *json_replace(X, P, Y)*, *json_set(X, P, Y)*, *json_tree(X, [P])*, *json_type(X, [P])*, *json_valid(X)*, *json(X)*, *yaml_to_json(yaml)*

12.9.95 json_remove(*X*, *P*)

Removes paths from a JSON value

Parameters

- **X*** — The JSON value to update
- **P** — The paths to remove

Examples

To remove elements of an array:

```
;SELECT json_remove('[1,2,3]', '$[1]', '$[1]')
[1]
```

To remove object properties:

```
;SELECT json_remove('{"a":1,"b":2}', '$.b')
{"a":1}
```

See Also

jget(json, ptr, [default]), *json_array_length(X, [P])*, *json_array(X)*, *json_concat(json, value)*, *json_contains(json, value)*, *json_each(X, [P])*, *json_extract(X, P)*, *json_group_array(value)*, *json_group_object(name, value)*, *json_insert(X, P, Y)*, *json_object(N, V)*, *json_quote(X)*, *json_replace(X, P, Y)*, *json_set(X, P, Y)*, *json_tree(X, [P])*, *json_type(X, [P])*, *json_valid(X)*, *json(X)*, *yaml_to_json(yaml)*

12.9.96 json_replace(*X*, *P*, *Y*)

Replaces existing values in a JSON object/array at the given locations

Parameters

- **X*** — The JSON value to update
- **P*** — The path to replace

- **Y*** — The new value for the property

Examples

To replace an existing value:

```
;SELECT json_replace('{"a": 1}', '$.a', 2)
{"a":2}
```

To replace a value without creating a new property:

```
;SELECT json_replace('{"a": 1}', '$.a', 2, '$.b', 3)
{"a":2}
```

See Also

jget(json, ptr, [default]), *json_array_length(X, [P])*, *json_array(X)*, *json_concat(json, value)*, *json_contains(json, value)*, *json_each(X, [P])*, *json_extract(X, P)*, *json_group_array(value)*, *json_group_object(name, value)*, *json_insert(X, P, Y)*, *json_object(N, V)*, *json_quote(X)*, *json_remove(X, P)*, *json_set(X, P, Y)*, *json_tree(X, [P])*, *json_type(X, [P])*, *json_valid(X)*, *json(X)*, *yaml_to_json(yaml)*

12.9.97 json_set(*X*, *P*, *Y*)

Inserts or replaces existing values in a JSON object/array at the given locations

Parameters

- **X*** — The JSON value to update
- **P*** — The path to the insertion point. A '#' array index means append the value
- **Y*** — The value to set

Examples

To replace an existing array element:

```
;SELECT json_set('[1, 2]', '$[1]', 3)
[1,3]
```

To replace a value and create a new property:

```
;SELECT json_set('{"a": 1}', '$.a', 2, '$.b', 3)
{"a":2,"b":3}
```

See Also

jget(json, ptr, [default]), *json_array_length(X, [P])*, *json_array(X)*, *json_concat(json, value)*, *json_contains(json, value)*, *json_each(X, [P])*, *json_extract(X, P)*, *json_group_array(value)*, *json_group_object(name, value)*, *json_insert(X, P, Y)*, *json_object(N, V)*, *json_quote(X)*, *json_remove(X, P)*, *json_replace(X, P, Y)*, *json_tree(X, [P])*, *json_type(X, [P])*, *json_valid(X)*, *json(X)*, *yaml_to_json(yaml)*

12.9.98 json_tree(*X*, *[P]*)

A table-valued-function that recursively descends through a JSON value

Parameters

- **X*** — The JSON value to query
- **P** — The path to the value to query

Examples

To iterate over an array:

```
;SELECT key,value,type,atom,fullkey,path FROM json_tree('[null,1,"two",{
↪"three":4.5}]')
 key        value       type    atom   fullkey   path
<NULL> [null,1":4.5}] array   <NULL> $          $
     0          <NULL> null    <NULL> $[0]       $
     1               1 integer      1 $[1]       $
     2             two text       two $[2]       $
     3   {"three":4.5} object  <NULL> $[3]       $
 three             4.5 real       4.5 $[3].three $[3]
```

See Also

jget(json, ptr, [default]), *json_array_length(X, [P])*, *json_array(X)*, *json_concat(json, value)*, *json_contains(json, value)*, *json_each(X, [P])*, *json_extract(X, P)*, *json_group_array(value)*, *json_group_object(name, value)*, *json_insert(X, P, Y)*, *json_object(N, V)*, *json_quote(X)*, *json_remove(X, P)*, *json_replace(X, P, Y)*, *json_set(X, P, Y)*, *json_type(X, [P])*, *json_valid(X)*, *json(X)*, *yaml_to_json(yaml)*

12.9.99 json_type(*X*, *[P]*)

Returns the type of a JSON value

Parameters

- **X*** — The JSON value to query
- **P** — The path to the value

Examples

To get the type of a value:

```
;SELECT json_type('[null,1,2.1,"three",{"four":5}]')
array
```

To get the type of an array element:

```
;SELECT json_type('[null,1,2.1,"three",{"four":5}]', '$[0]')
null
```

To get the type of a string:

```
;SELECT json_type('[null,1,2.1,"three",{"four":5}]', '$[3]')
text
```

See Also

jget(json, ptr, [default]), *json_array_length(X, [P])*, *json_array(X)*, *json_concat(json, value)*, *json_contains(json, value)*, *json_each(X, [P])*, *json_extract(X, P)*, *json_group_array(value)*, *json_group_object(name, value)*, *json_insert(X, P, Y)*, *json_object(N, V)*, *json_quote(X)*, *json_remove(X, P)*, *json_replace(X, P, Y)*, *json_set(X, P, Y)*, *json_tree(X, [P])*, *json_valid(X)*, *json(X)*, *yaml_to_json(yaml)*

12.9.100 json_valid(*X*)

Tests if the given value is valid JSON

Parameters

- **X*** — The value to check

Examples

To check an empty string:

```
;SELECT json_valid('')
0
```

To check a string:

```
;SELECT json_valid('"a"')
1
```

See Also

jget(json, ptr, [default]), *json_array_length(X, [P])*, *json_array(X)*, *json_concat(json, value)*, *json_contains(json, value)*, *json_each(X, [P])*, *json_extract(X, P)*, *json_group_array(value)*, *json_group_object(name, value)*, *json_insert(X, P, Y)*, *json_object(N, V)*, *json_quote(X)*, *json_remove(X, P)*, *json_replace(X, P, Y)*, *json_set(X, P, Y)*, *json_tree(X, [P])*, *json_type(X, [P])*, *json(X)*, *yaml_to_json(yaml)*

12.9.101 julianday(*timestring*, *modifier*)

Returns the number of days since noon in Greenwich on November 24, 4714 B.C.

Parameters

- **timestring*** — The string to convert to a date with time.
- **modifier** — A transformation that is applied to the value to the left.

Examples

To get the julian day from the timestamp '2017-01-02T03:04:05':

```
;SELECT julianday('2017-01-02T03:04:05')
2457755.627835648
```

To get the julian day from the timestamp '2017-01-02T03:04:05' plus one minute:

```
;SELECT julianday('2017-01-02T03:04:05', '+1 minute')
2457755.6285300925
```

To get the julian day from the timestamp 1491341842:

```
;SELECT julianday(1491341842, 'unixepoch')
2457848.400949074
```

See Also

date(timestring, modifier), *datetime(timestring, modifier)*, *humanize_duration(secs)*, *strftime(format, timestring, modifier)*, *time(timestring, modifier)*, *timediff(time1, time2)*, *timeslice(time, slice)*, *timezone(tz, ts)*

12.9.102 lag(*expr*, *[offset]*, *[default]*)

Returns the result of evaluating the expression against the previous row in the partition.

Parameters

- **expr*** — The expression to execute over the previous row
- **offset** — The offset from the current row in the partition
- **default** — The default value if the previous row does not exist instead of NULL

See Also

cume_dist(), *dense_rank()*, *first_value(expr)*, *last_value(expr)*, *lead(expr, [offset], [default])*, *nth_value(expr, N)*, *ntile(groups)*, *percent_rank()*, *rank()*, *row_number()*

12.9.103 last_insert_rowid()

Returns the ROWID of the last row insert from the database connection which invoked the function

---

12.9.104 last_value(*expr*)

Returns the result of evaluating the expression against the last row in the window frame.

Parameters

- **expr*** — The expression to execute over the last row

See Also

cume_dist(), *dense_rank()*, *first_value(expr)*, *lag(expr, [offset], [default])*, *lead(expr, [offset], [default])*, *nth_value(expr, N)*, *ntile(groups)*, *percent_rank()*, *rank()*, *row_number()*

12.9.105 lead(*expr*, *[offset]*, *[default]*)

Returns the result of evaluating the expression against the next row in the partition.

Parameters

- **expr*** — The expression to execute over the next row
- **offset** — The offset from the current row in the partition
- **default** — The default value if the next row does not exist instead of NULL

See Also

cume_dist(), *dense_rank()*, *first_value(expr)*, *lag(expr, [offset], [default])*, *last_value(expr)*, *nth_value(expr, N)*, *ntile(groups)*, *percent_rank()*, *rank()*, *row_number()*

12.9.106 leftstr(*str*, *N*)

Returns the N leftmost (UTF-8) characters in the given string.

Parameters

- **str*** — The string to return subset.
- **N*** — The number of characters from the left side of the string to return.

Examples

To get the first character of the string 'abc':

```
;SELECT leftstr('abc', 1)
a
```

To get the first ten characters of a string, regardless of size:

```
;SELECT leftstr('abc', 10)
abc
```

See Also

anonymize(value), *char(X)*, *charindex(needle, haystack, [start])*, *concat_ws(sep, X)*, *concat(X)*, *decode(value, algorithm)*, *encode(value, algorithm)*, *endswith(str, suffix)*, *extract(str)*, *fuzzy_match(pattern, str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *gunzip(b)*, *gzip(value)*, *humanize_duration(secs)*, *humanize_file_size(value)*, *humanize_id(id)*, *instr(haystack, needle)*, *length(str)*, *logfmt2json(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *parse_url(url)*, *pretty_print(str)*, *printf(format, X)*, *proper(str)*, *regexp_capture_into_json(string, pattern, [options])*, *regexp_capture(string, pattern)*, *regexp_match(re, str)*, *regexp_replace(str, re, repl)*, *replace(str, old, replacement)*, *replicate(str, N)*, *reverse(str)*, *rightstr(str, N)*, *rtrim(str, [chars])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *timezone(tz, ts)*, *trim(str, [chars])*, *unhex(X)*, *unicode(X)*, *unparse_url(obj)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.9.107 length(*str*)

Returns the number of characters (not bytes) in the given string prior to the first NUL character

Parameters

- **str*** — The string to determine the length of

Examples

To get the length of the string 'abc':

```
;SELECT length('abc')
3
```

See Also

anonymize(value), *char(X)*, *charindex(needle, haystack, [start])*, *concat_ws(sep, X)*, *concat(X)*, *decode(value, algorithm)*, *encode(value, algorithm)*, *endswith(str, suffix)*, *extract(str)*, *fuzzy_match(pattern, str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *gunzip(b)*, *gzip(value)*, *humanize_duration(secs)*, *humanize_file_size(value)*, *humanize_id(id)*, *instr(haystack, needle)*, *leftstr(str, N)*, *logfmt2json(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *parse_url(url)*, *pretty_print(str)*, *printf(format, X)*, *proper(str)*, *regexp_capture_into_json(string, pattern, [options])*, *regexp_capture(string, pattern)*, *regexp_match(re, str)*, *regexp_replace(str, re, repl)*, *replace(str, old, replacement)*, *replicate(str,*

N), *reverse(str)*, *rightstr(str, N)*, *rtrim(str, [chars])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *timezone(tz, ts)*, *trim(str, [chars])*, *unhex(X)*, *unicode(X)*, *unparse_url(obj)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.9.108 like(*pattern*, *str*, *[escape]*)

Match a string against a pattern

Parameters

- **pattern*** — The pattern to match. A percent symbol (%) will match zero or more characters and an underscore (_) will match a single character.
- **str*** — The string to match
- **escape** — The escape character that can be used to prefix a literal percent or underscore in the pattern.

Examples

To test if the string 'aabcc' contains the letter 'b':

```
;SELECT like('%b%', 'aabcc')
1
```

To test if the string 'aab%' ends with 'b%':

```
;SELECT like('%b:%', 'aab%', ':')
1
```

12.9.109 likelihood(*value*, *probability*)

Provides a hint to the query planner that the first argument is a boolean that is true with the given probability

Parameters

- **value*** — The boolean value to return
- **probability*** — A floating point constant between 0.0 and 1.0

12.9.110 likely(*value*)

Short-hand for likelihood(X,0.9375)

Parameters

- **value*** — The boolean value to return

12.9.111 lnav_top_file()

Return the name of the file that the top line in the current view came from.

PRQL Name: lnav.view.top_file

12.9.112 lnav_version()

Return the current version of lnav

PRQL Name: lnav.version

12.9.113 load_extension(*path*, *[entry-point]*)

Loads SQLite extensions out of the given shared library file using the given entry point.

Parameters

- **path*** — The path to the shared library containing the extension.

12.9.114 log(*x*)

Returns the natural logarithm of x

Parameters

- **x*** — The number

Examples

To get the natual logarithm of 8:

```
;SELECT log(8)
2.0794415416798357
```

See Also

abs(x), *acos(num)*, *acosh(num)*, *asin(num)*, *asinh(num)*, *atan2(y, x)*, *atan(num)*, *atanh(num)*, *atn2(y, x)*, *avg(X)*, *ceil(num)*, *degrees(radians)*, *exp(x)*, *floor(num)*, *log10(x)*, *max(X)*, *min(X)*, *pi()*, *power(base, exp)*, *radians(degrees)*, *round(num, [digits])*, *sign(num)*, *square(num)*, *sum(X)*, *total(X)*

12.9.115 log10(*x*)

Returns the base-10 logarithm of X

Parameters

- **x*** — The number

Examples

To get the logarithm of 100:

```
;SELECT log10(100)
2
```

See Also

abs(x), *acos(num)*, *acosh(num)*, *asin(num)*, *asinh(num)*, *atan2(y, x)*, *atan(num)*, *atanh(num)*, *atn2(y, x)*, *avg(X)*, *ceil(num)*, *degrees(radians)*, *exp(x)*, *floor(num)*, *log(x)*, *max(X)*, *min(X)*, *pi()*, *power(base, exp)*, *radians(degrees)*, *round(num, [digits])*, *sign(num)*, *square(num)*, *sum(X)*, *total(X)*

12.9.116 log_msg_line()

Return the starting line number of the focused log message.

PRQL Name: lnav.view.msg_line

12.9.117 log_top_datetime()

Return the timestamp of the line at the top of the log view.

PRQL Name: lnav.view.top_datetime

12.9.118 log_top_line()

Return the number of the focused line of the log view.

PRQL Name: lnav.view.top_line

12.9.119 logfmt2json(*str*)

Convert a logfmt-encoded string into JSON

PRQL Name: logfmt.to_json

Parameters

- **str*** — The logfmt message to parse

Examples

To extract key/value pairs from a log message:

```
;SELECT logfmt2json('foo=1 bar=2 name="Rolo Tomassi"')
{"foo":1,"bar":2,"name":"Rolo Tomassi"}
```

See Also

anonymize(value), *char(X)*, *charindex(needle, haystack, [start])*, *concat_ws(sep, X)*, *concat(X)*, *decode(value, algorithm)*, *encode(value, algorithm)*, *endswith(str, suffix)*, *extract(str)*, *fuzzy_match(pattern, str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *gunzip(b)*, *gzip(value)*, *humanize_duration(secs)*, *humanize_file_size(value)*, *humanize_id(id)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *parse_url(url)*, *pretty_print(str)*, *printf(format, X)*, *proper(str)*, *regexp_capture_into_json(string, pattern, [options])*, *regexp_capture(string, pattern)*, *regexp_match(re, str)*, *regexp_replace(str, re, repl)*, *replace(str, old, replacement)*, *replicate(str, N)*, *reverse(str)*, *rightstr(str, N)*, *rtrim(str, [chars])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *timezone(tz, ts)*, *trim(str, [chars])*, *unhex(X)*, *unicode(X)*, *unparse_url(obj)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.9.120 lower(*str*)

Returns a copy of the given string with all ASCII characters converted to lower case.

Parameters

- **str*** — The string to convert.

Examples

To lowercase the string 'AbC':

```
;SELECT lower('AbC')
abc
```

See Also

anonymize(value), *char(X)*, *charindex(needle, haystack, [start])*, *concat_ws(sep, X)*, *concat(X)*, *decode(value, algorithm)*, *encode(value, algorithm)*, *endswith(str, suffix)*, *extract(str)*, *fuzzy_match(pattern, str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *gunzip(b)*, *gzip(value)*, *humanize_duration(secs)*, *humanize_file_size(value)*, *humanize_id(id)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *logfmt2json(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *parse_url(url)*, *pretty_print(str)*, *printf(format, X)*, *proper(str)*, *regexp_capture_into_json(string, pattern, [options])*, *regexp_capture(string, pattern)*, *regexp_match(re, str)*, *regexp_replace(str, re, repl)*, *replace(str, old, replacement)*, *replicate(str, N)*, *reverse(str)*, *rightstr(str, N)*, *rtrim(str, [chars])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *timezone(tz, ts)*, *trim(str, [chars])*, *unhex(X)*, *unicode(X)*, *unparse_url(obj)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.9.121 ltrim(*str*, *[chars]*)

Returns a string formed by removing any and all characters that appear in the second argument from the left side of the first.

Parameters

- **str*** — The string to trim characters from the left side
- **chars** — The characters to trim. Defaults to spaces.

Examples

To trim the leading space characters from the string ' abc':

```
;SELECT ltrim('   abc')
abc
```

To trim the characters 'a' or 'b' from the left side of the string 'aaaabbbc':

```
;SELECT ltrim('aaaabbbc', 'ab')
c
```

See Also

anonymize(value), *char(X)*, *charindex(needle, haystack, [start])*, *concat_ws(sep, X)*, *concat(X)*, *decode(value, algorithm)*, *encode(value, algorithm)*, *endswith(str, suffix)*, *extract(str)*, *fuzzy_match(pattern, str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *gunzip(b)*, *gzip(value)*, *humanize_duration(secs)*, *humanize_file_size(value)*, *humanize_id(id)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *logfmt2json(str)*, *lower(str)*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *parse_url(url)*, *pretty_print(str)*, *printf(format, X)*, *proper(str)*, *regexp_capture_into_json(string, pattern, [options])*, *regexp_capture(string, pattern)*, *regexp_match(re, str)*, *regexp_replace(str,*

re, repl), *replace(str, old, replacement)*, *replicate(str, N)*, *reverse(str)*, *rightstr(str, N)*, *rtrim(str, [chars])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *timezone(tz, ts)*, *trim(str, [chars])*, *unhex(X)*, *unicode(X)*, *unparse_url(obj)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.9.122 max(*X*)

Returns the argument with the maximum value, or return NULL if any argument is NULL.

Parameters

- **X** — The numbers to find the maximum of. If only one argument is given, this function operates as an aggregate.

Examples

To get the largest value from the parameters:

```
;SELECT max(2, 1, 3)
3
```

To get the largest value from an aggregate:

```
;SELECT max(status) FROM http_status_codes
511
```

See Also

abs(x), *acos(num)*, *acosh(num)*, *asin(num)*, *asinh(num)*, *atan2(y, x)*, *atan(num)*, *atanh(num)*, *atn2(y, x)*, *avg(X)*, *ceil(num)*, *degrees(radians)*, *exp(x)*, *floor(num)*, *log10(x)*, *log(x)*, *min(X)*, *pi()*, *power(base, exp)*, *radians(degrees)*, *round(num, [digits])*, *sign(num)*, *square(num)*, *sum(X)*, *total(X)*

12.9.123 min(*X*)

Returns the argument with the minimum value, or return NULL if any argument is NULL.

Parameters

- **X** — The numbers to find the minimum of. If only one argument is given, this function operates as an aggregate.

Examples

To get the smallest value from the parameters:

```
;SELECT min(2, 1, 3)
1
```

To get the smallest value from an aggregate:

```
;SELECT min(status) FROM http_status_codes
100
```

See Also

abs(x), *acos(num)*, *acosh(num)*, *asin(num)*, *asinh(num)*, *atan2(y, x)*, *atan(num)*, *atanh(num)*, *atn2(y, x)*, *avg(X)*, *ceil(num)*, *degrees(radians)*, *exp(x)*, *floor(num)*, *log10(x)*, *log(x)*, *max(X)*, *pi()*, *power(base, exp)*, *radians(degrees)*, *round(num, [digits])*, *sign(num)*, *square(num)*, *sum(X)*, *total(X)*

12.9.124 nth_value(*expr*, *N*)

Returns the result of evaluating the expression against the nth row in the window frame.

Parameters

- **expr*** — The expression to execute over the nth row
- **N*** — The row number

See Also

cume_dist(), *dense_rank()*, *first_value(expr)*, *lag(expr, [offset], [default])*, *last_value(expr)*, *lead(expr, [offset], [default])*, *ntile(groups)*, *percent_rank()*, *rank()*, *row_number()*

12.9.125 ntile(*groups*)

Returns the number of the group that the current row is a part of

Parameters

- **groups*** — The number of groups

See Also

cume_dist(), *dense_rank()*, *first_value(expr)*, *lag(expr, [offset], [default])*, *last_value(expr)*, *lead(expr, [offset], [default])*, *nth_value(expr, N)*, *percent_rank()*, *rank()*, *row_number()*

12.9.126 nullif(*X*, *Y*)

Returns its first argument if the arguments are different and NULL if the arguments are the same.

Parameters

- **X*** — The first argument to compare.
- **Y*** — The argument to compare against the first.

Examples

To test if 1 is different from 1:

```
;SELECT nullif(1, 1)
<NULL>
```

To test if 1 is different from 2:

```
;SELECT nullif(1, 2)
1
```

12.9.127 octet_length(*X*)

Returns the number of bytes in the given string as encoded in the database

Parameters

- **X*** — The value to examine

Examples

To get the number of bytes for a string:

```
;SELECT octet_length('Hello, World!')
13
```

To get the number of bytes for a number:

```
;SELECT octet_length(42)
2
```

12.9.128 padc(*str*, *len*)

Pad the given string with enough spaces to make it centered within the given length

Parameters

- **str*** — The string to pad
- **len*** — The minimum desired length of the output string

Examples

To pad the string 'abc' to a length of six characters:

```
;SELECT padc('abc', 6) || 'def'
 abc  def
```

To pad the string 'abcdef' to a length of eight characters:

```
;SELECT padc('abcdef', 8) || 'ghi'
 abcdef ghi
```

See Also

anonymize(value), *char(X)*, *charindex(needle, haystack, [start])*, *concat_ws(sep, X)*, *concat(X)*, *decode(value, algorithm)*, *encode(value, algorithm)*, *endswith(str, suffix)*, *extract(str)*, *fuzzy_match(pattern, str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *gunzip(b)*, *gzip(value)*, *humanize_duration(secs)*, *humanize_file_size(value)*, *humanize_id(id)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *logfmt2json(str)*, *lower(str)*, *ltrim(str, [chars])*, *padl(str, len)*, *padr(str, len)*, *parse_url(url)*, *pretty_print(str)*, *printf(format, X)*, *proper(str)*, *regexp_capture_into_json(string, pattern, [options])*, *regexp_capture(string, pattern)*, *regexp_match(re, str)*, *regexp_replace(str, re, repl)*, *replace(str, old, replacement)*, *replicate(str, N)*, *reverse(str)*, *rightstr(str, N)*, *rtrim(str, [chars])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *timezone(tz, ts)*, *trim(str, [chars])*, *unhex(X)*, *unicode(X)*, *unparse_url(obj)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.9.129 padl(*str*, *len*)

Pad the given string with leading spaces until it reaches the desired length

Parameters

- **str*** — The string to pad
- **len*** — The minimum desired length of the output string

Examples

To pad the string 'abc' to a length of six characters:

```
;SELECT padl('abc', 6)
   abc
```

To pad the string 'abcdef' to a length of four characters:

```
;SELECT padl('abcdef', 4)
abcdef
```

See Also

anonymize(value), *char(X)*, *charindex(needle, haystack, [start])*, *concat_ws(sep, X)*, *concat(X)*, *decode(value, algorithm)*, *encode(value, algorithm)*, *endswith(str, suffix)*, *extract(str)*, *fuzzy_match(pattern, str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *gunzip(b)*, *gzip(value)*, *humanize_duration(secs)*, *humanize_file_size(value)*, *humanize_id(id)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *logfmt2json(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padr(str, len)*, *parse_url(url)*, *pretty_print(str)*, *printf(format, X)*, *proper(str)*, *regexp_capture_into_json(string, pattern, [options])*, *regexp_capture(string, pattern)*, *regexp_match(re, str)*, *regexp_replace(str, re, repl)*, *replace(str, old, replacement)*, *replicate(str, N)*, *reverse(str)*, *rightstr(str, N)*, *rtrim(str, [chars])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *timezone(tz, ts)*, *trim(str, [chars])*, *unhex(X)*, *unicode(X)*, *unparse_url(obj)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.9.130 padr(*str*, *len*)

Pad the given string with trailing spaces until it reaches the desired length

Parameters

- **str*** — The string to pad
- **len*** — The minimum desired length of the output string

Examples

To pad the string 'abc' to a length of six characters:

```
;SELECT padr('abc', 6) || 'def'
abc   def
```

To pad the string 'abcdef' to a length of four characters:

```
;SELECT padr('abcdef', 4) || 'ghi'
abcdefghi
```

See Also

anonymize(value), *char(X)*, *charindex(needle, haystack, [start])*, *concat_ws(sep, X)*, *concat(X)*, *decode(value, algorithm)*, *encode(value, algorithm)*, *endswith(str, suffix)*, *extract(str)*, *fuzzy_match(pattern, str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *gunzip(b)*, *gzip(value)*, *humanize_duration(secs)*, *humanize_file_size(value)*, *humanize_id(id)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *logfmt2json(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *parse_url(url)*, *pretty_print(str)*, *printf(format, X)*, *proper(str)*, *regexp_capture_into_json(string, pattern, [options])*, *regexp_capture(string, pattern)*, *regexp_match(re, str)*, *regexp_replace(str, re, repl)*, *replace(str, old, replacement)*, *replicate(str, N)*, *reverse(str)*, *rightstr(str, N)*, *rtrim(str, [chars])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *timezone(tz, ts)*, *trim(str, [chars])*, *unhex(X)*, *unicode(X)*, *unparse_url(obj)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.9.131 parse_url(*url*)

Parse a URL and return the components in a JSON object. Limitations: not all URL schemes are supported and repeated query parameters are not captured.

Parameters

- **url*** — The URL to parse

Examples

To parse the URL 'https://example.com/search?q=hello%20world':

```
;SELECT parse_url('https://example.com/search?q=hello%20world')
{"scheme":"https","username":null,"password":null,"host":"example.com",
↪"port":null,"path":"/search","query":"q=hello%20world","parameters":{"q":
↪"hello world"},"fragment":null}
```

To parse the URL 'https://alice@{[}fe80::14ff:4ee5:1215:2fb2]':

```
;SELECT parse_url('https://alice@[fe80::14ff:4ee5:1215:2fb2]')
{"scheme":"https","username":"alice","password":null,"host":
↪"[fe80::14ff:4ee5:1215:2fb2]","port":null,"path":"/","query":null,
↪"parameters":null,"fragment":null}
```

See Also

anonymize(value), *char(X)*, *charindex(needle, haystack, [start])*, *concat_ws(sep, X)*, *concat(X)*, *decode(value, algorithm)*, *encode(value, algorithm)*, *endswith(str, suffix)*, *extract(str)*, *fuzzy_match(pattern, str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *gunzip(b)*, *gzip(value)*, *humanize_duration(secs)*, *humanize_file_size(value)*, *humanize_id(id)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *logfmt2json(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *pretty_print(str)*, *printf(format, X)*, *proper(str)*, *regexp_capture_into_json(string, pattern, [options])*, *regexp_capture(string, pattern)*, *regexp_match(re, str)*, *regexp_replace(str, re, repl)*, *replace(str, old, replacement)*, *replicate(str, N)*, *reverse(str)*, *rightstr(str, N)*, *rtrim(str, [chars])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *timezone(tz, ts)*, *trim(str, [chars])*, *unhex(X)*, *unicode(X)*, *unparse_url(obj)*, *unparse_url(obj)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.9.132 percent_rank()

Returns (rank - 1) / (partition-rows - 1)

See Also

cume_dist(), *dense_rank()*, *first_value(expr)*, *lag(expr, [offset], [default])*, *last_value(expr)*, *lead(expr, [offset], [default])*, *nth_value(expr, N)*, *ntile(groups)*, *rank()*, *row_number()*

12.9.133 pi()

Returns the value of PI

Examples

To get the value of PI:

```
;SELECT pi()
3.141592653589793
```

See Also

abs(x), *acos(num)*, *acosh(num)*, *asin(num)*, *asinh(num)*, *atan2(y, x)*, *atan(num)*, *atanh(num)*, *atn2(y, x)*, *avg(X)*, *ceil(num)*, *degrees(radians)*, *exp(x)*, *floor(num)*, *log10(x)*, *log(x)*, *max(X)*, *min(X)*, *power(base, exp)*, *radians(degrees)*, *round(num, [digits])*, *sign(num)*, *square(num)*, *sum(X)*, *total(X)*

12.9.134 power(*base*, *exp*)

Returns the base to the given exponent

Parameters

- **base*** — The base number
- **exp*** — The exponent

Examples

To raise two to the power of three:

```
;SELECT power(2, 3)
8
```

See Also

abs(x), *acos(num)*, *acosh(num)*, *asin(num)*, *asinh(num)*, *atan2(y, x)*, *atan(num)*, *atanh(num)*, *atn2(y, x)*, *avg(X)*, *ceil(num)*, *degrees(radians)*, *exp(x)*, *floor(num)*, *log10(x)*, *log(x)*, *max(X)*, *min(X)*, *pi()*, *radians(degrees)*, *round(num, [digits])*, *sign(num)*, *square(num)*, *sum(X)*, *total(X)*

12.9.135 pretty_print(*str*)

Pretty-print the given string

PRQL Name: text.pretty

Parameters

- **str*** — The string to format

Examples

To pretty-print the string '{"scheme": "https", "host": "example.com"}':

```
;SELECT pretty_print('{"scheme": "https", "host": "example.com"}')
{
    "scheme": "https",
    "host": "example.com"
}
```

See Also

anonymize(value), *char(X)*, *charindex(needle, haystack, [start])*, *concat_ws(sep, X)*, *concat(X)*, *decode(value, algorithm)*, *encode(value, algorithm)*, *endswith(str, suffix)*, *extract(str)*, *fuzzy_match(pattern, str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *gunzip(b)*, *gzip(value)*, *humanize_duration(secs)*, *humanize_file_size(value)*, *humanize_id(id)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *logfmt2json(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *parse_url(url)*, *printf(format, X)*, *proper(str)*, *regexp_capture_into_json(string, pattern, [options])*, *regexp_capture(string, pattern)*, *regexp_match(re, str)*, *regexp_replace(str, re, repl)*, *replace(str, old, replacement)*, *replicate(str, N)*, *reverse(str)*, *rightstr(str, N)*, *rtrim(str,*

[chars]), *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *timezone(tz, ts)*, *trim(str, [chars])*, *unhex(X)*, *unicode(X)*, *unparse_url(obj)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.9.136 printf(*format*, *X*)

Returns a string with this functions arguments substituted into the given format. Substitution points are specified using percent (%) options, much like the standard C printf() function.

Parameters

- **format*** — The format of the string to return.
- **X*** — The argument to substitute at a given position in the format.

Examples

To substitute 'World' into the string 'Hello, %s!':

```
;SELECT printf('Hello, %s!', 'World')
Hello, World!
```

To right-align 'small' in the string 'align:' with a column width of 10:

```
;SELECT printf('align: % 10s', 'small')
align:      small
```

To format 11 with a width of five characters and leading zeroes:

```
;SELECT printf('value: %05d', 11)
value: 00011
```

See Also

anonymize(value), *char(X)*, *charindex(needle, haystack, [start])*, *concat_ws(sep, X)*, *concat(X)*, *decode(value, algorithm)*, *encode(value, algorithm)*, *endswith(str, suffix)*, *extract(str)*, *fuzzy_match(pattern, str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *gunzip(b)*, *gzip(value)*, *humanize_duration(secs)*, *humanize_file_size(value)*, *humanize_id(id)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *logfmt2json(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *parse_url(url)*, *pretty_print(str)*, *proper(str)*, *regexp_capture_into_json(string, pattern, [options])*, *regexp_capture(string, pattern)*, *regexp_match(re, str)*, *regexp_replace(str, re, repl)*, *replace(str, old, replacement)*, *replicate(str, N)*, *reverse(str)*, *rightstr(str, N)*, *rtrim(str, [chars])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *timezone(tz, ts)*, *trim(str, [chars])*, *unhex(X)*, *unicode(X)*, *unparse_url(obj)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.9.137 proper(*str*)

Capitalize the first character of words in the given string

Parameters

- **str*** — The string to capitalize.

Examples

To capitalize the words in the string 'hello, world!':

```
;SELECT proper('hello, world!')
Hello, World!
```

See Also

anonymize(value), *char(X)*, *charindex(needle, haystack, [start])*, *concat_ws(sep, X)*, *concat(X)*, *decode(value, algorithm)*, *encode(value, algorithm)*, *endswith(str, suffix)*, *extract(str)*, *fuzzy_match(pattern, str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *gunzip(b)*, *gzip(value)*, *humanize_duration(secs)*, *humanize_file_size(value)*, *humanize_id(id)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *logfmt2json(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *parse_url(url)*, *pretty_print(str)*, *printf(format, X)*, *regexp_capture_into_json(string, pattern, [options])*, *regexp_capture(string, pattern)*, *regexp_match(re, str)*, *regexp_replace(str, re, repl)*, *replace(str, old, replacement)*, *replicate(str, N)*, *reverse(str)*, *rightstr(str, N)*, *rtrim(str, [chars])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *timezone(tz, ts)*, *trim(str, [chars])*, *unhex(X)*, *unicode(X)*, *unparse_url(obj)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.9.138 quote(*X*)

Returns the text of an SQL literal which is the value of its argument suitable for inclusion into an SQL statement.

Parameters

- **X*** — The string to quote.

Examples

To quote the string 'abc':

```
;SELECT quote('abc')
'abc'
```

To quote the string 'abc'123':

```
;SELECT quote('abc''123')
'abc''123'
```

12.9.139 radians(*degrees*)

Converts degrees to radians

Parameters

- **degrees*** — The degrees value to convert to radians

Examples

To convert 180 degrees to radians:

```
;SELECT radians(180)
3.141592653589793
```

See Also

abs(x), *acos(num)*, *acosh(num)*, *asin(num)*, *asinh(num)*, *atan2(y, x)*, *atan(num)*, *atanh(num)*, *atn2(y, x)*, *avg(X)*, *ceil(num)*, *degrees(radians)*, *exp(x)*, *floor(num)*, *log10(x)*, *log(x)*, *max(X)*, *min(X)*, *pi()*, *power(base, exp)*, *round(num, [digits])*, *sign(num)*, *square(num)*, *sum(X)*, *total(X)*

12.9.140 raise_error(*msg*, *[reason]*, *[help]*)

Raises an error with the given message when executed

Parameters

- **msg*** — The error message
- **reason** — The reason the error occurred
- **help** — A possible resolution to the error

Examples

To raise an error if a variable is not set:

```
;SELECT ifnull($val, raise_error('please set $val', 'because'))
  error: please set $val
reason: because
--> raise_error:1
| SELECT ifnull($val, raise_error('please set $val', 'because'))
```

12.9.141 random()

Returns a pseudo-random integer between -9223372036854775808 and +9223372036854775807.

12.9.142 randomblob(*N*)

Return an N-byte blob containing pseudo-random bytes.

Parameters

- **N*** — The size of the blob in bytes.

12.9.143 rank()

Returns the row_number() of the first peer in each group with gaps

See Also

cume_dist(), *dense_rank()*, *first_value(expr)*, *lag(expr, [offset], [default])*, *last_value(expr)*, *lead(expr, [offset], [default])*, *nth_value(expr, N)*, *ntile(groups)*, *percent_rank()*, *row_number()*

12.9.144 readlink(*path*)

Read the target of a symbolic link.

PRQL Name: fs.readlink

Parameters

- **path*** — The path to the symbolic link.

See Also

basename(path), *dirname(path)*, *joinpath(path)*, *realpath(path)*

12.9.145 realpath(*path*)

Returns the resolved version of the given path, expanding symbolic links and resolving '.' and '..' references.

PRQL Name: fs.realpath

Parameters

- **path*** — The path to resolve.

See Also

basename(path), *dirname(path)*, *joinpath(path)*, *readlink(path)*

12.9.146 regexp(*re*, *str*)

Test if a string matches a regular expression

Parameters

- **re*** — The regular expression to use
- **str*** — The string to test against the regular expression

---

12.9.147 regexp_capture(*string*, *pattern*)

A table-valued function that executes a regular-expression over a string and returns the captured values. If the regex only matches a subset of the input string, it will be rerun on the remaining parts of the string until no more matches are found.

Parameters

- **string*** — The string to match against the given pattern.
- **pattern*** — The regular expression to match.

Examples

To extract the key/value pairs 'a'/1 and 'b'/2 from the string 'a=1; b=2':

```
;SELECT * FROM regexp_capture('a=1; b=2', '(\w+)=(\d+)')
match_index capture_index capture_name capture_count range_start range_
↪stop content
          0             0 <NULL>                    3           1         ␣
```

(continues on next page)

(continued from previous page)

```
↪4      a=1
          0              1 <NULL>                3              1          ␣
↪2        a
          0              2 <NULL>                3              3          ␣
↪4        1
          1              0 <NULL>                3              6          ␣
↪9      b=2
          1              1 <NULL>                3              6          ␣
↪7        b
          1              2 <NULL>                3              8          ␣
↪9        2
```

See Also

anonymize(value), *char(X)*, *charindex(needle, haystack, [start])*, *concat_ws(sep, X)*, *concat(X)*, *decode(value, algorithm)*, *encode(value, algorithm)*, *endswith(str, suffix)*, *extract(str)*, *fuzzy_match(pattern, str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *gunzip(b)*, *gzip(value)*, *humanize_duration(secs)*, *humanize_file_size(value)*, *humanize_id(id)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *logfmt2json(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *parse_url(url)*, *pretty_print(str)*, *printf(format, X)*, *proper(str)*, *regexp_capture_into_json(string, pattern, [options])*, *regexp_match(re, str)*, *regexp_replace(str, re, repl)*, *replace(str, old, replacement)*, *replicate(str, N)*, *reverse(str)*, *rightstr(str, N)*, *rtrim(str, [chars])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *timezone(tz, ts)*, *trim(str, [chars])*, *unhex(X)*, *unicode(X)*, *unparse_url(obj)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.9.148 regexp_capture_into_json(*string*, *pattern*, *[options]*)

A table-valued function that executes a regular-expression over a string and returns the captured values as a JSON object. If the regex only matches a subset of the input string, it will be rerun on the remaining parts of the string until no more matches are found.

Parameters

- **string*** — The string to match against the given pattern.
- **pattern*** — The regular expression to match.
- **options** — A JSON object with the following option: convert-numbers - True (default) if text that looks like numeric data should be converted to JSON numbers, false if they should be captured as strings.

Examples

To extract the key/value pairs ‘a’/1 and ‘b’/2 from the string ‘a=1; b=2’:

```
;SELECT * FROM regexp_capture_into_json('a=1; b=2', '(\w+)=(\d+)')
match_index     content
          0 {"col_0_1":1}
          1 {"col_0_1":2}
```

See Also

anonymize(value), *char(X)*, *charindex(needle, haystack, [start])*, *concat_ws(sep, X)*, *concat(X)*, *decode(value, algorithm)*, *encode(value, algorithm)*, *endswith(str, suffix)*, *extract(str)*, *fuzzy_match(pattern, str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *gunzip(b)*, *gzip(value)*, *humanize_duration(secs)*, *humanize_file_size(value)*, *humanize_id(id)*, *instr(haystack, needle)*,

leftstr(str, N), *length(str)*, *logfmt2json(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *parse_url(url)*, *pretty_print(str)*, *printf(format, X)*, *proper(str)*, *regexp_capture(string, pattern)*, *regexp_match(re, str)*, *regexp_replace(str, re, repl)*, *replace(str, old, replacement)*, *replicate(str, N)*, *reverse(str)*, *rightstr(str, N)*, *rtrim(str, [chars])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *timezone(tz, ts)*, *trim(str, [chars])*, *unhex(X)*, *unicode(X)*, *unparse_url(obj)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.9.149 regexp_match(*re*, *str*)

Match a string against a regular expression and return the capture groups as JSON.

PRQL Name: text.regexp_match

Parameters

- **re*** — The regular expression to use
- **str*** — The string to test against the regular expression

Examples

To capture the digits from the string '123':

```
;SELECT regexp_match('(\d+)', '123')
123
```

To capture a number and word into a JSON object with the properties 'col_0' and 'col_1':

```
;SELECT regexp_match('(\d+) (\w+)', '123 four')
{"col_0":123,"col_1":"four"}
```

To capture a number and word into a JSON object with the named properties 'num' and 'str':

```
;SELECT regexp_match('(?<num>\d+) (?<str>\w+)', '123 four')
{"num":123,"str":"four"}
```

See Also

anonymize(value), *char(X)*, *charindex(needle, haystack, [start])*, *concat_ws(sep, X)*, *concat(X)*, *decode(value, algorithm)*, *encode(value, algorithm)*, *endswith(str, suffix)*, *extract(str)*, *fuzzy_match(pattern, str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *gunzip(b)*, *gzip(value)*, *humanize_duration(secs)*, *humanize_file_size(value)*, *humanize_id(id)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *logfmt2json(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *parse_url(url)*, *pretty_print(str)*, *printf(format, X)*, *proper(str)*, *regexp_capture_into_json(string, pattern, [options])*, *regexp_capture(string, pattern)*, *regexp_replace(str, re, repl)*, *regexp_replace(str, re, repl)*, *replace(str, old, replacement)*, *replicate(str, N)*, *reverse(str)*, *rightstr(str, N)*, *rtrim(str, [chars])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *timezone(tz, ts)*, *trim(str, [chars])*, *unhex(X)*, *unicode(X)*, *unparse_url(obj)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.9.150 regexp_replace(*str*, *re*, *repl*)

Replace the parts of a string that match a regular expression.

PRQL Name: text.regexp_replace

Parameters

- **str*** — The string to perform replacements on
- **re*** — The regular expression to match
- **repl*** — The replacement string. You can reference capture groups with a backslash followed by the number of the group, starting with 1.

Examples

To replace the word at the start of the string 'Hello, World!' with 'Goodbye':

```
;SELECT regexp_replace('Hello, World!', '^(\w+)', 'Goodbye')
Goodbye, World!
```

To wrap alphanumeric words with angle brackets:

```
;SELECT regexp_replace('123 abc', '(\w+)', '<\1>')
<123> <abc>
```

See Also

anonymize(value), *char(X)*, *charindex(needle, haystack, [start])*, *concat_ws(sep, X)*, *concat(X)*, *decode(value, algorithm)*, *encode(value, algorithm)*, *endswith(str, suffix)*, *extract(str)*, *fuzzy_match(pattern, str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *gunzip(b)*, *gzip(value)*, *humanize_duration(secs)*, *humanize_file_size(value)*, *humanize_id(id)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *logfmt2json(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *parse_url(url)*, *pretty_print(str)*, *printf(format, X)*, *proper(str)*, *regexp_capture_into_json(string, pattern, [options])*, *regexp_capture(string, pattern)*, *regexp_match(re, str)*, *regexp_match(re, str)*, *replace(str, old, replacement)*, *replicate(str, N)*, *reverse(str)*, *rightstr(str, N)*, *rtrim(str, [chars])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *timezone(tz, ts)*, *trim(str, [chars])*, *unhex(X)*, *unicode(X)*, *unparse_url(obj)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.9.151 replace(*str*, *old*, *replacement*)

Returns a string formed by substituting the replacement string for every occurrence of the old string in the given string.

Parameters

- **str*** — The string to perform substitutions on.
- **old*** — The string to be replaced.
- **replacement*** — The string to replace any occurrences of the old string with.

Examples

To replace the string 'x' with 'z' in 'abc':

```
;SELECT replace('abc', 'x', 'z')
abc
```

To replace the string 'a' with 'z' in 'abc':

```
;SELECT replace('abc', 'a', 'z')
zbc
```

See Also

anonymize(value), *char(X)*, *charindex(needle, haystack, [start])*, *concat_ws(sep, X)*, *concat(X)*, *decode(value, algorithm)*, *encode(value, algorithm)*, *endswith(str, suffix)*, *extract(str)*, *fuzzy_match(pattern, str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *gunzip(b)*, *gzip(value)*, *humanize_duration(secs)*, *humanize_file_size(value)*, *humanize_id(id)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *logfmt2json(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *parse_url(url)*, *pretty_print(str)*, *printf(format, X)*, *proper(str)*, *regexp_capture_into_json(string, pattern, [options])*, *regexp_capture(string, pattern)*, *regexp_match(re, str)*, *regexp_replace(str, re, repl)*, *replicate(str, N)*, *reverse(str)*, *rightstr(str, N)*, *rtrim(str, [chars])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *timezone(tz, ts)*, *trim(str, [chars])*, *unhex(X)*, *unicode(X)*, *unparse_url(obj)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.9.152 replicate(*str*, *N*)

Returns the given string concatenated N times.

Parameters

- **str*** — The string to replicate.
- **N*** — The number of times to replicate the string.

Examples

To repeat the string 'abc' three times:

```
;SELECT replicate('abc', 3)
abcabcabc
```

See Also

anonymize(value), *char(X)*, *charindex(needle, haystack, [start])*, *concat_ws(sep, X)*, *concat(X)*, *decode(value, algorithm)*, *encode(value, algorithm)*, *endswith(str, suffix)*, *extract(str)*, *fuzzy_match(pattern, str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *gunzip(b)*, *gzip(value)*, *humanize_duration(secs)*, *humanize_file_size(value)*, *humanize_id(id)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *logfmt2json(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *parse_url(url)*, *pretty_print(str)*, *printf(format, X)*, *proper(str)*, *regexp_capture_into_json(string, pattern, [options])*, *regexp_capture(string, pattern)*, *regexp_match(re, str)*, *regexp_replace(str, re, repl)*, *replace(str, old, replacement)*, *reverse(str)*, *rightstr(str, N)*, *rtrim(str, [chars])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *timezone(tz, ts)*, *trim(str, [chars])*, *unhex(X)*, *unicode(X)*, *unparse_url(obj)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.9.153 reverse(*str*)

Returns the reverse of the given string.

PRQL Name: text.reverse

Parameters

- **str*** — The string to reverse.

Examples

To reverse the string 'abc':

```
;SELECT reverse('abc')
cba
```

See Also

anonymize(value), *char(X)*, *charindex(needle, haystack, [start])*, *concat_ws(sep, X)*, *concat(X)*, *decode(value, algorithm)*, *encode(value, algorithm)*, *endswith(str, suffix)*, *extract(str)*, *fuzzy_match(pattern, str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *gunzip(b)*, *gzip(value)*, *humanize_duration(secs)*, *humanize_file_size(value)*, *humanize_id(id)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *logfmt2json(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *parse_url(url)*, *pretty_print(str)*, *printf(format, X)*, *proper(str)*, *regexp_capture_into_json(string, pattern, [options])*, *regexp_capture(string, pattern)*, *regexp_match(re, str)*, *regexp_replace(str, re, repl)*, *replace(str, old, replacement)*, *replicate(str, N)*, *rightstr(str, N)*, *rtrim(str, [chars])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *timezone(tz, ts)*, *trim(str, [chars])*, *unhex(X)*, *unicode(X)*, *unparse_url(obj)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.9.154 rightstr(*str*, *N*)

Returns the N rightmost (UTF-8) characters in the given string.

Parameters

- **str*** — The string to return subset.
- **N*** — The number of characters from the right side of the string to return.

Examples

To get the last character of the string 'abc':

```
;SELECT rightstr('abc', 1)
c
```

To get the last ten characters of a string, regardless of size:

```
;SELECT rightstr('abc', 10)
abc
```

See Also

anonymize(value), *char(X)*, *charindex(needle, haystack, [start])*, *concat_ws(sep, X)*, *concat(X)*, *decode(value, algorithm)*, *encode(value, algorithm)*, *endswith(str, suffix)*, *extract(str)*, *fuzzy_match(pattern, str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *gunzip(b)*, *gzip(value)*, *humanize_duration(secs)*, *humanize_file_size(value)*, *humanize_id(id)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *logfmt2json(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *parse_url(url)*, *pretty_print(str)*, *printf(format, X)*, *proper(str)*, *regexp_capture_into_json(string, pattern, [options])*, *regexp_capture(string, pattern)*, *regexp_match(re, str)*, *regexp_replace(str, re, repl)*, *replace(str, old, replacement)*, *replicate(str, N)*, *reverse(str)*, *rtrim(str, [chars])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *timezone(tz, ts)*, *trim(str, [chars])*, *unhex(X)*, *unicode(X)*, *unparse_url(obj)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.9.155 round(*num*, *[digits]*)

Returns a floating-point value rounded to the given number of digits to the right of the decimal point.

Parameters

- **num*** — The value to round.
- **digits** — The number of digits to the right of the decimal to round to.

Examples

To round the number 123.456 to an integer:

```
;SELECT round(123.456)
123
```

To round the number 123.456 to a precision of 1:

```
;SELECT round(123.456, 1)
123.5
```

To round the number 123.456 to a precision of 5:

```
;SELECT round(123.456, 5)
123.456
```

See Also

abs(x), *acos(num)*, *acosh(num)*, *asin(num)*, *asinh(num)*, *atan2(y, x)*, *atan(num)*, *atanh(num)*, *atn2(y, x)*, *avg(X)*, *ceil(num)*, *degrees(radians)*, *exp(x)*, *floor(num)*, *log10(x)*, *log(x)*, *max(X)*, *min(X)*, *pi()*, *power(base, exp)*, *radians(degrees)*, *sign(num)*, *square(num)*, *sum(X)*, *total(X)*

12.9.156 row_number()

Returns the number of the row within the current partition, starting from 1.

Examples

To number messages from a process:

```
;SELECT row_number() OVER (PARTITION BY ex_procname ORDER BY log_line) AS␣
↪msg_num, ex_procname, log_body FROM lnav_example_log
 msg_num   ex_procname    log_body
         1 gw           Goodbye, World!
         2 gw           Goodbye, World!
         3 gw           Goodbye, World!
         1 hw           Hello, World!
```

See Also

cume_dist(), *dense_rank()*, *first_value(expr)*, *lag(expr, [offset], [default])*, *last_value(expr)*, *lead(expr, [offset], [default])*, *nth_value(expr, N)*, *ntile(groups)*, *percent_rank()*, *rank()*

12.9.157 rtrim(*str*, *[chars]*)

Returns a string formed by removing any and all characters that appear in the second argument from the right side of the first.

Parameters

- **str*** — The string to trim characters from the right side
- **chars** — The characters to trim. Defaults to spaces.

Examples

To trim the space characters from the end of the string 'abc ':

```
;SELECT rtrim('abc   ')
abc
```

To trim the characters 'b' and 'c' from the string 'abbbbcccc':

```
;SELECT rtrim('abbbbcccc', 'bc')
a
```

See Also

anonymize(value), *char(X)*, *charindex(needle, haystack, [start])*, *concat_ws(sep, X)*, *concat(X)*, *decode(value, algorithm)*, *encode(value, algorithm)*, *endswith(str, suffix)*, *extract(str)*, *fuzzy_match(pattern, str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *gunzip(b)*, *gzip(value)*, *humanize_duration(secs)*, *humanize_file_size(value)*, *humanize_id(id)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *logfmt2json(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *parse_url(url)*, *pretty_print(str)*, *printf(format, X)*, *proper(str)*, *regexp_capture_into_json(string, pattern, [options])*, *regexp_capture(string, pattern)*, *regexp_match(re, str)*, *regexp_replace(str, re, repl)*, *replace(str, old, replacement)*, *replicate(str, N)*, *reverse(str)*, *rightstr(str, N)*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *timezone(tz, ts)*, *trim(str, [chars])*, *unhex(X)*, *unicode(X)*, *unparse_url(obj)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.9.158 shell_exec(*cmd*, *[input]*, *[options]*)

Executes a shell command and returns its output.

PRQL Name: shell.exec

Parameters

- **cmd*** — The command to execute.
- **input** — A blob of data to write to the command's standard input.
- **options** — A JSON object containing options for the execution with the following properties:
  - **env** — An object containing the environment variables to set or, if NULL, to unset.

See Also

12.9.159 sign(*num*)

Returns the sign of the given number as -1, 0, or 1

Parameters

- **num*** — The number

Examples

To get the sign of 10:

```
;SELECT sign(10)
1
```

To get the sign of 0:

```
;SELECT sign(0)
0
```

To get the sign of -10:

```
;SELECT sign(-10)
-1
```

See Also

abs(x), *acos(num)*, *acosh(num)*, *asin(num)*, *asinh(num)*, *atan2(y, x)*, *atan(num)*, *atanh(num)*, *atn2(y, x)*, *avg(X)*, *ceil(num)*, *degrees(radians)*, *exp(x)*, *floor(num)*, *log10(x)*, *log(x)*, *max(X)*, *min(X)*, *pi()*, *power(base, exp)*, *radians(degrees)*, *round(num, [digits])*, *square(num)*, *sum(X)*, *total(X)*

12.9.160 source_log_stmt(*path*)

A table-valued function for getting information about log statements that were found in source code added by the :add-source-path command.

Parameters

- **path*** — The source file path

12.9.161 sparkline(*value*, *[upper]*)

Function used to generate a sparkline bar chart. The non-aggregate version converts a single numeric value on a range to a bar chart character. The aggregate version returns a string with a bar character for every numeric input

PRQL Name: text.sparkline

Parameters

- **value*** — The numeric value to convert
- **upper** — The upper bound of the numeric range. The non-aggregate version defaults to 100. The aggregate version uses the largest value in the inputs.

Examples

To get the unicode block element for the value 32 in the range of 0-128:

```
;SELECT sparkline(32, 128)
```

To chart the values in a JSON array:

```
;SELECT sparkline(value) FROM json_each('[0, 1, 2, 3, 4, 5, 6, 7, 8]')
```

See Also

anonymize(value), *char(X)*, *charindex(needle, haystack, [start])*, *concat_ws(sep, X)*, *concat(X)*, *decode(value, algorithm)*, *encode(value, algorithm)*, *endswith(str, suffix)*, *extract(str)*, *fuzzy_match(pattern, str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *gunzip(b)*, *gzip(value)*, *humanize_duration(secs)*, *humanize_file_size(value)*, *humanize_id(id)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *logfmt2json(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *parse_url(url)*, *pretty_print(str)*, *printf(format, X)*, *proper(str)*, *regexp_capture_into_json(string, pattern, [options])*, *regexp_capture(string, pattern)*, *regexp_match(re, str)*, *regexp_replace(str, re, repl)*, *replace(str, old, replacement)*, *replicate(str, N)*, *reverse(str)*, *rightstr(str, N)*, *rtrim(str, [chars])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *timezone(tz, ts)*, *trim(str, [chars])*, *unhex(X)*, *unicode(X)*, *unparse_url(obj)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.9.162 spooky_hash(*str*)

Compute the hash value for the given arguments.

Parameters

- **str** — The string to hash

Examples

To produce a hash for the string 'Hello, World!':

```
;SELECT spooky_hash('Hello, World!')
0b1d52cc5427db4c6a9eed9d3e5700f4
```

To produce a hash for the parameters where one is NULL:

```
;SELECT spooky_hash('Hello, World!', NULL)
c96ee75d48e6ea444fee8af948f6da25
```

To produce a hash for the parameters where one is an empty string:

```
;SELECT spooky_hash('Hello, World!', '')
c96ee75d48e6ea444fee8af948f6da25
```

To produce a hash for the parameters where one is a number:

```
;SELECT spooky_hash('Hello, World!', 123)
f96b3d9c1a19f4394c97a1b79b1880df
```

See Also

anonymize(value), *char(X)*, *charindex(needle, haystack, [start])*, *concat_ws(sep, X)*, *concat(X)*, *decode(value, algorithm)*, *encode(value, algorithm)*, *endswith(str, suffix)*, *extract(str)*, *fuzzy_match(pattern, str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *gunzip(b)*, *gzip(value)*, *humanize_duration(secs)*, *humanize_file_size(value)*, *humanize_id(id)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *logfmt2json(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *parse_url(url)*, *pretty_print(str)*, *printf(format, X)*, *proper(str)*, *regexp_capture_into_json(string, pattern, [options])*, *regexp_capture(string, pattern)*, *regexp_match(re, str)*, *regexp_replace(str, re, repl)*, *replace(str, old, replacement)*, *replicate(str, N)*, *reverse(str)*, *rightstr(str, N)*, *rtrim(str, [chars])*, *sparkline(value, [upper])*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *timezone(tz, ts)*, *trim(str, [chars])*, *unhex(X)*, *unicode(X)*, *unparse_url(obj)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.9.163 sqlite_compileoption_get(*N*)

Returns the N-th compile-time option used to build SQLite or NULL if N is out of range.

Parameters

- **N*** — The option number to get

12.9.164 sqlite_compileoption_used(*option*)

Returns true (1) or false (0) depending on whether or not that compile-time option was used during the build.

Parameters

- **option*** — The name of the compile-time option.

Examples

To check if the SQLite library was compiled with ENABLE_FTS3:

```
;SELECT sqlite_compileoption_used('ENABLE_FTS3')
1
```

12.9.165 sqlite_source_id()

Returns a string that identifies the specific version of the source code that was used to build the SQLite library.

12.9.166 sqlite_version()

Returns the version string for the SQLite library that is running.

12.9.167 square(*num*)

Returns the square of the argument

Parameters

- **num*** — The number to square

Examples

To get the square of two:

```
;SELECT square(2)
4
```

See Also

abs(x), *acos(num)*, *acosh(num)*, *asin(num)*, *asinh(num)*, *atan2(y, x)*, *atan(num)*, *atanh(num)*, *atn2(y, x)*, *avg(X)*, *ceil(num)*, *degrees(radians)*, *exp(x)*, *floor(num)*, *log10(x)*, *log(x)*, *max(X)*, *min(X)*, *pi()*, *power(base, exp)*, *radians(degrees)*, *round(num, [digits])*, *sign(num)*, *sum(X)*, *total(X)*

12.9.168 startswith(*str*, *prefix*)

Test if a string begins with the given prefix

Parameters

- **str*** — The string to test
- **prefix*** — The prefix to check in the string

Examples

To test if the string 'foobar' starts with 'foo':

```
;SELECT startswith('foobar', 'foo')
1
```

To test if the string 'foobar' starts with 'bar':

```
;SELECT startswith('foobar', 'bar')
0
```

See Also

anonymize(value), *char(X)*, *charindex(needle, haystack, [start])*, *concat_ws(sep, X)*, *concat(X)*, *decode(value, algorithm)*, *encode(value, algorithm)*, *endswith(str, suffix)*, *extract(str)*, *fuzzy_match(pattern, str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *gunzip(b)*, *gzip(value)*, *humanize_duration(secs)*, *humanize_file_size(value)*, *humanize_id(id)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *logfmt2json(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *parse_url(url)*, *pretty_print(str)*, *printf(format, X)*, *proper(str)*, *regexp_capture_into_json(string, pattern, [options])*, *regexp_capture(string, pattern)*, *regexp_match(re, str)*, *regexp_replace(str, re, repl)*, *replace(str, old, replacement)*, *replicate(str, N)*, *reverse(str)*, *rightstr(str, N)*, *rtrim(str, [chars])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *strfilter(source, include)*, *substr(str, start, [size])*, *timezone(tz, ts)*, *trim(str, [chars])*, *unhex(X)*, *unicode(X)*, *unparse_url(obj)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.9.169 strfilter(*source*, *include*)

Returns the source string with only the characters given in the second parameter

Parameters

- **source*** — The string to filter
- **include*** — The characters to include in the result

Examples

To get the 'b', 'c', and 'd' characters from the string 'abcabc':

```
;SELECT strfilter('abcabc', 'bcd')
bcbc
```

See Also

anonymize(value), *char(X)*, *charindex(needle, haystack, [start])*, *concat_ws(sep, X)*, *concat(X)*, *decode(value, algorithm)*, *encode(value, algorithm)*, *endswith(str, suffix)*, *extract(str)*, *fuzzy_match(pattern, str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *gunzip(b)*, *gzip(value)*, *humanize_duration(secs)*, *humanize_file_size(value)*, *humanize_id(id)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *logfmt2json(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *parse_url(url)*, *pretty_print(str)*, *printf(format, X)*, *proper(str)*,

regexp_capture_into_json(string, pattern, [options]), *regexp_capture(string, pattern)*, *regexp_match(re, str)*, *regexp_replace(str, re, repl)*, *replace(str, old, replacement)*, *replicate(str, N)*, *reverse(str)*, *rightstr(str, N)*, *rtrim(str, [chars])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *substr(str, start, [size])*, *timezone(tz, ts)*, *trim(str, [chars])*, *unhex(X)*, *unicode(X)*, *unparse_url(obj)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.9.170 strftime(*format*, *timestring*, *modifier*)

Returns the date formatted according to the format string specified as the first argument.

Parameters

- **format*** — A format string with substitutions similar to those found in the strftime() standard C library.
- **timestring*** — The string to convert to a date with time.
- **modifier** — A transformation that is applied to the value to the left.

Examples

To get the year from the timestamp '2017-01-02T03:04:05':

```
;SELECT strftime('%Y', '2017-01-02T03:04:05')
2017
```

To create a string with the time from the timestamp '2017-01-02T03:04:05' plus one minute:

```
;SELECT strftime('The time is: %H:%M:%S', '2017-01-02T03:04:05', '+1 minute
↪')
The time is: 03:05:05
```

To create a string with the Julian day from the epoch timestamp 1491341842:

```
;SELECT strftime('Julian day: %J', 1491341842, 'unixepoch')
Julian day: 2457848.400949074
```

See Also

date(timestring, modifier), *datetime(timestring, modifier)*, *humanize_duration(secs)*, *julianday(timestring, modifier)*, *time(timestring, modifier)*, *timediff(time1, time2)*, *timeslice(time, slice)*, *timezone(tz, ts)*

12.9.171 substr(*str*, *start*, *[size]*)

Returns a substring of input string X that begins with the Y-th character and which is Z characters long.

Parameters

- **str*** — The string to extract a substring from.
- **start*** — The index within 'str' that is the start of the substring. Indexes begin at 1. A negative value means that the substring is found by counting from the right rather than the left.
- **size** — The size of the substring. If not given, then all characters through the end of the string are returned. If the value is negative, then the characters before the start are returned.

Examples

To get the substring starting at the second character until the end of the string 'abc':

```
;SELECT substr('abc', 2)
bc
```

To get the substring of size one starting at the second character of the string 'abc':

```
;SELECT substr('abc', 2, 1)
b
```

To get the substring starting at the last character until the end of the string 'abc':

```
;SELECT substr('abc', -1)
c
```

To get the substring starting at the last character and going backwards one step of the string 'abc':

```
;SELECT substr('abc', -1, -1)
b
```

See Also

anonymize(value), *char(X)*, *charindex(needle, haystack, [start])*, *concat_ws(sep, X)*, *concat(X)*, *decode(value, algorithm)*, *encode(value, algorithm)*, *endswith(str, suffix)*, *extract(str)*, *fuzzy_match(pattern, str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *gunzip(b)*, *gzip(value)*, *humanize_duration(secs)*, *humanize_file_size(value)*, *humanize_id(id)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *logfmt2json(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *parse_url(url)*, *pretty_print(str)*, *printf(format, X)*, *proper(str)*, *regexp_capture_into_json(string, pattern, [options])*, *regexp_capture(string, pattern)*, *regexp_match(re, str)*, *regexp_replace(str, re, repl)*, *replace(str, old, replacement)*, *replicate(str, N)*, *reverse(str)*, *rightstr(str, N)*, *rtrim(str, [chars])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *timezone(tz, ts)*, *trim(str, [chars])*, *unhex(X)*, *unicode(X)*, *unparse_url(obj)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.9.172 sum(*X*)

Returns the sum of the values in the group as an integer.

Parameters

- **X*** — The values to add.

Examples

To sum all of the values in the column 'ex_duration' from the table 'lnav_example_log':

```
;SELECT sum(ex_duration) FROM lnav_example_log
17
```

See Also

abs(x), *acos(num)*, *acosh(num)*, *asin(num)*, *asinh(num)*, *atan2(y, x)*, *atan(num)*, *atanh(num)*, *atn2(y, x)*, *avg(X)*, *ceil(num)*, *degrees(radians)*, *exp(x)*, *floor(num)*, *log10(x)*, *log(x)*, *max(X)*, *min(X)*, *pi()*, *power(base, exp)*, *radians(degrees)*, *round(num, [digits])*, *sign(num)*, *square(num)*, *total(X)*

12.9.173 time(*timestring*, *modifier*)

Returns the time in this format: HH:MM:SS.

Parameters

- **timestring*** — The string to convert to a time.
- **modifier** — A transformation that is applied to the value to the left.

Examples

To get the time portion of the timestamp '2017-01-02T03:04:05':

```
;SELECT time('2017-01-02T03:04:05')
03:04:05
```

To get the time portion of the timestamp '2017-01-02T03:04:05' plus one minute:

```
;SELECT time('2017-01-02T03:04:05', '+1 minute')
03:05:05
```

To get the time portion of the epoch timestamp 1491341842:

```
;SELECT time(1491341842, 'unixepoch')
21:37:22
```

See Also

date(timestring, modifier), *datetime(timestring, modifier)*, *humanize_duration(secs)*, *julianday(timestring, modifier)*, *strftime(format, timestring, modifier)*, *timediff(time1, time2)*, *timeslice(time, slice)*, *timezone(tz, ts)*

12.9.174 timediff(*time1*, *time2*)

Compute the difference between two timestamps in seconds

PRQL Name: time.diff

Parameters

- **time1*** — The first timestamp
- **time2*** — The timestamp to subtract from the first

Examples

To get the difference between two timestamps:

```
;SELECT timediff('2017-02-03T04:05:06', '2017-02-03T04:05:00')
6
```

To get the difference between relative timestamps:

```
;SELECT timediff('today', 'yesterday')
86400
```

See Also

date(timestring, modifier), *datetime(timestring, modifier)*, *humanize_duration(secs)*, *julianday(timestring, modifier)*, *strftime(format, timestring, modifier)*, *time(timestring, modifier)*, *timeslice(time, slice)*, *timezone(tz, ts)*

12.9.175 timeslice(*time*, *slice*)

Return the start of the slice of time that the given timestamp falls in. If the time falls outside of the slice, NULL is returned.

PRQL Name: time.slice

Parameters

- **time*** — The timestamp to get the time slice for.
- **slice*** — The size of the time slices

Examples

To get the timestamp rounded down to the start of the ten minute slice:

```
;SELECT timeslice('2017-01-01T05:05:00', '10m')
2017-01-01 05:00:00.000000
```

To group log messages into five minute buckets and count them:

```
;SELECT timeslice(log_time_msecs, '5m') AS slice, count(1)
```

FROM lnav_example_log GROUP BY slice

slice count(1)

2017-02.000000 2 2017-02.000000 1 2017-02.000000 1

To group log messages by those before 4:30am and after:

```
;SELECT timeslice(log_time_msecs, 'before 4:30am') AS slice, count(1) FROM␣
↪lnav_example_log GROUP BY slice
    slice       count(1)
        <NULL>          1
2017-02.000000          3
```

See Also

date(timestring, modifier), *datetime(timestring, modifier)*, *humanize_duration(secs)*, *julianday(timestring, modifier)*, *strftime(format, timestring, modifier)*, *time(timestring, modifier)*, *timediff(time1, time2)*, *timezone(tz, ts)*

12.9.176 timezone(*tz*, *ts*)

Convert a timestamp to the given timezone

PRQL Name: time.to_zone

Parameters

- **tz*** — The target timezone
- **ts*** — The source timestamp

Examples

To convert a time to America/Los_Angeles:

```
;SELECT timezone('America/Los_Angeles', '2022-03-02T10:00')
2022-03-02T02:00:00.000000-0800
```

See Also

anonymize(value), *char(X)*, *charindex(needle, haystack, [start])*, *concat_ws(sep, X)*, *concat(X)*, *date(timestring, modifier)*, *datetime(timestring, modifier)*, *decode(value, algorithm)*, *encode(value, algorithm)*, *endswith(str, suffix)*, *extract(str)*, *fuzzy_match(pattern, str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *gunzip(b)*, *gzip(value)*, *humanize_duration(secs)*, *humanize_duration(secs)*, *humanize_file_size(value)*, *humanize_id(id)*, *instr(haystack, needle)*, *julianday(timestring, modifier)*, *leftstr(str, N)*, *length(str)*, *logfmt2json(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *parse_url(url)*, *pretty_print(str)*, *printf(format, X)*, *proper(str)*, *regexp_capture_into_json(string, pattern, [options])*, *regexp_capture(string, pattern)*, *regexp_match(re, str)*, *regexp_replace(str, re, repl)*, *replace(str, old, replacement)*, *replicate(str, N)*, *reverse(str)*, *rightstr(str, N)*, *rtrim(str, [chars])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *strftime(format, timestring, modifier)*, *substr(str, start, [size])*, *time(timestring, modifier)*, *timediff(time1, time2)*, *timeslice(time, slice)*, *trim(str, [chars])*, *unhex(X)*, *unicode(X)*, *unparse_url(obj)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.9.177 total(*X*)

Returns the sum of the values in the group as a floating-point.

Parameters

- **X*** — The values to add.

Examples

To total all of the values in the column 'ex_duration' from the table 'lnav_example_log':

```
;SELECT total(ex_duration) FROM lnav_example_log
17
```

See Also

abs(x), *acos(num)*, *acosh(num)*, *asin(num)*, *asinh(num)*, *atan2(y, x)*, *atan(num)*, *atanh(num)*, *atn2(y, x)*, *avg(X)*, *ceil(num)*, *degrees(radians)*, *exp(x)*, *floor(num)*, *log10(x)*, *log(x)*, *max(X)*, *min(X)*, *pi()*, *power(base, exp)*, *radians(degrees)*, *round(num, [digits])*, *sign(num)*, *square(num)*, *sum(X)*

12.9.178 total_changes()

Returns the number of row changes caused by INSERT, UPDATE or DELETE statements since the current database connection was opened.

12.9.179 trim(*str*, *[chars]*)

Returns a string formed by removing any and all characters that appear in the second argument from the left and right sides of the first.

Parameters

- **str*** — The string to trim characters from the left and right sides.
- **chars** — The characters to trim. Defaults to spaces.

Examples

To trim spaces from the start and end of the string ' abc ':

```
;SELECT trim('    abc   ')
abc
```

To trim the characters '-' and '+' from the string '-+abc+-':

```
;SELECT trim('-+abc+-', '-+')
abc
```

See Also

anonymize(value), *char(X)*, *charindex(needle, haystack, [start])*, *concat_ws(sep, X)*, *concat(X)*, *decode(value, algorithm)*, *encode(value, algorithm)*, *endswith(str, suffix)*, *extract(str)*, *fuzzy_match(pattern, str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *gunzip(b)*, *gzip(value)*, *humanize_duration(secs)*, *humanize_file_size(value)*, *humanize_id(id)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *logfmt2json(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *parse_url(url)*, *pretty_print(str)*, *printf(format, X)*, *proper(str)*, *regexp_capture_into_json(string, pattern, [options])*, *regexp_capture(string, pattern)*, *regexp_match(re, str)*, *regexp_replace(str, re, repl)*, *replace(str, old, replacement)*, *replicate(str, N)*, *reverse(str)*, *rightstr(str, N)*, *rtrim(str, [chars])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *timezone(tz, ts)*, *unhex(X)*, *unicode(X)*, *unparse_url(obj)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.9.180 typeof(*X*)

Returns a string that indicates the datatype of the expression X: "null", "integer", "real", "text", or "blob".

Parameters

- **X*** — The expression to check.

Examples

To get the type of the number 1:

```
;SELECT typeof(1)
integer
```

To get the type of the string 'abc':

```
;SELECT typeof('abc')
text
```

12.9.181 unhex(*X*)

Returns a blob value that is a decoding of the given hex string

Parameters

- **X*** — The hex string to decode

Examples

To decode the string 'Hello' encoded in hex:

```
;SELECT unhex('48656c6c6f')
Hello
```

See Also

anonymize(value), *char(X)*, *charindex(needle, haystack, [start])*, *concat_ws(sep, X)*, *concat(X)*, *decode(value, algorithm)*, *encode(value, algorithm)*, *endswith(str, suffix)*, *extract(str)*, *fuzzy_match(pattern, str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *gunzip(b)*, *gzip(value)*, *humanize_duration(secs)*, *humanize_file_size(value)*, *humanize_id(id)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *logfmt2json(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *parse_url(url)*, *pretty_print(str)*, *printf(format, X)*, *proper(str)*, *regexp_capture_into_json(string, pattern, [options])*, *regexp_capture(string, pattern)*, *regexp_match(re, str)*, *regexp_replace(str, re, repl)*, *replace(str, old, replacement)*, *replicate(str, N)*, *reverse(str)*, *rightstr(str, N)*, *rtrim(str, [chars])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *timezone(tz, ts)*, *trim(str, [chars])*, *unicode(X)*, *unparse_url(obj)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.9.182 unicode(*X*)

Returns the numeric unicode code point corresponding to the first character of the string X.

Parameters

- **X*** — The string to examine.

Examples

To get the unicode code point for the first character of 'abc':

```
;SELECT unicode('abc')
97
```

See Also

anonymize(value), *char(X)*, *charindex(needle, haystack, [start])*, *concat_ws(sep, X)*, *concat(X)*, *decode(value, algorithm)*, *encode(value, algorithm)*, *endswith(str, suffix)*, *extract(str)*, *fuzzy_match(pattern, str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *gunzip(b)*, *gzip(value)*, *humanize_duration(secs)*, *humanize_file_size(value)*, *humanize_id(id)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *logfmt2json(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *parse_url(url)*, *pretty_print(str)*, *printf(format, X)*, *proper(str)*, *regexp_capture_into_json(string, pattern, [options])*, *regexp_capture(string, pattern)*, *regexp_match(re, str)*, *regexp_replace(str, re, repl)*, *replace(str, old, replacement)*, *replicate(str, N)*, *reverse(str)*, *rightstr(str, N)*, *rtrim(str, [chars])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *timezone(tz, ts)*, *trim(str, [chars])*, *unhex(X)*, *unparse_url(obj)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.9.183 unlikely(*value*)

Short-hand for likelihood(X, 0.0625)

Parameters

- **value*** — The boolean value to return

12.9.184 unparse_url(*obj*)

Convert a JSON object containing the parts of a URL into a URL string

Parameters

- **obj*** — The JSON object containing the URL parts

Examples

To unparse the object '{"scheme": "https", "host": "example.com"}':

```
;SELECT unparse_url('{"scheme": "https", "host": "example.com"}')
https://example.com/
```

See Also

anonymize(value), *char(X)*, *charindex(needle, haystack, [start])*, *concat_ws(sep, X)*, *concat(X)*, *decode(value, algorithm)*, *encode(value, algorithm)*, *endswith(str, suffix)*, *extract(str)*, *fuzzy_match(pattern, str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *gunzip(b)*, *gzip(value)*, *humanize_duration(secs)*, *humanize_file_size(value)*, *humanize_id(id)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *logfmt2json(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *parse_url(url)*, *parse_url(url)*, *pretty_print(str)*, *printf(format, X)*, *proper(str)*, *regexp_capture_into_json(string, pattern, [options])*, *regexp_capture(string, pattern)*, *regexp_match(re, str)*, *regexp_replace(str, re, repl)*, *replace(str, old, replacement)*, *replicate(str, N)*, *reverse(str)*, *rightstr(str, N)*, *rtrim(str, [chars])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *timezone(tz, ts)*, *trim(str, [chars])*, *unhex(X)*, *unicode(X)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.9.185 upper(*str*)

Returns a copy of the given string with all ASCII characters converted to upper case.

Parameters

- **str*** — The string to convert.

Examples

To uppercase the string 'aBc':

```
;SELECT upper('aBc')
ABC
```

See Also

anonymize(value), *char(X)*, *charindex(needle, haystack, [start])*, *concat_ws(sep, X)*, *concat(X)*, *decode(value, algorithm)*, *encode(value, algorithm)*, *endswith(str, suffix)*, *extract(str)*, *fuzzy_match(pattern, str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *gunzip(b)*, *gzip(value)*, *humanize_duration(secs)*, *humanize_file_size(value)*, *humanize_id(id)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *logfmt2json(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *parse_url(url)*, *pretty_print(str)*, *printf(format, X)*, *proper(str)*, *regexp_capture_into_json(string, pattern, [options])*, *regexp_capture(string, pattern)*, *regexp_match(re, str)*, *regexp_replace(str, re, repl)*, *replace(str, old, replacement)*, *replicate(str, N)*, *reverse(str)*, *rightstr(str, N)*, *rtrim(str, [chars])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *timezone(tz, ts)*, *trim(str, [chars])*, *unhex(X)*, *unicode(X)*, *unparse_url(obj)*, *xpath(xpath, xmldoc)*

12.9.186 xpath(*xpath*, *xmldoc*)

A table-valued function that executes an xpath expression over an XML string and returns the selected values.

Parameters

- **xpath*** — The XPATH expression to evaluate over the XML document.
- **xmldoc*** — The XML document as a string.

Examples

To select the XML nodes on the path '/abc/def':

```
;SELECT * FROM xpath('/abc/def', '<abc><def a="b">Hello</def><def>Bye</def>
↪</abc>')
    result       node_path  node_attr node_text
<def a=</def> /abc/def[1] {"a":"b"} Hello
<def>Bye</def> /abc/def[2] {}         Bye
```

To select all 'a' attributes on the path '/abc/def':

```
;SELECT * FROM xpath('/abc/def/@a', '<abc><def a="b">Hello</def><def>Bye</
↪def></abc>')
result   node_path    node_attr node_text
b      /abc/def[1]/@a {"a":"b"} Hello
```

To select the text nodes on the path '/abc/def':

```
;SELECT * FROM xpath('/abc/def/text()', '<abc><def a="b">Hello &#x2605;</
↪def></abc>')
result     node_path    node_attr node_text
Hello | /abc/def/text() {}         Hello |
```

See Also

anonymize(value), *char(X)*, *charindex(needle, haystack, [start])*, *concat_ws(sep, X)*, *concat(X)*, *decode(value, algorithm)*, *encode(value, algorithm)*, *endswith(str, suffix)*, *extract(str)*, *fuzzy_match(pattern, str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *gunzip(b)*, *gzip(value)*, *humanize_duration(secs)*, *humanize_file_size(value)*, *humanize_id(id)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *logfmt2json(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *parse_url(url)*, *pretty_print(str)*, *printf(format, X)*, *proper(str)*, *regexp_capture_into_json(string, pattern, [options])*, *regexp_capture(string, pattern)*, *regexp_match(re, str)*, *regexp_replace(str, re, repl)*, *replace(str, old, replacement)*, *replicate(str, N)*, *reverse(str)*, *rightstr(str, N)*, *rtrim(str, [chars])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *timezone(tz, ts)*, *trim(str, [chars])*, *unhex(X)*, *unicode(X)*, *unparse_url(obj)*, *upper(str)*

12.9.187 yaml_to_json(*yaml*)

Convert a YAML document to a JSON-encoded string

PRQL Name: yaml.to_json

Parameters

- **yaml*** — The YAML value to convert to JSON.

Examples

To convert the document "abc: def":

```
;SELECT yaml_to_json('abc: def')
{"abc": "def"}
```

See Also

jget(json, ptr, [default]), *json_array_length(X, [P])*, *json_array(X)*, *json_concat(json, value)*, *json_contains(json, value)*, *json_each(X, [P])*, *json_extract(X, P)*, *json_group_array(value)*, *json_group_object(name, value)*, *json_insert(X, P, Y)*, *json_object(N, V)*, *json_quote(X)*, *json_remove(X, P)*, *json_replace(X, P, Y)*, *json_set(X, P, Y)*, *json_tree(X, [P])*, *json_type(X, [P])*, *json_valid(X)*, *json(X)*

12.9.188 zeroblob(*N*)

Returns a BLOB consisting of N bytes of 0x00.

Parameters

- **N*** — The size of the BLOB.

12.9.189 ;.dump *path table*

Dump one or more tables to a file as a series of SQL statements

Parameters

- **path*** — The path to the file to write
- **table** — The name of the table to dump

See Also

:append-to path, *;.read path*, *;.save path*, *:echo [-n] msg*, *echoln(value)*, *:export-session-to path*, *:open [–since] [–until] path*, *:pipe-line-to shell-cmd*, *:pipe-to shell-cmd*, *:redirect-to [path]*, *:write-csv-to [–anonymize] path*, *:write-json-cols-to [–anonymize] path*, *:write-json-to [–anonymize] path*, *:write-jsonlines-to [–all] [–anonymize] path*, *:write-raw-to [–view] [–anonymize] path*, *:write-screen-to [–anonymize] path*, *:write-table-to [–anonymize] path*, *:write-to [–anonymize] path*, *:write-view-to [–anonymize] path*, *:xopen path*

12.9.190 ;.msgformats

Executes a query that will summarize the different message formats found in the logs

---

12.9.191 ;.read *path*

Execute the SQLite statements in the given file

Parameters

- **path*** — The path to the file to write

See Also

:append-to path, *;.dump path table*, *;.save path*, *:echo [-n] msg*, *echoln(value)*, *:export-session-to path*, *:open [–since] [–until] path*, *:pipe-line-to shell-cmd*, *:pipe-to shell-cmd*, *:redirect-to [path]*, *:write-csv-to [–anonymize] path*, *:write-json-cols-to [–anonymize] path*, *:write-json-to [–anonymize] path*, *:write-jsonlines-to [–all] [–anonymize] path*, *:write-raw-to [–view] [–anonymize] path*, *:write-screen-to [–anonymize] path*, *:write-table-to [–anonymize] path*, *:write-to [–anonymize] path*, *:write-view-to [–anonymize] path*, *:xopen path*

12.9.192 ;.save *path*

Save the current database to a SQLite file

Parameters

- **path*** — The path to the file to write

See Also

:append-to path, *;.dump path table*, *;.read path*, *:echo [-n] msg*, *echoln(value)*, *:export-session-to path*, *:open [–since] [–until] path*, *:pipe-line-to shell-cmd*, *:pipe-to shell-cmd*, *:redirect-to [path]*, *:write-csv-to [–anonymize] path*, *:write-json-cols-to [–anonymize] path*, *:write-json-to [–anonymize] path*, *:write-jsonlines-to [–all] [–anonymize] path*, *:write-raw-to [–view] [–anonymize] path*, *:write-screen-to [–anonymize] path*, *:write-table-to [–anonymize] path*, *:write-to [–anonymize] path*, *:write-view-to [–anonymize] path*, *:xopen path*

12.9.193 ;.schema *name*

Switch to the SCHEMA view that contains a dump of the current database schema

Parameters

- **name*** — The name of a table to jump to

12.9.194 aggregate *expr*

PRQL transform to summarize many rows into one

Parameters

- **expr*** — The aggregate expression(s)

Examples

To group values into a JSON array:

```
;from [{a=1}, {a=2}] | aggregate { arr = json.group_array a }
[1,2]
```

See Also

append table, *derive column*, *filter expr*, *from table*, *group key_columns pipeline*, *join [side:inner] table condition*, *select expr*, *sort expr*, *take n_or_range*, *stats.average_of col*, *stats.by col values*, *stats.count_by column*, *stats.hist col [slice:'1h'] [top:10]*, *stats.sum_of col*, *utils.distinct col*

12.9.195 append *table*

PRQL transform to concatenate tables together

Parameters

- **table*** — The table to use as a source

See Also

aggregate expr, *derive column*, *filter expr*, *from table*, *group key_columns pipeline*, *join [side:inner] table condition*, *select expr*, *sort expr*, *take n_or_range*, *stats.average_of col*, *stats.by col values*, *stats.count_by column*, *stats.hist col [slice:'1h'] [top:10]*, *stats.sum_of col*, *utils.distinct col*

12.9.196 derive *column*

PRQL transform to derive one or more columns

Parameters

- **column*** — The new column

Examples

To add a column that is a multiplication of another:

```
;from [{a=1}, {a=2}] | derive b = a * 2
    a           b
          1           2
          2           4
```

See Also

aggregate expr, *append table*, *filter expr*, *from table*, *group key_columns pipeline*, *join [side:inner] table condition*, *select expr*, *sort expr*, *take n_or_range*, *stats.average_of col*, *stats.by col values*, *stats.count_by column*, *stats.hist col [slice:'1h'] [top:10]*, *stats.sum_of col*, *utils.distinct col*

12.9.197 filter *expr*

PRQL transform to pick rows based on their values

Parameters

- **expr*** — The expression to evaluate over each row

Examples

To pick rows where 'a' is greater than one:

```
;from [{a=1}, {a=2}] | filter a > 1
2
```

See Also

aggregate expr, *append table*, *derive column*, *from table*, *group key_columns pipeline*, *join [side:inner] table condition*, *select expr*, *sort expr*, *take n_or_range*, *stats.average_of col*, *stats.by col values*, *stats.count_by column*, *stats.hist col [slice:'1h'] [top:10]*, *stats.sum_of col*, *utils.distinct col*

12.9.198 from *table*

PRQL command to specify a data source

Parameters

- **table*** — The table to use as a source

Examples

To pull data from the 'http_status_codes' database table:

```
;from http_status_codes | take 3
  status       message
       100 Continue
       101 Switchiotocols
       102 Processing
```

To use an array literal as a source:

```
;from [{ col1=1, col2='abc' }, { col1=2, col2='def' }]
   col1    col2
          1 abc
          2 def
```

See Also

aggregate expr, *append table*, *derive column*, *filter expr*, *group key_columns pipeline*, *join [side:inner] table condition*, *select expr*, *sort expr*, *take n_or_range*, *stats.average_of col*, *stats.by col values*, *stats.count_by column*, *stats.hist col [slice:'1h'] [top:10]*, *stats.sum_of col*, *utils.distinct col*

12.9.199 group *key_columns pipeline*

PRQL transform to partition rows into groups

Parameters

- **key_columns*** — The columns that define the group
- **pipeline*** — The pipeline to execute over a group

Examples

To group by log_level and count the rows in each partition:

```
;from lnav_example_log | group { log_level } (aggregate { count this })
log_level  COUNT(*)
debug              1
info               1
warn               1
error              1
```

See Also

aggregate expr, *append table*, *derive column*, *filter expr*, *from table*, *join [side:inner] table condition*, *select expr*, *sort expr*, *take n_or_range*, *stats.average_of col*, *stats.by col values*, *stats.count_by column*, *stats.hist col [slice:'1h'] [top:10]*, *stats.sum_of col*, *utils.distinct col*

12.9.200 join *[side:inner] table condition*

PRQL transform to add columns from another table

Parameters

- **side** — Specifies which rows to include
- **table*** — The other table to join with the current rows
- **condition*** — The condition used to join rows

See Also

aggregate expr, *append table*, *derive column*, *filter expr*, *from table*, *group key_columns pipeline*, *select expr*, *sort expr*, *take n_or_range*, *stats.average_of col*, *stats.by col values*, *stats.count_by column*, *stats.hist col [slice:'1h'] [top:10]*, *stats.sum_of col*, *utils.distinct col*

12.9.201 select *expr*

PRQL transform to pick and compute columns

Parameters

- **expr*** — The columns to include in the result set

Examples

To pick the 'b' column from the rows:

```
;from [{a=1, b='abc'}, {a=2, b='def'}] | select b
 b
abc
def
```

To compute a new column from an input:

```
;from [{a=1}, {a=2}] | select b = a * 2
    b
        2
        4
```

See Also

aggregate expr, *append table*, *derive column*, *filter expr*, *from table*, *group key_columns pipeline*, *join [side:inner] table condition*, *sort expr*, *take n_or_range*, *stats.average_of col*, *stats.by col values*, *stats.count_by column*, *stats.hist col [slice:'1h'] [top:10]*, *stats.sum_of col*, *utils.distinct col*

12.9.202 sort *expr*

PRQL transform to sort rows

Parameters

- **expr*** — The values to use when ordering the result set

Examples

To sort the rows in descending order:

```
;from [{a=1}, {a=2}] | sort {-a}
    a
         2
         1
```

See Also

aggregate expr, *append table*, *derive column*, *filter expr*, *from table*, *group key_columns pipeline*, *join [side:inner] table condition*, *select expr*, *take n_or_range*, *stats.average_of col*, *stats.by col values*, *stats.count_by column*, *stats.hist col [slice:'1h'] [top:10]*, *stats.sum_of col*, *utils.distinct col*

12.9.203 stats.average_of *col*

Compute the average of col

Parameters

- **col*** — The column to average

Examples

To get the average of a:

```
;from [{a=1}, {a=1}, {a=2}] | stats.average_of a
1.3333333333333333
```

See Also

aggregate expr, *append table*, *derive column*, *filter expr*, *from table*, *group key_columns pipeline*, *join [side:inner] table condition*, *select expr*, *sort expr*, *take n_or_range*, *stats.by col values*, *stats.count_by column*, *stats.hist col [slice:'1h'] [top:10]*, *stats.sum_of col*, *utils.distinct col*

12.9.204 stats.by *col values*

A shorthand for grouping and aggregating

Parameters

- **col*** — The column to sum
- **values*** — The aggregations to perform

Examples

To partition by a and get the sum of b:

```
;from [{a=1, b=1}, {a=1, b=1}, {a=2, b=1}] | stats.by a {sum b}
    a       COALESC(b), 0)
         1                2
         2                1
```

See Also

aggregate expr, *append table*, *derive column*, *filter expr*, *from table*, *group key_columns pipeline*, *join [side:inner] table condition*, *select expr*, *sort expr*, *take n_or_range*, *stats.average_of col*, *stats.count_by column*, *stats.hist col [slice:'1h'] [top:10]*, *stats.sum_of col*, *utils.distinct col*

12.9.205 stats.count_by *column*

Partition rows and count the number of rows in each partition

Parameters

- **column** — The columns to group by

Examples

To count rows for a particular value of column 'a':

```
;from [{a=1}, {a=1}, {a=2}] | stats.count_by a
    a         total
          1           2
          2           1
```

See Also

aggregate expr, *append table*, *derive column*, *filter expr*, *from table*, *group key_columns pipeline*, *join [side:inner] table condition*, *select expr*, *sort expr*, *take n_or_range*, *stats.average_of col*, *stats.by col values*, *stats.hist col [slice:'1h'] [top:10]*, *stats.sum_of col*, *utils.distinct col*

12.9.206 stats.hist *col [slice:'1h'] [top:10]*

Count the top values per bucket of time

Parameters

- **col*** — The column to count
- **slice** — The time slice
- **top** — The limit on the number of values to report

Examples

To chart the values of ex_procname over time:

```
;from lnav_example_log | stats.hist ex_procname
    tslice             v
2017-02.000000 {"gw":3,"hw":1}
```

To chart the values of ex_procname for every second:

```
;from lnav_example_log | stats.hist ex_procname slice:'1s'
    tslice             v
2017-02.000000 {"gw":1,"hw":1}
2017-02.000000 {"gw":1}
2017-02.000000 {"gw":1}
```

See Also

aggregate expr, *append table*, *derive column*, *filter expr*, *from table*, *group key_columns pipeline*, *join [side:inner] table condition*, *select expr*, *sort expr*, *take n_or_range*, *stats.average_of col*, *stats.by col values*, *stats.count_by column*, *stats.sum_of col*, *utils.distinct col*

12.9.207 stats.sum_of *col*

Compute the sum of col

Parameters

- **col*** — The column to sum

Examples

To get the sum of a:

```
;from [{a=1}, {a=1}, {a=2}] | stats.sum_of a
4
```

See Also

aggregate expr, *append table*, *derive column*, *filter expr*, *from table*, *group key_columns pipeline*, *join [side:inner] table condition*, *select expr*, *sort expr*, *take n_or_range*, *stats.average_of col*, *stats.by col values*, *stats.count_by column*, *stats.hist col [slice:'1h'] [top:10]*, *utils.distinct col*

12.9.208 take *n_or_range*

PRQL command to pick rows based on their position

Parameters

- **n_or_range*** — The number of rows or range

Examples

To pick the first row:

```
;from [{a=1}, {a=2}, {a=3}] | take 1
1
```

To pick the second and third rows:

```
;from [{a=1}, {a=2}, {a=3}] | take 2..3
    a
         2
         3
```

See Also

aggregate expr, *append table*, *derive column*, *filter expr*, *from table*, *group key_columns pipeline*, *join [side:inner] table condition*, *select expr*, *sort expr*, *stats.average_of col*, *stats.by col values*, *stats.count_by column*, *stats.hist col [slice:'1h'] [top:10]*, *stats.sum_of col*, *utils.distinct col*

12.9.209 utils.distinct *col*

A shorthand for getting distinct values of col

Parameters

- **col*** — The column to sum

Examples

To get the distinct values of a:

```
;from [{a=1}, {a=1}, {a=2}] | utils.distinct a
    a
         1
         2
```

See Also

aggregate expr, *append table*, *derive column*, *filter expr*, *from table*, *group key_columns pipeline*, *join [side:inner] table condition*, *select expr*, *sort expr*, *take n_or_range*, *stats.average_of col*, *stats.by col values*, *stats.count_by column*, *stats.hist col [slice:'1h'] [top:10]*, *stats.sum_of col*

CHAPTER

THIRTEEN

SQLITE TABLES REFERENCE

In addition to the tables generated for each log format, **lnav** includes the following tables/views:

- *environ*
- *fstat(<path|pattern>)*
- *lnav_events*
- *lnav_file*
- *lnav_file_metadata*
- *lnav_log_breakpoints*
- *lnav_user_notifications*
- *lnav_views*
- *lnav_views_echo*
- *lnav_view_files*
- *lnav_view_stack*
- *lnav_view_filters*
- *lnav_view_filter_stats*
- *lnav_view_filters_and_stats*
- *lnav_top_view*
- *all_logs*
- *all_opids*
- *all_thread_ids*
- *lnav_focused_msg*
- *http_status_codes*
- *regexp_capture(<string>, <regex>)*

These extra tables provide useful information and can let you manipulate **lnav**'s internal state. You can get a dump of the entire database schema by executing the '.schema' SQL command, like so:

```
;.schema
```

> **Note**
>
> The tables created by lnav are in the `lnav_db` database. The `main` SQLite database is left empty for your own use. If you end up creating any tables while processing your logs, you can use the *;.save path* SQL command to save the `main` database to a SQLite database file.

13.1 environ

The `environ` table gives you access to the **lnav** process' environment variables. You can `SELECT`, `INSERT`, and `UPDATE` environment variables, like so:

```
;SELECT * FROM environ WHERE name = 'SHELL'
 name   value
SHELL /bin/tcsh

;UPDATE environ SET value = '/bin/sh' WHERE name = 'SHELL'
```

Environment variables can be used to store simple values or pass values from **lnav**'s SQL environment to **lnav**'s commands. For example, the `:open` command will do variable substitution, so you can insert a variable named "FILENAME" and then open it in **lnav** by referencing it with "$FILENAME":

```
;INSERT INTO environ VALUES ('FILENAME', '/path/to/file')
:open $FILENAME
```

13.2 fstat(<path|pattern>)

The `fstat` table-valued function provides access to the local file system. The function takes a file path or a glob pattern and returns the results of `lstat(2)` for the matching files. If the parameter is a pattern that matches nothing, no rows will be returned. If the parameter is a path for a non-existent file, a row will be returned with the `error` column set and the stat columns as `NULL`. To read the contents of a file, you can `SELECT` the hidden `data` column.

13.3 lnav_events

The `lnav_events` table allows you to react to events that occur while **lnav** is running using SQLite triggers. For example, when a file is opened, a row is inserted into the `lnav_events` table that contains a timestamp and a JSON object with the event ID and the path of the file. The following columns are available in this table:

ts
: The timestamp of the event.

content
: A JSON object that contains the event information. See the *Reference* for more information about the types of events that are available.

13.4 lnav_file

The `lnav_file` table allows you to examine and perform limited updates to the metadata for the files that are currently loaded into **lnav**. The following columns are available in this table:

device
: The device the file is stored on.

inode
The inode for the file on the device.

filepath
If this is a real file, it will be the absolute path. Otherwise, it is a symbolic name. If it is a symbolic name, it can be UPDATEd so that this file will be considered when saving and loading session information.

mimetype
The detected MIME type of the file.

content_id
The hash of some unique content in the file.

format
The log file format for the file.

lines
The number of lines in the file.

time_offset
The millisecond offset for timestamps. This column can be UPDATEd to change the offset of timestamps in the file.

options_path
Options can be applied to files based on a path or glob pattern. If this file matches a set of options, the matching path/pattern is available in this column and the actual options themselves are in the `options` column.

options
The options that are applicable to this file. Currently, the only options available are for the timezone set by the *:set-file-timezone* command.

13.5 lnav_file_metadata

The `lnav_file_metadata` table gives access to metadata associated with a loaded file. Currently,

filepath
The path to the file.

descriptor
A descriptor that identifies the source of the metadata. The following descriptors are supported:

net.zlib.gzip.header
The header on a gzipped file. The content is a JSON object with the following properties:

name
The original name of the file.

mtime
The last modified time of the file when it was compressed.

comment
A text comment associated with the file.

net.daringfireball.markdown.frontmatter
The frontmatter on a markdown file. If the frontmatter is delimited by three dashes (`---`), the `mimetype` will be `application/yaml`. If the frontmatter is delimited by three pluses (`+++`) the `mimetype` will be `application/toml`.

mimetype
The MIME type of the metadata.

content
The metadata itself.

13.6 lnav_log_breakpoints

The `lnav_log_breakpoints` table allows you to view and manage breakpoints set on log messages. Breakpoints mark log messages that share a particular source file location or message schema, making it easy to navigate between related log lines using the `F7` and `F8` keys.

You can `SELECT`, `INSERT`, `UPDATE`, and `DELETE` breakpoints through this table. The columns in the table are as follows:

schema_id
The schema identifier for the breakpoint. This value matches the `log_msg_schema` column in the `all_logs` table.

description
A human-readable description of the breakpoint (e.g. `format_name:file.cc:42`).

type
The source of the schema ID, either `src_location` or `message_schema`.

enabled
Indicates whether the breakpoint is active (1 or 0).

```
;SELECT * FROM lnav_log_breakpoints

;UPDATE lnav_log_breakpoints SET enabled = 0 WHERE description LIKE '%main.cc%'

;DELETE FROM lnav_log_breakpoints WHERE description LIKE '%test%'
```

13.7 lnav_user_notifications

The `lnav_user_notifications` table allows you to display a custom message in the top-right corner of the UI. For example, to display "Hello, World!", you can enter:

```
;REPLACE INTO lnav_user_notifications (message) VALUES ('Hello, World!')
```

There are additional columns to have finer control of what is displayed and when:

id
The unique ID for the message, defaults to "org.lnav.user". This is the primary key for the table, so more than one type of message is not allowed.

priority
The priority of the message. Higher priority messages will be displayed until they are cleared or are expired.

created
The time the message was created.

expiration
The time when the message should expire or NULL if it should not automatically expire.

views
A JSON array of view names where the message is applicable or NULL if the message should be shown in all views.

message
The message itself.

This table will most likely be used in combination with *Events (v0.11.0+)* and the *lnav_views_echo* table.

13.8 lnav_views

The `lnav_views` table allows you to SELECT and UPDATE information related to **lnav**'s "views" (e.g. log, text, ...). The following columns are available in this table:

name
The name of the view.

top
The line number at the top of the view. This value can be UPDATEd to move the view to the given line.

left
The left-most column number to display. This value can be UPDATEd to move the view left or right.

height
The number of lines that are displayed on the screen.

inner_height
The number of lines of content being displayed.

top_time
The timestamp of the top line in the view or NULL if the view is not time-based. This value can be UPDATEd to move the view to the given time.

top_file
The file the top line in the view is from.

paused
Indicates if the view is paused and will not load new data.

search
The search string for this view. This value can be UPDATEd to initiate a text search in this view.

filtering
Indicates if the view is applying filters.

movement
The movement mode, either 'top' or 'cursor'.

top_meta
A JSON object that contains metadata related to the top line in the view.

selection
The number of the line that is focused for selection.

options
A JSON object that contains optional settings for this view.

13.9 lnav_views_echo

The `lnav_views_echo` table is a real SQLite table that you can create TRIGGERs on in order to react to users moving around in a view.

> **Note**
>
> The table is periodically updated to reflect the current state of the views. The changes are *not* performed immediately after the user action.

13.10 lnav_view_files

The `lnav_view_files` table provides access to details about the files displayed in a particular view. The main purpose of this table is to allow you to programmatically control which files are shown / hidden in the view. The following columns are available in this table:

view_name
: The name of the view.

filepath
: The file's path.

visible
: Determines whether the file is visible in the view. This column can be changed using an `UPDATE` statement to hide or show the file.

13.11 lnav_view_stack

The `lnav_view_stack` table allows you to `SELECT` and `DELETE` from the stack of **lnav** "views" (e.g. log, text, ...). The following columns are available in this table:

name
: The name of the view.

13.12 lnav_view_filters

The `lnav_view_filters` table allows you to manipulate the filters in the **lnav** views. The following columns are available in this table:

view_name
: The name of the view the filter is applied to.

filter_id
: The filter identifier. This will be assigned on insertion.

enabled
: Indicates whether this filter is enabled or disabled.

type
: The type of filter, either 'in' or 'out'.

pattern
: The regular expression to filter on.

This table supports `SELECT`, `INSERT`, `UPDATE`, and `DELETE` on the table rows to read, create, update, and delete filters for the views.

13.13 lnav_view_filter_stats

The `lnav_view_filter_stats` table allows you to get information about how many lines matched a given filter. The following columns are available in this table:

view_name
The name of the view.

filter_id
The filter identifier.

hits
The number of lines that matched this filter.

This table is read-only.

13.14 lnav_view_filters_and_stats

The `lnav_view_filters_and_stats` view joins the `lnav_view_filters` table with the `lnav_view_filter_stats` table into a single view for ease of use.

13.15 lnav_top_view

The `lnav_top_view` view returns the row for the top view on the view stack.

13.16 all_logs

The `all_logs` table lets you query the format derived from the **lnav** log message parser that is used to automatically extract data, see *Extracting Data* for more details.

13.17 all_opids

The `all_opids` table contains information about all opids that were found in the log files or set via the `log_opid` column on the log vtables. The information in this table is the same as available through the *TIMELINE* view. The `description` column can be `SET` in an `UPDATE` statement to customize the description shown in the timeline.

13.18 all_thread_ids

The `all_thread_ids` table contains information about all the thread identifiers that were found in logs. Log formats can specify which field is a thread identifier with the `thread-id-field` property.

13.19 lnav_focused_msg

The `lnav_focused_msg` view returns the row for the focused log message from the `all_logs` table.

13.20 http_status_codes

The `http_status_codes` table is a handy reference that can be used to turn HTTP status codes into human-readable messages.

13.21 regexp_capture(<string>, <regex>)

The `regexp_capture()` table-valued function applies the regular expression to the given string and returns detailed results for the captured portions of the string.

CHAPTER

FOURTEEN

EVENTS (V0.11.0+)

The events mechanism allows **lnav** to be automated based on events that occur during processing. For example, filters could be added only when a particular log file format is detected instead of always installing them. Events are published through the *lnav_events* SQLite table. Reacting to events can be done by creating a SQLite trigger on the table and inspecting the content of the event.

14.1 Trigger Example

The following is an example of a trigger that adds an out filter when a syslog file is loaded. You can copy the code into an `.sql` file and install it by running `lnav -i my_trigger.sql`.

Listing 1: my_trigger.sql

```
CREATE TRIGGER IF NOT EXISTS lnav_db.add_format_specific_filters
  AFTER INSERT ON lnav_db.lnav_events WHEN
    -- Check the event type
    jget(NEW.content, '/$schema') =
      'https://lnav.org/event-file-format-detected-v1.schema.json' AND
    -- Only create the filter when a given format is seen
    jget(NEW.content, '/format') = 'syslog_log' AND
    -- Don't create the filter if it's already there
    NOT EXISTS (
      SELECT 1 FROM lnav_view_filters WHERE pattern = 'noisy message')
BEGIN
INSERT INTO lnav_view_filters (view_name, enabled, type, pattern) VALUES
    ('log', 1, 'OUT', 'noisy message');
END;
```

14.2 Reference

The following tables describe the schema of the event JSON objects.

14.2.1 https://lnav.org/event-file-open-v1.schema.json

Event fired when a file is opened.

https://lnav.org/event-file-open-v1.schema.json	
properties	
• filename	*/filename*
	The path of the file that was opened
	type *string*
additionalProperties	False

14.2.2 https://lnav.org/event-file-format-detected-v1.schema.json

Event fired when a log format is detected for a file.

https://lnav.org/event-file-format-detected-v1.schema.json	
properties	
• filename	*/filename*
	The path of the file for which a matching format was found
	type *string*
• format	*/format*
	The name of the format
	type *string*
additionalProperties	False

14.2.3 https://lnav.org/event-log-msg-detected-v1.schema.json

Event fired when a log message is detected by a watch expression.

https://lnav.org/event-log-msg-detected-v1.schema.json		
properties		
• watch-name	*/watch-name*	
	The name of the watch expression that matched this log message	
	type *string*	
• filename	*/filename*	
	The path of the file containing the log message	
	type *string*	
• line-number	*/line-number*	
	The line number in the file, starting from zero	
	type *integer*	
• format	*/format*	
	The name of the log format that matched this log message	
	type *string*	
• timestamp	*/timestamp*	
	The timestamp of the log message	
	type *string*	
• values	*/values*	
	The log message values captured by the log format	
	type *object*	
	patternProperties	
	• ^([\w\-]+)$	*/values/<name>*
		type *boolean / integer / string*
	additionalProperties	False
additionalProperties	False	

14.2.4 https://lnav.org/event-session-loaded-v1.schema.json

Event fired when a session is loaded.

https://lnav.org/event-session-loaded-v1.schema.json	
additionalProperties	False

CHAPTER

FIFTEEN

EXTERNAL ACCESS (V0.14.0+)

The "External Access" feature opens a local HTTP port that can be used to interact with an lnav instance outside of the TUI. This feature can be enabled with the *:external-access port api-key* command. Once the port is open, HTTP requests can be sent to access static files, execute commands, or poll for changes. When the external port is open, a globe icon () is displayed in the top-right corner. Clicking that icon will open a URL in a browser and log you into the server. The *:external-access-login [app]* command can also be used to login.

Note

The server only binds to `localhost`, so it will not be accessible over the network. If you need to access lnav remotely, consider using SSH forwarding.

15.1 Authentication

All requests to lnav's external access server are authenticated. A request must have one of the following:

- An `X-Api-Key` header with the Base64-encoded value of the API-key that was passed to the `:external-access` command. This header should be used for automations.
- An `lnav_session_id` cookie. This cookie will be set through the flow initiated by the *:external-access-login [app]* command that opens the `/login` URL using the configured *external-opener*. The `/login` URL accepts a one-time-password query parameter and, if it matches, the session cookie will be created. A single one-time-password is possible at any time and sessions are only valid for the current invocation of lnav.

15.2 Endpoints

The following routes are available:

- `GET /api/version`

  Get the version of the lnav instance.

- `POST /api/exec`
  `Content-Type: text/x-lnav-script`

  Execute an lnav *script* and receive the resulting output.

- `POST /api/poll`
  `Content-Type: application/json`

  Perform a long-poll of lnav's TUI state. The first request to this API should be a `null` to get the current state. The response contains the following fields:

- `next_input` - Subsequent calls should send this object so the server knows when there has been a state-change with respect to this client. Currently, the `view_states/log_selection` field is the only stable field and refers to the focused message in the LOG view.
- `background_tasks` - A list of background task progress updates.

15.3 Apps

To support custom user-interfaces on top of lnav, "apps" can be installed that are reachable via the external access server. These browser-based apps can provide a rich interface for executing queries and presenting their results. For example, a dashboard for `access_log` files can display multiple charts for interesting request statistics. Apps are reachable from the landing page for the external access server or you can open one directly by passing its ID to the *:external-access-login [app]* command. An app ID has the form `<publisher>/<app-name>` (e.g. `lnav/api-test`).

Creating an app can be done using the *management CLI*, like so:

```
lnav -m apps create mydash
```

This command creates a directory in the `configs` directory and populates it with the necessary configuration file and a sample `index.md` file.

15.3.1 Test Harness

The `lnav/api-test` app is included by default as a demonstration of the external access server APIs.

15.3.2 Reference

The following are the configuration properties necessary to define an app:

/apps

Application definitions			
type	*object*		
patternProperties			
• ^([\w\]+)$	*/apps/<publisher>*		
	The publisher of the application		
	type	*object*	
	patternProperties		
	• ^([\w\]+)$	*/apps/<publisher>/<app_name>*	
		The application definition	
		type	*object*
		properties	
		• root	*/apps/<publisher>/<app_name>/root*
			The path to use as the root for files that are a part of this application. The path is relative to the location of the file containing this property.
			type *string*
			pattern [\w\-\.]+
		• description	*/apps/<publisher>/<app_name>/description*
			A description of this application
			type *string*
		additionalProperties	False
	additionalProperties	False	
additionalProperties	False		

CHAPTER

SIXTEEN

EXTRACTING DATA

Note: This feature is still in **BETA**, you should expect bugs and incompatible changes in the future.

Log messages contain a good deal of useful data, but it's not always easy to get at. The log parser built into **lnav** is able to extract data as described by *Log Formats* as well as discovering data in plain text messages. This data can then be queried and processed using the SQLite front-end that is also incorporated into **lnav**. As an example, the following Syslog message from sudo can be processed to extract several key/value pairs:

```
Jul 31 11:42:26 Example-MacBook-Pro.local sudo[87024]:  testuser : TTY=ttys004 ; PWD=/
↪Users/testuser/github/lbuild ; USER=root ; COMMAND=/usr/bin/make install
```

The data that can be extracted by the parser is viewable directly in **lnav** by pressing the 'p' key. The results will be shown in an overlay like the following:

```
Current Time: 2013-07-31T11:42:26.000  Original Time: 2013-07-31T11:42:26.000  Offset:␣
↪+0.000
Known message fields:
├ log_hostname = Example-MacBook-Pro.local
├ log_procname = sudo
├ log_pid      = 87024
Discovered message fields:
├ col_0        = testuser
├ TTY          = ttys004
├ PWD          = /Users/testuser/github/lbuild
├ USER         = root
└ COMMAND      = /usr/bin/make install
```

Notice that the parser has detected pairs of the form '<key>=<value>'. The data parser will also look for pairs separated by a colon. If there are no clearly demarcated pairs, then the parser will extract anything that looks like data values and assign them keys of the form 'col_N'. For example, two data values, an IPv4 address and a symbol, will be extracted from the following log message:

```
Apr 29 08:13:43 sample-centos5 avahi-daemon[2467]: Registering new address record for 10.
↪1.10.62 on eth0.
```

Since there are no keys for the values in the message, the parser will assign 'col_0' for the IP address and 'col_1' for the symbol, as seen here:

```
Current Time: 2013-04-29T08:13:43.000  Original Time: 2013-04-29T08:13:43.000  Offset:␣
↪+0.000
Known message fields:
├ log_hostname = sample-centos5
├ log_procname = avahi-daemon
```

(continues on next page)

(continued from previous page)

```
├ log_pid      = 2467
Discovered message fields:
├ col_0        = 10.1.10.62
└ col_1        = eth0
```

Now that you have an idea of how the parser works, you can begin to perform queries on the data that is being extracted. The SQLite database engine is embedded into **lnav** and its Virtual Table mechanism is used to provide a means to process this log data. Each log format has its own table that can be used to access all of the loaded messages that are in that format. For accessing log message content that is more free-form, like the examples given here, the **logline** table can be used. The **logline** table is recreated for each query and is based on the format and pairs discovered in the log message at the top of the display.

Queries can be performed by pressing the semi-colon (;) key in **lnav**. After pressing the key, the overlay showing any known or discovered fields will be displayed to give you an idea of what data is available. The query can be any SQL query supported by SQLite. To make analysis easier, **lnav** includes many extra functions for processing strings, paths, and IP addresses. See *SQLite Interface* for more information.

As an example, the simplest query to perform initially would be a "select all", like so:

```
SELECT * FROM logline
```

When this query is run against the second example log message given above, the following results are received:

```
log_line log_part         log_time        log_idle_msecs log_level  log_hostname  log_
↪procname log_pid           col_0          col_1

     292 p.0      2013-04-11T16:42:51.000              0 info      localhost      avahi-
↪daemon     2480    fe80::a00:27ff:fe98:7f6e eth0
     293 p.0      2013-04-11T16:42:51.000              0 info      localhost      avahi-
↪daemon     2480    10.0.2.15                eth0
     330 p.0      2013-04-11T16:47:02.000              0 info      localhost      avahi-
↪daemon     2480    fe80::a00:27ff:fe98:7f6e eth0
     336 p.0      2013-04-11T16:47:02.000              0 info      localhost      avahi-
↪daemon     2480    10.1.10.75               eth0
     343 p.0      2013-04-11T16:47:02.000              0 info      localhost      avahi-
↪daemon     2480    10.1.10.75               eth0
     370 p.0      2013-04-11T16:59:39.000              0 info      localhost      avahi-
↪daemon     2480    10.1.10.75               eth0
     377 p.0      2013-04-11T16:59:39.000              0 info      localhost      avahi-
↪daemon     2480    10.1.10.75               eth0
     382 p.0      2013-04-11T16:59:41.000              0 info      localhost      avahi-
↪daemon     2480    fe80::a00:27ff:fe98:7f6e eth0
     401 p.0      2013-04-11T17:20:45.000              0 info      localhost      avahi-
↪daemon     4247    fe80::a00:27ff:fe98:7f6e eth0
     402 p.0      2013-04-11T17:20:45.000              0 info      localhost      avahi-
↪daemon     4247    10.1.10.75               eth0

     735 p.0      2013-04-11T17:41:46.000              0 info      sample-centos5 avahi-
↪daemon     2465    fe80::a00:27ff:fe98:7f6e eth0
     736 p.0      2013-04-11T17:41:46.000              0 info      sample-centos5 avahi-
↪daemon     2465    10.1.10.75               eth0
     781 p.0      2013-04-12T03:32:30.000              0 info      sample-centos5 avahi-
↪daemon     2465    10.1.10.64               eth0
```

(continues on next page)

(continued from previous page)

```
     788 p.0      2013-04-12T03:32:30.000              0 info      sample-centos5 avahi-
↪daemon     2465    10.1.10.64               eth0
    1166 p.0      2013-04-25T10:56:00.000              0 info      sample-centos5 avahi-
↪daemon     2467    fe80::a00:27ff:fe98:7f6e eth0
    1167 p.0      2013-04-25T10:56:00.000              0 info      sample-centos5 avahi-
↪daemon     2467    10.1.10.111              eth0
    1246 p.0      2013-04-26T06:06:25.000              0 info      sample-centos5 avahi-
↪daemon     2467    10.1.10.49               eth0
    1253 p.0      2013-04-26T06:06:25.000              0 info      sample-centos5 avahi-
↪daemon     2467    10.1.10.49               eth0
    1454 p.0      2013-04-28T06:53:55.000              0 info      sample-centos5 avahi-
↪daemon     2467    10.1.10.103              eth0
    1461 p.0      2013-04-28T06:53:55.000              0 info      sample-centos5 avahi-
↪daemon     2467    10.1.10.103              eth0

    1497 p.0      2013-04-29T08:13:43.000              0 info      sample-centos5 avahi-
↪daemon     2467    10.1.10.62               eth0
    1504 p.0      2013-04-29T08:13:43.000              0 info      sample-centos5 avahi-
↪daemon     2467    10.1.10.62               eth0
```

Note that **lnav** is not returning results for all messages that are in this syslog file. Rather, it searches for messages that match the format for the given line and returns only those messages in results. In this case, that format is "Registering new address record for <IP> on <symbol>", which corresponds to the parts of the message that were not recognized as data.

More sophisticated queries can be done, of course. For example, to find out the frequency of IP addresses mentioned in these messages, you can run:

```
SELECT col_0,count(*) FROM logline GROUP BY col_0
```

The results for this query are:

```
          col_0          count(*)

10.0.2.15                      1
10.1.10.49                     2
10.1.10.62                     2
10.1.10.64                     2
10.1.10.75                     6
10.1.10.103                    2
10.1.10.111                    1
fe80::a00:27ff:fe98:7f6e       6
```

Since this type of query is fairly common, **lnav** includes a "summarize" command that will compute the frequencies of identifiers as well as min, max, average, median, and standard deviation for number columns. In this case, you can run the following to compute the frequencies and return an ordered set of results:

```
:summarize col_0
```

16.1 Recognized Data Types

When searching for data to extract from log messages, **lnav** looks for the following set of patterns:

Strings
Single and double-quoted strings. Example: “The quick brown fox.”

URLs
URLs that contain the ‘://’ separator. Example: http://example.com

Paths
File system paths. Examples: /path/to/file, ./relative/path

MAC Address
Ethernet MAC addresses. Example: c4:2c:03:0e:e4:4a

Hex Dumps
A colon-separated string of hex numbers. Example: e8:06:88:ff

Date/Time
Date and time stamps of the form “YYYY-mm-DD” and “HH:MM:SS”.

IP Addresses
IPv4 and IPv6 addresses. Examples: 127.0.0.1, fe80::c62c:3ff:fe0e:e44a%en0

UUID
The common formatting for 128-bit UUIDs. Example: 0E305E39-F1E9-4DE4-B10B-5829E5DF54D0

Version Numbers
Dot-separated version numbers. Example: 3.7.17

Numbers
Numbers in base ten, hex, and octal formats. Examples: 1234, 0xbeef, 0777

E-Mail Address
Strings that look close to an e-mail address. Example: gary@example.com

Constants
Common constants in languages, like: true, false, null, None.

Symbols
Words that follow the common conventions for symbols in programming languages. For example, containing all capital letters, or separated by colons. Example: SOME_CONSTANT_VALUE, namespace::value

CHAPTER

SEVENTEEN

PERFORMANCE

This section provides some performance measurements of **lnav** and standard utilities. The numbers in here are for a fairly large file to demonstrate that lnav is at least competitive with existing tools if not better. Since lnav is purpose-built for working with logs, it is able to perform better and provide a friendlier experience than traditional tooling.

The measurements are done using this E-commerce access log file available on Kaggle. The file is 3.3GB in size with over 10 million lines. Tests are run on an M2 MacBook Air with 24GB of RAM and macOS 15.2. Memory usage was measured with Activity Monitor. If you have any concerns about the results in here, please open an issue.

The following chart displays the performance results. The results are stored in this SQLite DB. The DB is updated using the update-perf-db.lnav script. The report is generated using the report-perf.lnav script.

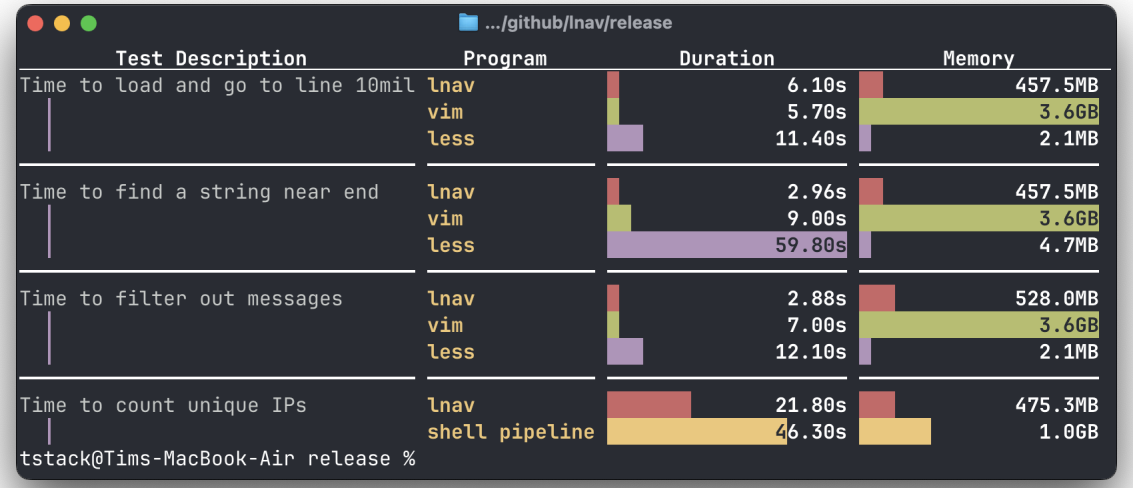

Fig. 1: Performance results as displayed in lnav's DB view.

The rest of this section goes into more detail about the tests themselves.

17.1 Indexing

The indexing benchmark tests how long it takes to open the test file and go to line 10 million.

- `lnav` displays the UI immediately and shows progress while loading the file. The medium-sized memory footprint is due to `lnav` creating an index entry for each line in the file. The index allows for immediate jumps to different parts of the log by time, log level, and line number.

- `vim` loads the entire file without providing any feedback during the process. The large memory footprint is due to `vim` loading the entire file into memory.
- `less` displays the start of the file immediately, but gives no feedback when scanning to the end of the file. The small memory footprint means there is no index and `less` needs to scan the file for many operations.

17.2 Searching

The search benchmark tests how long it takes to search for the text "26/Jan/2019:08:59:13", which is at the end of the file. The search is started after the file has been loaded by the program.

- `lnav` runs the search in the background and moves a bar back-and-forth to show activity.
- `vim` blocks the UI while it scans for the string.
- `less` blocks the UI while it scans for the string.

17.3 Filtering

The filtering benchmark tests how long it takes to filter out the word "Panasonic" and then go to the end of the file.

- `lnav` blocks the UI and moves a bar back-and-forth to show activity while evaluating the filter and updating the index.
- `vim` blocks the UI while evaluating the filter and then pops up a sub-window displaying the results.
- `less` applies the filter immediately and then blocks while computing line numbers to go to the end of the file.

17.4 Statistics

The statistics benchmark tests how long it takes to count the number of unique IP addresses. Since `vim` and `less` cannot do statistics, a pipeline is built with `cut`, `sort`, and `uniq`. Working with the client IP was chosen since it was first column in the log message. Performing analyses on other parts of the log message would require more work to extract the relevant data with standard tooling.

- `lnav` runs the query in the foreground and progress is shown in the bottom right. When finished, lnav switches to the DB view where the results are shown with a bar chart in the "total" column to help visualize the values. The PRQL query `from access_log | stats.count_by { c_ip }` was used to count the unique IPs.
- `shell pipeline` refers to the following pipeline: `cut -d ' ' -f 1 shop-access.log | sort | uniq | sort -n`. Executing that pipeline executes the whole time without giving feedback and dumps to the terminal. The memory usage listed in the chart is from `sort`. I'm not sure why it is so big…

CHAPTER

EIGHTEEN

HOW IT WORKS

"Magic"

18.1 Internal Architecture

The ARCHITECTURE.md file in the source tree contains some information about lnav's internals.

CHAPTER

NINETEEN

FREQUENTLY ASKED QUESTIONS

19.1 Q: How can I copy & paste without decorations?

Answer

There are a couple ways to do this:

- Use the *bookmark* hotkeys to mark lines and then press `c` to copy to the local system keyboard. The system clipboard is accessed using commands like `pbcopy` and `xclip`. See the *Tuning* section for more details.

  If a system clipboard is not available, the OSC 52 terminal escape sequence will be tried. If your terminal supports this escape sequence, the selected text will be copied to the clipboard, even if you are on an SSH connection.

- Press `CTRL + l` to temporarily switch to "lo-fi" mode where the contents of the current view are printed to the terminal. This option is useful when you are logged into a remote host.

19.2 Q: How can I force a format for a file?

Answer

The log format for a file is automatically detected and cannot be forced.

Solution

First, you need to get an idea of why the file is not being detected by the expected format using one of the following methods:

- Use the *format test* management command to check the file against the format. This command will give some details about why the file is not matching the format. For example, the following command will check the file `my_access.log` against the `access_log` format:

```
lnav -m format access_log test my_access.log
```

- Add some of the log file lines to the *sample* array and then startup lnav to get a detailed explanation of where the format patterns are not matching the sample lines.

After you get an idea of why the regular expressions are not matching, you can update the existing regexes or add a new one. For formats where you are not the author, you can create a *patch*.

Details

The first lines of the file are matched against the *regular expressions defined in the format definitions*. The order of the formats is automatically determined so that more specific formats are tried before more generic ones. Therefore, if the expected format is not being chosen for a file, then it means the regular expressions defined by that format are not matching the first few lines of the file.

See *Format Order When Scanning a File* for more information.

19.3 Q: How can I search backwards, like pressing ? in less?

Answer

Searches in **lnav** runs in the background and do not block input waiting to find the first hit. While the search prompt is open, pressing `CTRL` + `j` will jump to the previous hit that was found. A preview panel is also opened that shows the hits that have been found so far.

After pressing `Enter` at the search prompt, the view will jump to the first hit that was found. Then, you can press `n` to move to the next search hit and `N` to move to the previous one. If you would like to add a hotkey for jumping to the previous hit by default, enter the following configuration command:

```
:config /ui/keymap-defs/default/x3f/command :prompt --alt search ?
```

19.4 Q: Why isn't my log file highlighted correctly?

TBD

19.5 Q: Why isn't a file being displayed?

Answer

Plaintext files are displayed separately from log files in the TEXT view.

Solution

Press the `t` key to switch to the text view. Or, open the files configuration panel by pressing `TAB` to cycle through the panels, and then press `/` to search for the file you're interested in. If the file is a log, a new *log format* will need to be created or an existing one modified.

Details

If a file being monitored by lnav does not match a known log file format, it is treated as plaintext and will be displayed in the TEXT view.

19.6 Q: How can I search for an exact word?

Solution

Surround the word to search for with `\b`.

Details

The `\b` means "word break" and matches a position where a "word" ends. That is, right before a space character, punctuation, etc, or at EOL.

CHAPTER

TWENTY

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

Symbols

A

C

E

F

H

P

R

T

U

X