Inav Documentation

Release 0.9.1

Tim Stack

Mar 02, 2021

CONTENTS

The Log File Navigator (**lnav**) is an advanced log file viewer for the console.

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*.

1.1 Dependencies

When compiling from source, the following dependencies are required:

- NCurses
- PCRE – Versions greater than 8.20 give better performance since the PCRE JIT will be leveraged.
- SQLite
- ZLib
- Bzip2
- Readline
- libcurl
- libarchive

1.2 Installation

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

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

1.3 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
```

1.4 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.

1.4.1 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

uk United Kingdom

us United States

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

1.4.2 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, eldar, grayscale, monocai, night-owl, solarized-dark, and solarized-light.

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

1.4.3 Log Formats

In order for **lnav** to understand your log files, it needs to 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.

CHAPTER

TWO

USAGE

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

2.1 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.

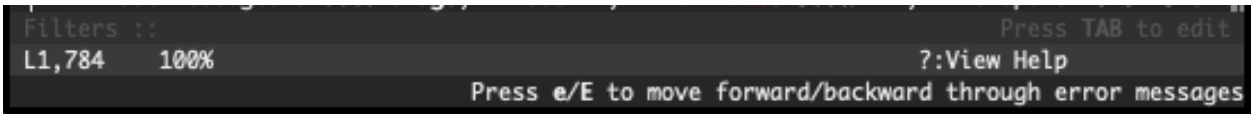

Fig. 1: When **lnav** is first open, it suggests using `e` and `Shift` + `e` to jump to error messages.

2.2 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 and displayed in the LOG view. Plain text files can be viewed in the TEXT view, which can be accessed by pressing `t`.

2.2.1 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 `$TMPDIR/lnav-${UID}-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.

2.3 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.

2.4 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.

2.4.1 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.

2.4.2 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.

2.4.3 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.

2.4.4 Log level

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

2.5 Search Tables

TBD

2.6 Taking Notes

A few of the columns in the log tables can be updated on a row-by-row basis to allow you to take notes. 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[1], 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
```

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

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.

CHAPTER

THREE

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.

3.1 Log Formats

TBD

3.1.1 Defining a New Format

TBD

3.2 Annotating Logs

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

3.2.1 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 1: 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)';
```

3.2.2 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 2: 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';
```

3.3 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.

3.3.1 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
```

3.3.2 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
```

3.3.3 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 3: 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 (
        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 (
        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
                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,
```

(continues on next page)

(continued from previous page)

```
56          printf('%9s', humanize_file_size(bytes)) as Amount,
57          sparkline(hits) AS Weeks
58      FROM weekly_hits
59      LEFT JOIN top_paths ON top_paths.cs_uri_stem = weekly_hits.cs_uri_stem
60      GROUP BY weekly_hits.cs_uri_stem
61      ORDER BY Hits DESC
62      LIMIT 10
63
64  :write-table-to -
65
66  :echo
67  :echo Failed Requests
68  :echo
69
70  ;SELECT printf('%,9d', count(1)) AS Hits,
71          printf('%,9d', count(distinct c_ip)) AS Visitors,
72          sc_status AS Status,
73          cs_method AS Method,
74          group_concat(distinct cs_version) AS Versions,
75          cs_uri_stem AS Path,
76          replicate('|', (cast(count(1) AS REAL) / $total_requests) * 100.0) AS "% of␣
    ↪Requests"
77      FROM access_log
78      WHERE sc_status >= 400
79      GROUP BY cs_method, cs_uri_stem
80      ORDER BY Hits DESC
81      LIMIT 10
82
83  :write-table-to -
```

CHAPTER

FOUR

HOW IT WORKS

“Magic”

CHAPTER

FIVE

CONFIGURATION

The configuration for **lnav** is stored in the following JSON files in `~/.lnav`:

- `config.json` – Contains local customizations that were done using the `:config` command.
- `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.
- `configs/installed/*.json` – Contains configuration files installed using the `-i` flag (e.g. `$ lnav -i /path/to/config.json`).
- `configs/*/*.json` – Other directories that contain `*.json` files will be loaded on startup. This structure is convenient for installing **lnav** configurations, like from a git repository.

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.

5.1 Options

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

5.1.1 /ui/keymap

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

5.1.2 /ui/theme

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

5.1.3 /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

5.1.4 /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*

5.1.5 /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*

5.2 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.

5.2.1 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:

- black
- red
- green
- yellow
- blue
- magenta

- cyan
- white

5.2.2 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.

5.2.3 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"
                    }
                }
            }
        }
    }
}
```

5.2.4 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	*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
• 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
• 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
• 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
• offset-time	*/ui/theme-defs/<theme_name>/styles/offset-time*
	Styling for hidden fields
	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
• 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

continues on next page

Table 1 – continued from previous page

additionalProperties	False

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

Styles for syntax highlighting in text files.	
type	*object*
properties	
• 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
• number	*/ui/theme-defs/<theme_name>/syntax-styles/number*
	Styling for numbers 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
• 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
	style
• active	*/ui/theme-defs/<theme_name>/status-styles/active*
	Styling for activity in 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
• 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
• hotkey	*/ui/theme-defs/<theme_name>/status-styles/hotkey*
	Styling for hotkey highlights of status bars
	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		
• color	*/color*	
	The foreground color value for this style. The value can be the name of an xterm color, the hexadecimal value, 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 hexadecimal 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.	
	type	*boolean*
additionalProperties	False	

5.3 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.

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

5.3.1 Key Sequence Encoding

Key presses are converted into a hex-encoded string that is used to lookup an entry in the keymap. Each byte of the keypress value is 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.

5.3.2 Reference

/ui/keymap-defs/<keymap_name>

The keymap definitions		
type	*object*	
patternProperties		
• ((?:x[0-9a-f]{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	
	• 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	

5.4 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.

5.4.1 /tuning

Internal settings			
type	*object*		
properties			
• archive-manager	*/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	
additionalProperties	False		

CHAPTER

SIX

COMMAND LINE INTERFACE

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*.

6.1 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.

-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. ':', ';', 'l'). This option can be given multiple times.

-f `<path>`
Execute the given command file. This option can be given multiple times.

-I `<path>`
Add a configuration directory.

-i
Install the format files in the `.lnav/formats/` directory. Individual files will be installed in the `installed` directory and git repositories 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).

-r
Recursively load files from the given base directories.

-t
Prepend timestamps to the lines of data being read in on the standard input.

-w <path>
Write the contents of the standard input to this file.

-V
Print the version of lnav.

-q
Do not print the log messages after executing all of the commands.

6.2 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`.

TZ
The timezone setting is used in some log formats to convert UTC timestamps to the local timezone.

6.3 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 with timestamps prepended:

```
$ make 2>&1 | lnav -t
```

CHAPTER

SEVEN

USER INTERFACE

The main part of the display shows the log messages from all files sorted by the message time. Status bars at the top and bottom of the screen can given you an idea of where you are in the logs. And, the last line is used for entering commands. Navigation is controlled by a series of hotkeys, see *Hotkey Reference* for more information.

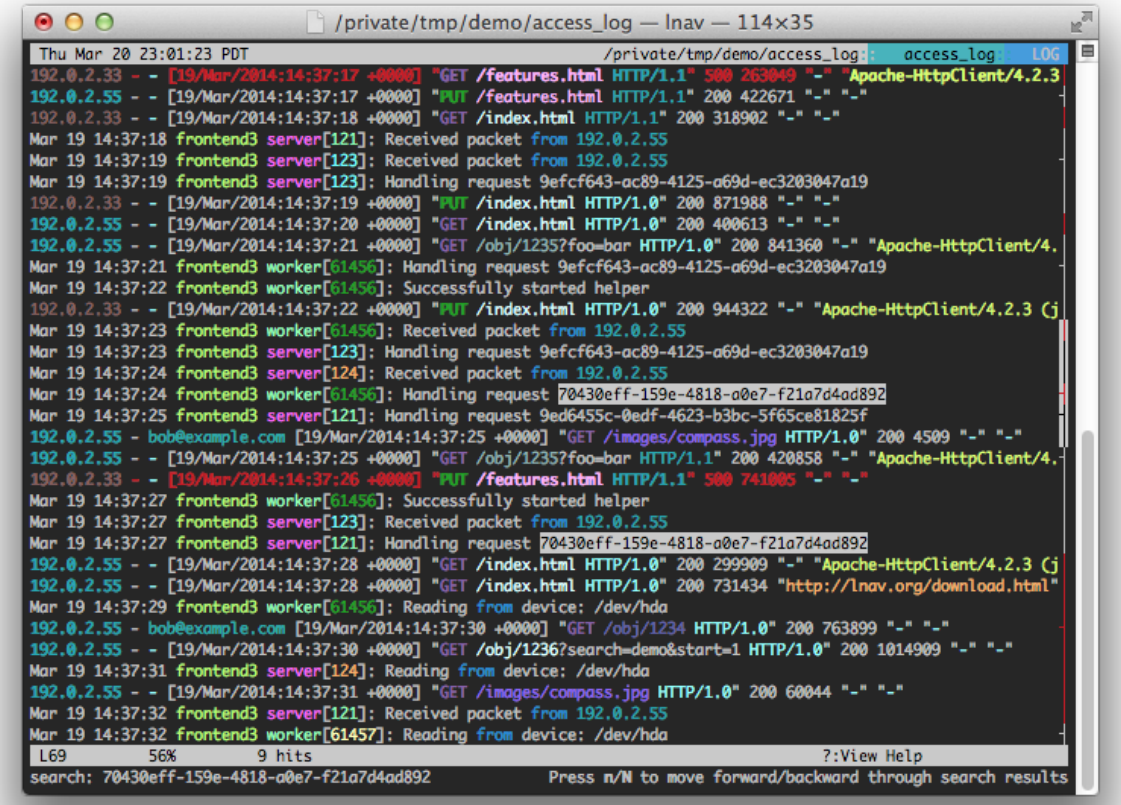

Fig. 1: Screenshot of **lnav** viewing syslog messages.

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).

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

- your current position in the file;
- the locations of errors/warnings in the log files by using a 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.

Above and below the main body are status lines that display:

- the current time;
- the name of the file the top line was pulled from;
- the log format for the top line;
- the current view;
- the line number for the top line in the display;
- the current search hit, the total number of hits, and the search term;

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.

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 command-line is by the readline library, so the usual set of keyboard shortcuts can be used.

The body of the display is also used to display other content, such as: the help file, histograms of the log messages over time, and SQL results. The views are organized into a stack so that any time you activate a new view with a key press or command, the new view is pushed onto the stack. Pressing the same key again will pop the view off of the stack and return you to the previous view. Note that you can always use 'q' to pop the top view off of the stack.

CHAPTER

EIGHT

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.

8.1 Spatial Navigation

Keypress			Command
Space	PgDn		Down a page
b	Backspace	PgUp	Up a page
j	Return	↓	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 source file name for each line. Pressing again will reveal the full path.
Shift + h	Shift + ←		Left ten columns
l	→		Right half a page
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
y	Shift + y		Next/previous SQL result
s	Shift + s		Next/previous slow down in the log message rate
{	}		Previous/next location in history

8.2 Chronological Navigation

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.

8.3 Bookmarks

Keypress	Command
m	Mark/unmark the top line
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

8.4 Display

Keypress	Command
?	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 elapsed time between lines
t	Switch to/from the text file view
i	Switch to/from the histogram view
Shift + i	Switch to/from the histogram view
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. In the SQL result view, cycle through columns to display as bar graphs
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.
=	Pause/unpause loading of new file data.

8.5 Session

Keypress	Command
Ctrl + R	Reset current session.

8.6 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

8.7 Customizing

You can customize the behavior of hotkeys by defining your own keymaps. Consult the *Keymaps* configuration section for more information.

CHAPTER

NINE

LOG 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
VMware vSphere Auto Deploy log format	autodeploy_log	The log format for the VMware Auto Deploy service
Generic Block	block_log	A generic format for logs, like cron, that have a date at the start of a block.
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.
CUPS log format	cups_log	Log format used by the Common Unix Printing System
Dpkg Log	dpkg_log	The debian dpkg log.
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
Common Error Log	error_log	The default web error log format for servers like Apache.
Fsck_hfs Log	fsck_hfs_log	Log for the fsck_hfs tool on Mac OS X.
Glog	glog_log	The google glog format.
HAProxy HTTP Log Format	haproxy_log	The HAProxy log format
Java log format	java_log	Log format used by log4j and output by most java programs
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.
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
CUPS Page Log	page_log	The CUPS server log of printed pages.
Papertrail Service	papertrail_log	Log format for the papertrail log management service
S3 Access Log	s3_log	S3 server access log format
SnapLogic Server Log	snaplogic_log	The SnapLogic server log format.
SSSD log format	sssd_log	Log format used by the System Security Services Daemon

continues on next page

Table 1 – continued from previous page

Name	Table Name	Description
Strace	strace_log	The strace output format.
sudo	sudo_log	The sudo privilege management tool.
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.
Uwsgi Log	uwsgi_log	The uwsgi log format.
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.
RHN server XMLRPC log format	xmlrpc_log	Generated by Satellite's XMLRPC component

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 Extend Log File Format is supported in lnav versions v0.9.1+. The W3C log format is self-describing, so **lnav** will read the header to determine the shape of the file.

9.1 Defining a New Format

New log formats can be defined by placing JSON configuration files in subdirectories of the `~/.lnav/formats/` directory. 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.

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

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

Each format to be defined in the file should a separate field in the top-level object. The field name should be the symbolic name of the format. 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 log file. Log files that contain JSON messages should not specify this field.

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

module-format If true, this regex will only be used to parse message bodies for formats that can act as containers, such as syslog. Default: false.

json True if each log line is JSON-encoded.

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 tranform each log message object into a string that contains the timestamp, followed by a space, and then the message body:

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

field The name 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.

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+)

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. '..').

(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

timestamp-field The name of the field that contains the log message timestamp. 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, %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).

%i Milliseconds from the epoch.

%6 Microseconds 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.

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".

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.

module-field The name of the field that contains the module identifier that distinguishes messages from one log source from another. This field should be used if this message format can act as a container for other types of log messages. For example, an Apache access log can be sent to syslog instead of written to a file. In this case, **lnav** will parse the syslog message and then separately parse the body of the message to determine the "sub" format. This module identifier is used to help **lnav** quickly identify the format to use when parsing message bodies.

hide-extra A boolean for JSON logs that indicates whether fields not present in the line-format should be displayed on their own lines.

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.

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**.

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) || ') '
```

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.

Example format:

```
{
    "$schema": "https://lnav.org/schemas/format-v1.schema.json",
    "example_log" : {
        "title" : "Example Log Format",
```

(continues on next page)

(continued from previous page)

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

9.2 Modifying 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!"
            }
        ]
    }
}
```

9.3 Scripts

Format directories may also contain '.sql' and '.lnav' files to help automate log file analysis. 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 (|). 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. 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.

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.
```

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}
```

9.4 Installing Formats

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.

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
```

9.5 Format Order When Scanning a File

When **lnav** loads a file, it tries each log format against the first ~1000 lines 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
```

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.
- Any active log filters or highlights.
- Hidden files.

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:

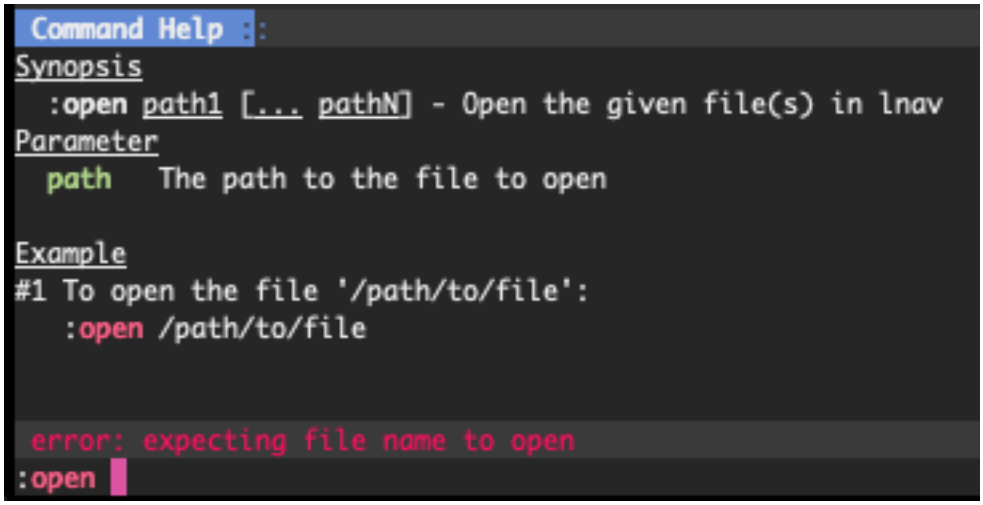

Fig. 1: Screenshot of the online help for the `:open` command.

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 PCRE 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 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

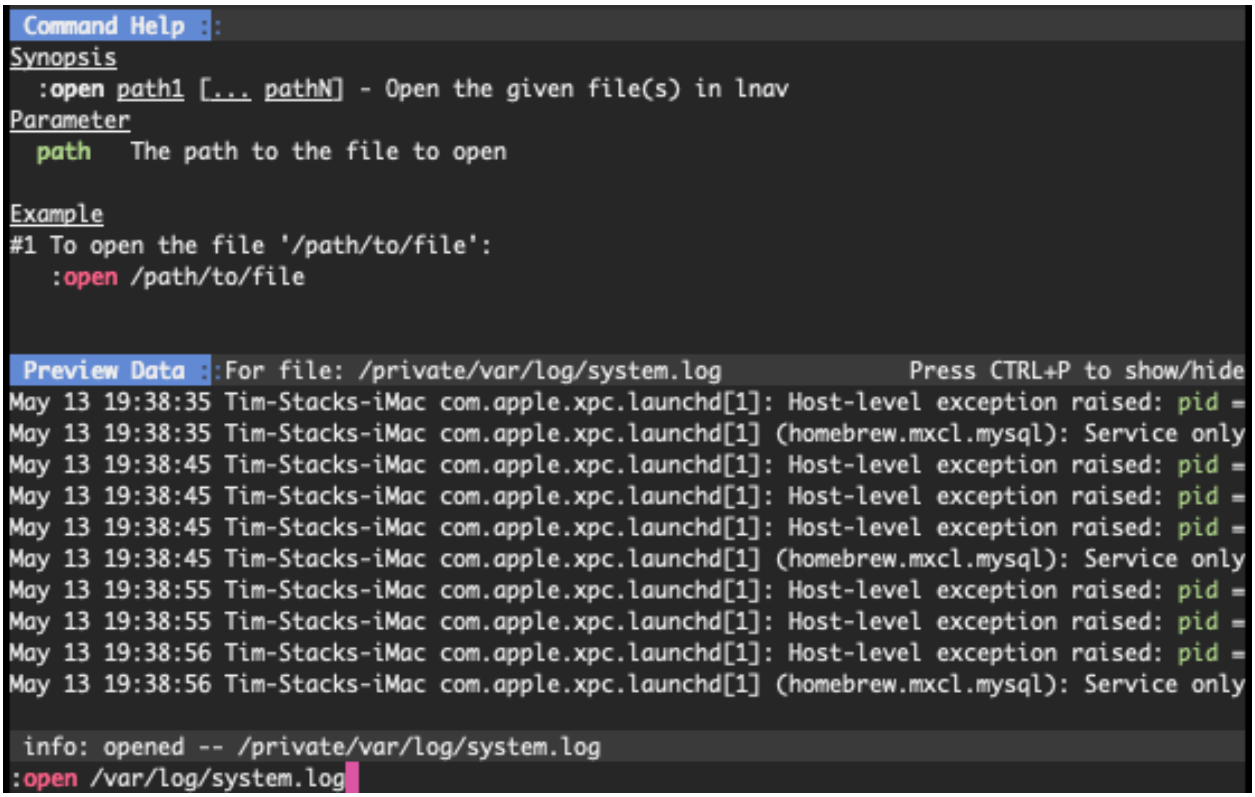

Fig. 2: Screenshot of the preview shown for the `:open` command.

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:

- `:open`
- `:pipe-to`
- `:pipe-line-to`
- `:write-*-to`

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

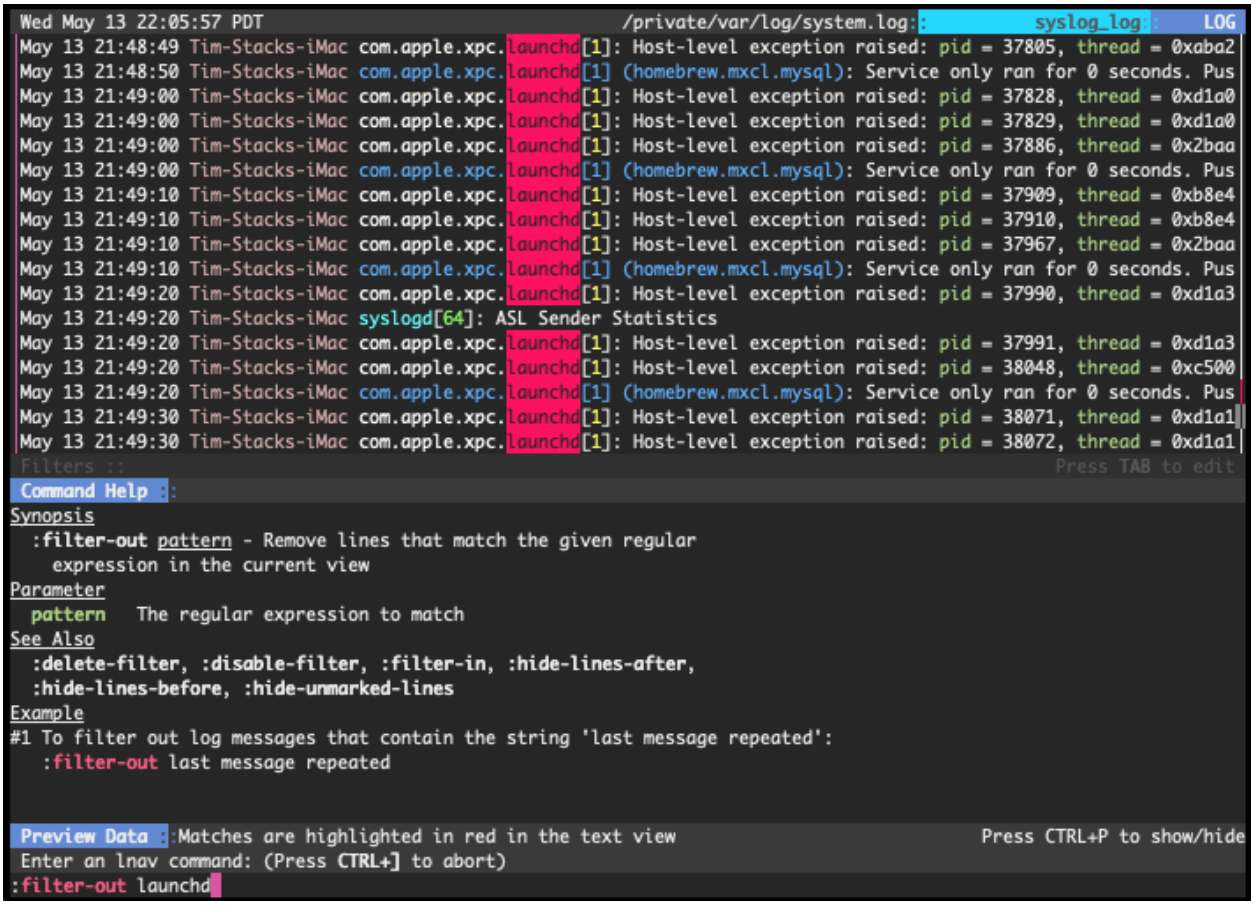

Fig. 3: Screenshot showing the preview for the `:filter-out launchd` command.

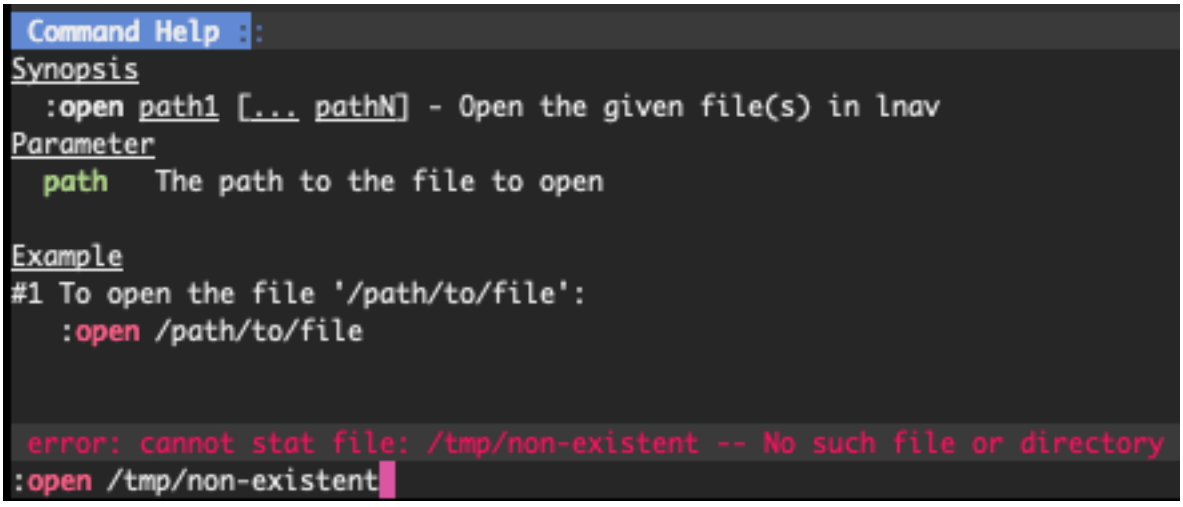

Fig. 4: Screenshot of the error shown when trying to open a non-existent file.

11.1 Reference

11.1.1 :adjust-log-time *timestamp*

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

Parameters

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

Examples To set the top timestamp to a given date:

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

To set the top timestamp back an hour:

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

11.1.2 :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 *:echo msg*, *:eval command*, *:redirect-to [path]*, *:write-csv-to path*, *:write-json-to path*, *:write-jsonlines-to path*, *:write-raw-to path*, *:write-screen-to path*, *:write-table-to path*, *:write-to path*

11.1.3 :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 *:echo msg*, *:pipe-line-to shell-cmd*, *:pipe-to shell-cmd*, *:redirect-to [path]*, *:write-csv-to path*, *:write-json-to path*, *:write-jsonlines-to path*, *:write-raw-to path*, *:write-screen-to path*, *:write-table-to path*, *:write-to path*

11.1.4 :clear-comment

Clear the comment attached to the top log line

See Also *:comment text*, *:tag tag*

11.1.5 :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*

11.1.6 :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 *:enable-word-wrap*, *:hide-fields field-name*, *:highlight pattern*

11.1.7 :clear-partition

Clear the partition the top line is a part of

11.1.8 :close

Close the top file in the view

11.1.9 :comment *text*

Attach a comment to the top log line

Parameters

- **text*** — The comment text

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

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

See Also *:clear-comment*, *:tag tag*

11.1.10 :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*

11.1.11 :create-logline-table *table-name*

Create an SQL table using the top 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 path*, *:write-json-to path*, *:write-jsonlines-to path*, *:write-raw-to path*, *:write-screen-to path*, *:write-table-to path*

11.1.12 :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 path*, *:write-json-to path*, *:write-jsonlines-to path*, *:write-raw-to path*, *:write-screen-to path*, *:write-table-to path*

11.1.13 :current-time

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

---

11.1.14 :delete-filter *pattern*

Delete the filter created with [1m:filter-in[0m or [1m:filter-out[0m

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*

11.1.15 :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 path*, *:write-json-to path*, *:write-jsonlines-to path*, *:write-raw-to path*, *:write-screen-to path*, *:write-table-to path*

11.1.16 :delete-search-table *table-name*

Create an SQL table based on a regex search

Parameters

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

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 path*, *:write-json-to path*, *:write-jsonlines-to path*, *:write-raw-to path*, *:write-screen-to path*, *:write-table-to path*

11.1.17 :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 *:comment text*, *:tag tag*

11.1.18 :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*

11.1.19 :disable-word-wrap

Disable word-wrapping for the current view

See Also *:enable-word-wrap*, *:hide-fields field-name*, *:highlight pattern*

11.1.20 :echo *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

- **msg*** — The message to display

Examples To output 'Hello, World!':

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

See Also *:alt-msg msg*, *:append-to path*, *:eval command*, *:pipe-line-to shell-cmd*, *:pipe-to shell-cmd*, *:redirect-to [path]*, *:redirect-to [path]*, *:write-csv-to path*, *:write-csv-to path*, *:write-json-to path*, *:write-json-to path*, *:write-jsonlines-to path*, *:write-jsonlines-to path*, *:write-raw-to path*, *:write-raw-to path*, *:write-screen-to path*, *:write-screen-to path*, *:write-table-to path*, *:write-table-to path*, *:write-to path*, *:write-to path*

11.1.21 :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*

11.1.22 :enable-word-wrap

Enable word-wrapping for the current view

See Also *:disable-word-wrap*, *:hide-fields field-name*, *:highlight pattern*

11.1.23 :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*, *:echo msg*, *:redirect-to [path]*, *:write-csv-to path*, *:write-json-to path*, *:write-jsonlines-to path*, *:write-raw-to path*, *:write-screen-to path*, *:write-table-to path*, *:write-to path*

11.1.24 :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*

11.1.25 :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*

11.1.26 :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*

11.1.27 :goto *line#|N%|date*

Go to the given location in the top view

Parameters

- **line#|N%|date*** — A line number, percent into the file, or a timestamp

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
```

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

11.1.28 :help

Open the help text view

11.1.29 :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 top 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 *:enable-word-wrap*, *:highlight pattern*, *:show-fields field-name*

11.1.30 :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

---

11.1.31 :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 top 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*

11.1.32 :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 top 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*

11.1.33 :hide-unmarked-lines

Hide lines that have not been bookmarked

See Also *:filter-in pattern*, *:filter-out pattern*, *:hide-lines-after date*, *:hide-lines-before date*, *:mark*, *:next-mark type*, *:prev-mark type*, *:toggle-filtering*

11.1.34 :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-highlight pattern*, *:enable-word-wrap*, *:hide-fields field-name*

11.1.35 :load-session

Load the latest session state

11.1.36 :mark

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

See Also *:hide-unmarked-lines*, *:next-mark type*, *:prev-mark type*

11.1.37 :next-location

Move to the next position in the location history

See Also *:goto line#|N%|date*, *:next-mark type*, *:prev-location*, *:prev-mark type*, *:relative-goto line-count|N%*

11.1.38 :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 *:goto line#\|N%\|date*, *:hide-unmarked-lines*, *:mark*, *:next-location*, *:prev-location*, *:prev-mark type*, *:prev-mark type*, *:relative-goto line-count\|N%*

11.1.39 :open *path*

Open the given file(s) or URLs in lnav

Parameters

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

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

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

11.1.40 :partition-name *name*

Mark the top 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 top line as the start of the partition named 'boot #1':

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

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

Pipe the top line to the given shell command

Parameters

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

Examples To write the top line to 'sed' for processing:

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

See Also *:append-to path*, *:echo msg*, *:pipe-to shell-cmd*, *:redirect-to [path]*, *:write-csv-to path*, *:write-json-to path*, *:write-jsonlines-to path*, *:write-raw-to path*, *:write-screen-to path*, *:write-table-to path*, *:write-to path*

11.1.42 :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*, *:echo msg*, *:pipe-line-to shell-cmd*, *:redirect-to [path]*, *:write-csv-to path*, *:write-json-to path*, *:write-jsonlines-to path*, *:write-raw-to path*, *:write-screen-to path*, *:write-table-to path*, *:write-to path*

11.1.43 :prev-location

Move to the previous position in the location history

See Also *:goto line#\N%\date*, *:next-location*, *:next-mark type*, *:prev-mark type*, *:relative-goto line-count\N%*

11.1.44 :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 *:goto line#\N%\date*, *:hide-unmarked-lines*, *:mark*, *:next-location*, *:next-mark type*, *:next-mark type*, *:prev-location*, *:relative-goto line-count\N%*

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

Open the given prompt

Parameters

- **type*** — The type of prompt – command, script, search, sql, user
- **–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? '
```

11.1.46 :pt-max-time

(null)

11.1.47 :pt-min-time

(null)

11.1.48 :quit

Quit lnav

11.1.49 :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*, *:echo msg*, *:echo msg*, *:eval command*, *:pipe-line-to shell-cmd*, *:pipe-to shell-cmd*, *:write-csv-to path*, *:write-csv-to path*, *:write-json-to path*, *:write-json-to path*, *:write-jsonlines-to path*, *:write-jsonlines-to path*, *:write-raw-to path*, *:write-raw-to path*, *:write-screen-to path*, *:write-screen-to path*, *:write-table-to path*, *:write-table-to path*, *:write-to path*, *:write-to path*

11.1.50 :redraw

Do a full redraw of the screen

11.1.51 :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%|date*, *:next-location*, *:next-mark type*, *:prev-location*, *:prev-mark type*

11.1.52 :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]*

11.1.53 :reset-session

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

---

11.1.54 :save-session

Save the current state as a session

11.1.55 :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
```

11.1.56 :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
```

11.1.57 :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 *:enable-word-wrap*, *:hide-fields field-name*, *:highlight pattern*

11.1.58 :show-file *path*

Show the given file(s) and resume indexing.

Parameters

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

---

11.1.59 :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*

11.1.60 :show-unmarked-lines

Show lines that have not been bookmarked

See Also *: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*

11.1.61 :spectrogram *field-name*

Visualize the given message field 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
```

11.1.62 :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
```

11.1.63 :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
```

11.1.64 :tag *tag*

Attach tags to the top log line

Parameters

- **tag** — The tags to attach

Examples To add the tags '#BUG123' and '#needs-review' to the top line:

```
:tag #BUG123 #needs-review
```

See Also *:comment text*, *:delete-tags tag*, *:untag tag*

11.1.65 :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*

11.1.66 :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
```

11.1.67 :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
```

11.1.68 :untag *tag*

Detach tags from the top log line

Parameters

- **tag** — The tags to detach

Examples To remove the tags '#BUG123' and '#needs-review' from the top line:

```
:untag #BUG123 #needs-review
```

See Also *:comment text*, *:tag tag*

11.1.69 :write-table-to *path*

Write SQL results to the given file in a tabular format

Parameters

- **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*, *:create-logline-table table-name*, *:create-search-table table-name [pattern]*, *:echo msg*, *:echo msg*, *:eval command*, *:pipe-line-to shell-cmd*, *:pipe-to shell-cmd*, *:redirect-to [path]*, *:redirect-to [path]*, *:write-csv-to path*, *:write-csv-to path*, *:write-csv-to path*, *:write-json-to path*, *:write-json-to path*, *:write-json-to path*, *:write-jsonlines-to path*, *:write-jsonlines-to path*, *:write-jsonlines-to path*, *:write-raw-to path*, *:write-raw-to path*, *:write-raw-to path*, *:write-screen-to path*, *:write-screen-to path*, *:write-screen-to path*, *:write-to path*, *:write-to path*

11.1.70 :write-csv-to *path*

Write SQL results to the given file in CSV format

Parameters

- **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*, *:create-logline-table table-name*, *:create-search-table table-name [pattern]*, *:echo msg*, *:echo msg*, *:eval command*, *:pipe-line-to shell-cmd*, *:pipe-to shell-cmd*, *:redirect-to [path]*, *:redirect-to [path]*, *:write-json-to path*, *:write-json-to path*, *:write-json-to path*, *:write-jsonlines-to path*, *:write-jsonlines-to path*, *:write-jsonlines-to path*, *:write-raw-to path*, *:write-raw-to path*, *:write-raw-to path*, *:write-screen-to path*, *:write-screen-to path*, *:write-screen-to path*, *:write-table-to path*, *:write-table-to path*, *:write-table-to path*, *:write-to path*, *:write-to path*

11.1.71 :write-json-to *path*

Write SQL results to the given file in JSON format

Parameters

- **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*, *:create-logline-table table-name*, *:create-search-table table-name [pattern]*, *:echo msg*, *:echo msg*, *:eval command*, *:pipe-line-to shell-cmd*, *:pipe-to shell-cmd*, *:redirect-to [path]*, *:redirect-to [path]*, *:write-csv-to path*, *:write-csv-to path*, *:write-csv-to path*, *:write-jsonlines-to path*, *:write-jsonlines-to path*, *:write-jsonlines-to path*, *:write-raw-to path*, *:write-raw-to path*, *:write-raw-to path*, *:write-screen-to path*, *:write-screen-to path*, *:write-screen-to path*, *:write-table-to path*, *:write-table-to path*, *:write-table-to path*, *:write-to path*, *:write-to path*

11.1.72 :write-jsonlines-to *path*

Write SQL results to the given file in JSON Lines format

Parameters

- **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*, *:create-logline-table table-name*, *:create-search-table table-name [pattern]*, *:echo msg*, *:echo msg*, *:eval command*, *:pipe-line-to shell-cmd*, *:pipe-to shell-cmd*, *:redirect-to [path]*, *:redirect-to [path]*, *:write-csv-to path*, *:write-csv-to path*, *:write-csv-to path*, *:write-json-to path*, *:write-json-to path*, *:write-json-to path*, *:write-raw-to path*, *:write-raw-to path*, *:write-raw-to path*, *:write-screen-to path*, *:write-screen-to path*, *:write-screen-to path*, *:write-table-to path*, *:write-table-to path*, *:write-table-to path*, *:write-to path*, *:write-to path*

11.1.73 :write-raw-to *path*

Write the text in the top view to the given file without any formatting

Parameters

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

Examples To write the top view to /tmp/table.txt:

```
:write-raw-to /tmp/table.txt
```

See Also *:alt-msg msg*, *:append-to path*, *:create-logline-table table-name*, *:create-search-table table-name [pattern]*, *:echo msg*, *:echo msg*, *:eval command*, *:pipe-line-to shell-cmd*, *:pipe-to shell-cmd*, *:redirect-to [path]*, *:redirect-to [path]*, *:write-csv-to path*, *:write-csv-to path*, *:write-csv-to path*, *:write-json-to path*, *:write-json-to path*, *:write-json-to path*, *:write-jsonlines-to path*, *:write-jsonlines-to path*, *:write-jsonlines-to path*, *:write-screen-to path*, *:write-screen-to path*, *:write-screen-to path*, *:write-table-to path*, *:write-table-to path*, *:write-table-to path*, *:write-to path*, *:write-to path*

11.1.74 :write-screen-to *path*

Write the displayed text or SQL results to the given file without any formatting

Parameters

- **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*, *:create-logline-table table-name*, *:create-search-table table-name [pattern]*, *:echo msg*, *:echo msg*, *:eval command*, *:pipe-line-to shell-cmd*, *:pipe-to shell-cmd*, *:redirect-to [path]*, *:redirect-to [path]*, *:write-csv-to path*, *:write-csv-to path*, *:write-csv-to path*, *:write-json-to path*, *:write-json-to path*, *:write-json-to path*, *:write-jsonlines-to path*, *:write-jsonlines-to path*, *:write-jsonlines-to path*, *:write-raw-to path*, *:write-raw-to path*, *:write-raw-to path*, *:write-table-to path*, *:write-table-to path*, *:write-table-to path*, *:write-to path*, *:write-to path*

11.1.75 :write-to *path*

Overwrite the given file with any marked lines in the current view

Parameters

- **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*, *:echo msg*, *:echo msg*, *:eval command*, *:pipe-line-to shell-cmd*, *:pipe-to shell-cmd*, *:redirect-to [path]*, *:redirect-to [path]*, *:write-csv-to path*, *:write-csv-to path*, *:write-json-to path*, *:write-json-to path*, *:write-jsonlines-to path*, *:write-jsonlines-to path*, *:write-raw-to path*, *:write-raw-to path*, *:write-screen-to path*, *:write-screen-to path*, *:write-table-to path*, *:write-table-to path*

11.1.76 :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_line	log_part	log_time	log_idle_msecs	log_level	log_mark	log_comment	log_tags	log_filename	c_ip	cs_method	cs_referer	cs_uri_query	cs_username	cs_user_agent	cs_version	sc_bytes	sc_status
0	<NULL>	2000-10-10 13:55:36.000	0	info	1	<NULL>	<NULL>	<NULL>	127.0.0.1	GET	<NULL>	<NULL>	/apache_pb.gif	<NULL>	frank	HTTP/1.0	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`, and `log_body`.

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.

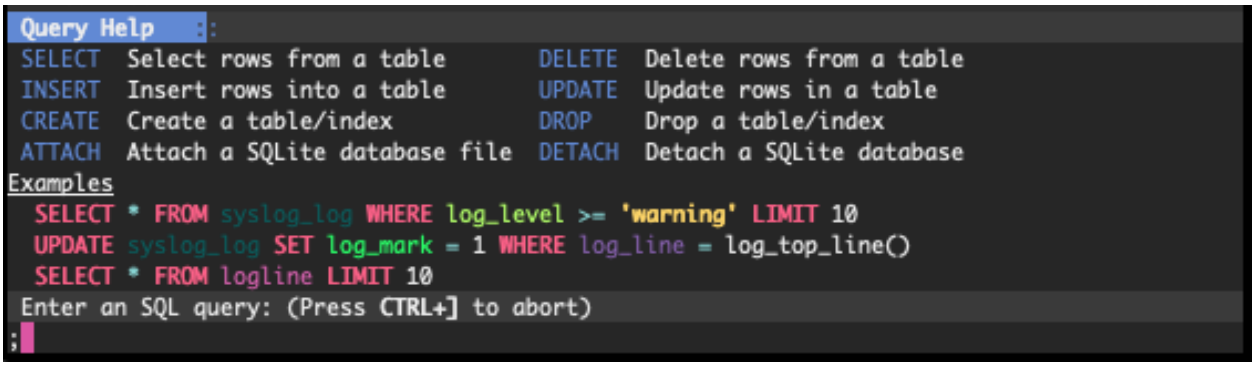

Fig. 1: Screenshot of the online help for the SQL prompt.

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:

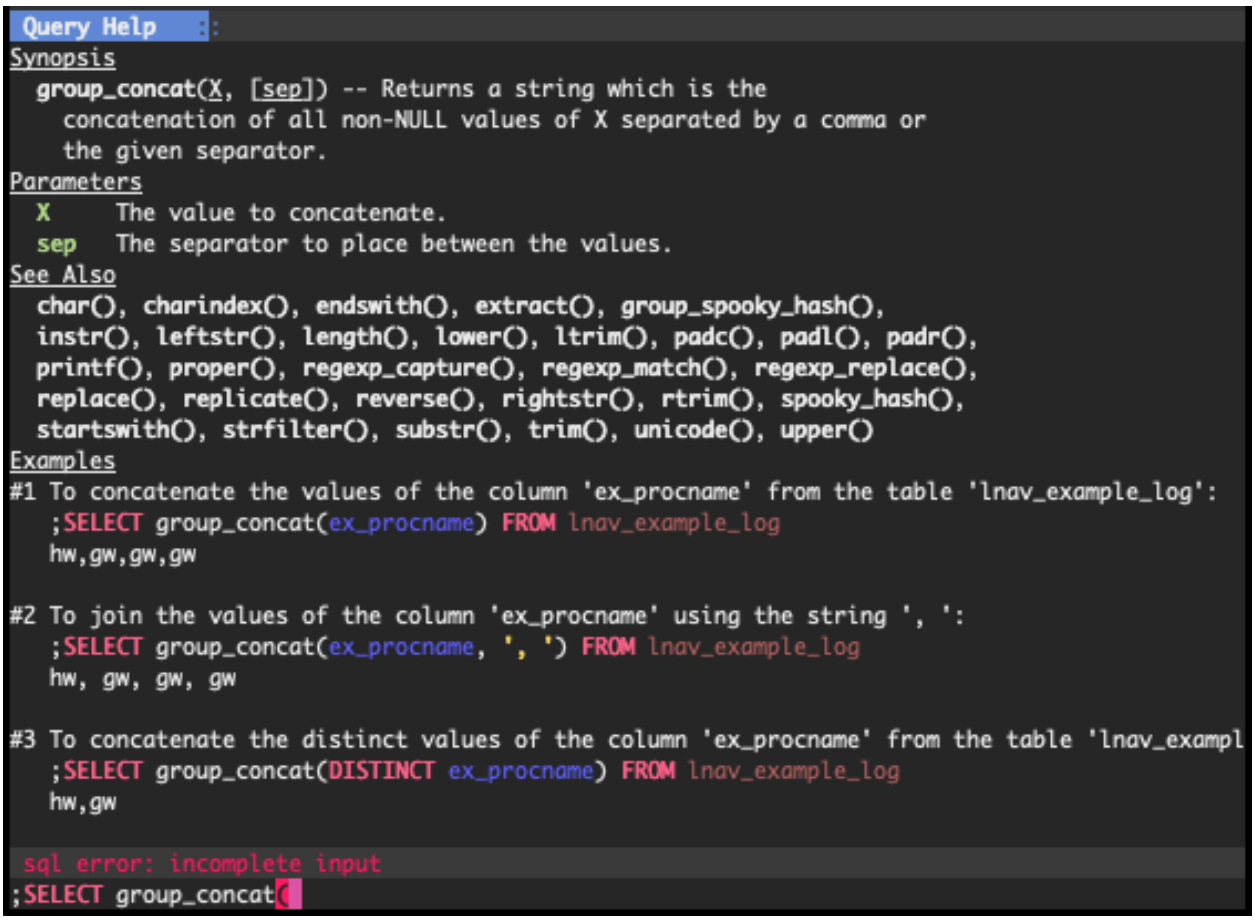

Fig. 2: Screenshot of the online help for the `group_concat()` function.

```
;SELECT c_ip, avg(sc_bytes), max(sc_bytes) FROM access_log GROUP BY c_ip
```

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

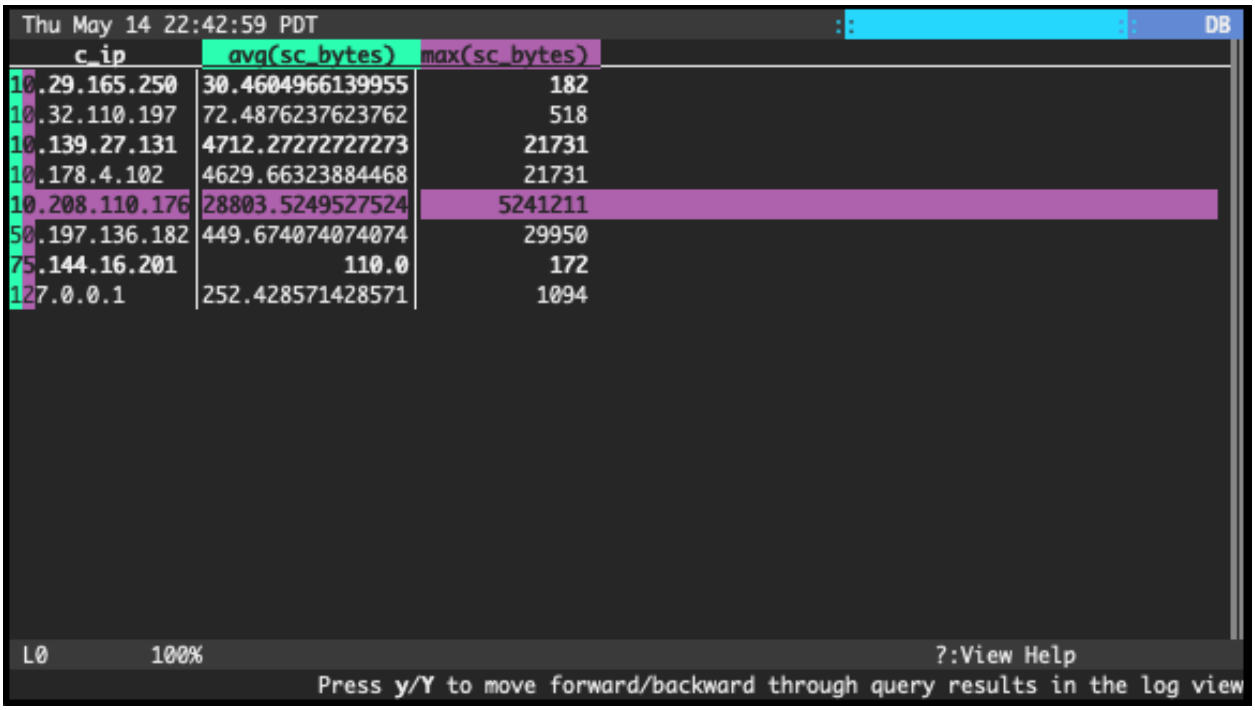

Fig. 3: Screenshot of the SQL results view.

The DB view has the following display features:

- Column headers stick to the top of the view when scrolling.
- A stacked bar chart of the numeric column values is displayed underneath the rows. Pressing `TAB` will cycle through displaying no columns, each individual column, or all columns.
- JSON columns in the top row can be pretty-printed by pressing `p`. The display will show the value and JSON-Pointer path that can be passed to the *jget* function.

12.1 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.

12.2 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.
- .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.

12.3 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.

12.4 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
```

12.5 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.

12.6 Reference

The following is a reference of the SQL syntax and functions that are available:

12.6.1 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
```

12.6.2 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
```

12.6.3 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

---

12.6.4 CREATE *[TEMP]* TABLE *[IF NOT EXISTS] [schema-name.] table-name* AS *select-stmt*

Create a table

12.6.5 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.

12.6.6 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
```

12.6.7 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
- **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
```

12.6.8 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' < 'a10' COLLATE naturalnocase)
             0                                      1
```

12.6.9 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
```

12.6.10 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.

12.6.11 DROP INDEX *[IF EXISTS] [schema-name.] index-name*

Drop an index

12.6.12 DROP TABLE *[IF EXISTS] [schema-name.] table-name*

Drop a table

12.6.13 DROP VIEW *[IF EXISTS] [schema-name.] view-name*

Drop a view

12.6.14 DROP TRIGGER *[IF EXISTS] [schema-name.] trigger-name*

Drop a trigger

12.6.15 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
```

12.6.16 expr *[NOT]* LIKE *pattern*

Match an expression against a text pattern.

Parameters

- **pattern*** — The pattern to match against.

Examples To check if a value matches the pattern 'Hello, %!':

```
;SELECT 'Hello, World!' LIKE 'Hello, %!'
1
```

12.6.17 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
```

12.6.18 SELECT *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

- **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 'syslog_log':

```
;SELECT * FROM syslog_log
```

12.6.19 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!')
```

12.6.20 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

---

12.6.21 OVER *window-name*

Executes the preceding function over a window

Parameters

- **window-name*** — The name of the window definition

12.6.22 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.
- **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
```

12.6.23 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)*

12.6.24 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 acos(0.2)
1.36943840600457
```

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)*

12.6.25 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)
0.622362503714779
```

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)*

12.6.26 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)
0.201357920790331
```

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)*

12.6.27 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)
0.198690110349241
```

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)*

12.6.28 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)
0.197395559849881
```

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)*

12.6.29 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))
45.0
```

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)*

12.6.30 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)
0.202732554054082
```

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)*

12.6.31 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))
45.0
```

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)*

12.6.32 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_duration)
gw                        5.0
hw                        2.0
```

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)*

12.6.33 basename(*path*)

Extract the base portion of a pathname.

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('/')
/
```

See Also *dirname(path)*, *joinpath(path)*, *readlink(path)*, *realpath(path)*

12.6.34 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)*

12.6.35 changes()

The number of database rows that were changed, inserted, or deleted by the most recent statement.

12.6.36 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 *charindex(needle, haystack, [start])*, *endswith(str, suffix)*, *extract(str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *humanize_file_size(value)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *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])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *trim(str, [chars])*, *unicode(X)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.6.37 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 *char(X)*, *endswith(str, suffix)*, *extract(str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *humanize_file_size(value)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *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])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *trim(str, [chars])*, *unicode(X)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.6.38 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
```

12.6.39 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
```

12.6.40 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()*

12.6.41 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)*, *julianday(timestring, modifier)*, *strftime(format, timestring, modifier)*, *time(timestring, modifier)*, *timediff(time1, time2)*, *timeslice(time, slice)*

12.6.42 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)*, *julianday(timestring, modifier)*, *strftime(format, timestring, modifier)*, *time(timestring, modifier)*, *timediff(time1, time2)*, *timeslice(time, slice)*

12.6.43 degrees(*radians*)

Converts radians to degrees

Parameters

- **radians*** — The radians value to convert to degrees

Examples To convert PI to degrees:

```
;SELECT degrees(pi())
180.0
```

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)*

12.6.44 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()*

12.6.45 dirname(*path*)

Extract the directory portion of a pathname.

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)*

12.6.46 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 *char(X)*, *charindex(needle, haystack, [start])*, *extract(str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *humanize_file_size(value)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *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])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *trim(str, [chars])*, *unicode(X)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.6.47 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)*

12.6.48 extract(*str*)

Automatically Parse and extract data from a string

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 *char(X)*, *charindex(needle, haystack, [start])*, *endswith(str, suffix)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *humanize_file_size(value)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *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])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *trim(str, [chars])*, *unicode(X)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.6.49 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()*

12.6.50 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)*

12.6.51 gethostbyaddr(*hostname*)

Get the hostname for the given IP address

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)*

12.6.52 gethostbyname(*hostname*)

Get the IP address for the given hostname

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)*

12.6.53 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
```

12.6.54 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 *char(X)*, *charindex(needle, haystack, [start])*, *endswith(str, suffix)*, *extract(str)*, *group_spooky_hash(str)*, *humanize_file_size(value)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *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])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *trim(str, [chars])*, *unicode(X)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.6.55 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 *char(X)*, *charindex(needle, haystack, [start])*, *endswith(str, suffix)*, *extract(str)*, *group_concat(X, [sep])*, *humanize_file_size(value)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *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])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *trim(str, [chars])*, *unicode(X)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.6.56 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
```

12.6.57 humanize_file_size(*value*)

Format the given file size as a human-friendly string

Parameters

- **value*** — The file size to format

Examples To format an amount:

```
;SELECT humanize_file_size(10 * 1024 * 1024)
10.0MB
```

See Also *char(X)*, *charindex(needle, haystack, [start])*, *endswith(str, suffix)*, *extract(str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *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])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *trim(str, [chars])*, *unicode(X)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.6.58 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
```

12.6.59 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 *char(X)*, *charindex(needle, haystack, [start])*, *endswith(str, suffix)*, *extract(str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *humanize_file_size(value)*, *leftstr(str, N)*, *length(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *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])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *trim(str, [chars])*, *unicode(X)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.6.60 jget(*json*, *ptr*, *[default]*)

Get the value from a JSON object using a JSON-Pointer.

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_concat(json, value)*, *json_contains(json, value)*, *json_group_array(value)*, *json_group_object(name, value)*

12.6.61 joinpath(*path*)

Join components of a path together.

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)*

12.6.62 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.

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_contains(json, value)*, *json_group_array(value)*, *json_group_object(name, value)*

12.6.63 json_contains(*json*, *value*)

Check if a JSON value contains the given element.

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_concat(json, value)*, *json_group_array(value)*, *json_group_object(name, value)*

12.6.64 json_group_array(*value*)

Collect the given values from a query into a JSON 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_concat(json, value)*, *json_contains(json, value)*, *json_group_object(name, value)*

12.6.65 json_group_object(*name*, *value*)

Collect the given values from a query into a JSON 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_concat(json, value)*, *json_contains(json, value)*, *json_group_array(value)*

12.6.66 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')
2457755.62783565
```

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')
2457755.62853009
```

To get the julian day from the timestamp 1491341842:

```
;SELECT julianday(1491341842, 'unixepoch')
2457848.40094907
```

See Also *date(timestring, modifier)*, *datetime(timestring, modifier)*, *strftime(format, timestring, modifier)*, *time(timestring, modifier)*, *timediff(time1, time2)*, *timeslice(time, slice)*

12.6.67 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()*

12.6.68 last_insert_rowid()

Returns the ROWID of the last row insert from the database connection which invoked the function

---

12.6.69 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()*

12.6.70 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()*

12.6.71 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 *char(X)*, *charindex(needle, haystack, [start])*, *endswith(str, suffix)*, *extract(str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *humanize_file_size(value)*, *instr(haystack, needle)*, *length(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *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])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *trim(str, [chars])*, *unicode(X)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.6.72 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 *char(X)*, *charindex(needle, haystack, [start])*, *endswith(str, suffix)*, *extract(str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *humanize_file_size(value)*, *instr(haystack, needle)*, *leftstr(str, N)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *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])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *trim(str, [chars])*, *unicode(X)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.6.73 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
```

12.6.74 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

12.6.75 likely(*value*)

Short-hand for likelihood(X,0.9375)

Parameters

- **value*** — The boolean value to return

12.6.76 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.

12.6.77 log(*x*)

Returns the natural logarithm of x

Parameters

- **x*** — The number

Examples To get the natual logarithm of 8:

```
;SELECT log(8)
2.07944154167984
```

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)*

12.6.78 log10(*x*)

Returns the base-10 logarithm of X

Parameters

- **x*** — The number

Examples To get the logarithm of 100:

```
;SELECT log10(100)
2.0
```

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)*

12.6.79 log_top_datetime()

Return the timestamp of the line at the top of the log view.

12.6.80 log_top_line()

Return the line number at the top of the log view.

12.6.81 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 *char(X)*, *charindex(needle, haystack, [start])*, *endswith(str, suffix)*, *extract(str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *humanize_file_size(value)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *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])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *trim(str, [chars])*, *unicode(X)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.6.82 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 whitespace 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 *char(X)*, *charindex(needle, haystack, [start])*, *endswith(str, suffix)*, *extract(str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *humanize_file_size(value)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *lower(str)*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *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])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *trim(str, [chars])*, *unicode(X)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.6.83 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)*

12.6.84 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)*

12.6.85 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()*

12.6.86 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()*

12.6.87 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
```

12.6.88 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 *char(X)*, *charindex(needle, haystack, [start])*, *endswith(str, suffix)*, *extract(str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *humanize_file_size(value)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *lower(str)*, *ltrim(str, [chars])*, *padl(str, len)*, *padr(str, len)*, *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])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *trim(str, [chars])*, *unicode(X)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.6.89 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 *char(X)*, *charindex(needle, haystack, [start])*, *endswith(str, suffix)*, *extract(str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *humanize_file_size(value)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padr(str, len)*, *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])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *trim(str, [chars])*, *unicode(X)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.6.90 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 *char(X)*, *charindex(needle, haystack, [start])*, *endswith(str, suffix)*, *extract(str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *humanize_file_size(value)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *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])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *trim(str, [chars])*, *unicode(X)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.6.91 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()*

12.6.92 pi()

Returns the value of PI

Examples To get the value of PI:

```
;SELECT pi()
3.14159265358979
```

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)*

12.6.93 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)
8.0
```

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)*

12.6.94 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 *char(X)*, *charindex(needle, haystack, [start])*, *endswith(str, suffix)*, *extract(str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *humanize_file_size(value)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *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])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *trim(str, [chars])*, *unicode(X)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.6.95 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 *char(X)*, *charindex(needle, haystack, [start])*, *endswith(str, suffix)*, *extract(str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *humanize_file_size(value)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *printf(format, X)*, *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])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *trim(str, [chars])*, *unicode(X)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.6.96 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'
```

12.6.97 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)
3.14159265358979
```

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)*

12.6.98 raise_error(*msg*)

Raises an error with the given message when executed

Parameters

- **msg*** — The error message

12.6.99 random()

Returns a pseudo-random integer between -9223372036854775808 and +9223372036854775807.

12.6.100 randomblob(*N*)

Return an N-byte blob containing pseudo-random bytes.

Parameters

- **N*** — The size of the blob in bytes.

12.6.101 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()*

12.6.102 readlink(*path*)

Read the target of a symbolic link.

Parameters

- **path*** — The path to the symbolic link.

See Also *basename(path)*, *dirname(path)*, *joinpath(path)*, *realpath(path)*

12.6.103 realpath(*path*)

Returns the resolved version of the given path, expanding symbolic links and resolving '.' and '..' references.

Parameters

- **path*** — The path to resolve.

See Also *basename(path)*, *dirname(path)*, *joinpath(path)*, *readlink(path)*

12.6.104 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

---

12.6.105 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
↪ 4 a=1
          0             1                           3           1
↪ 2 a
          0             2                           3           3
↪ 4 1
          1             0 <NULL>                    3           6
↪ 9 b=2
          1             1                           3           6
↪ 7 b
          1             2                           3           8
↪ 9 2
```

See Also *char(X)*, *charindex(needle, haystack, [start])*, *endswith(str, suffix)*, *extract(str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *humanize_file_size(value)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *printf(format, X)*, *proper(str)*, *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])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *trim(str, [chars])*, *unicode(X)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.6.106 regexp_match(*re*, *str*)

Match a string against a regular expression and return the capture groups as JSON.

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 *char(X)*, *charindex(needle, haystack, [start])*, *endswith(str, suffix)*, *extract(str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *humanize_file_size(value)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *printf(format, X)*, *proper(str)*, *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])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *trim(str, [chars])*, *unicode(X)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.6.107 regexp_replace(*str*, *re*, *repl*)

Replace the parts of a string that match a regular expression.

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 *char(X)*, *charindex(needle, haystack, [start])*, *endswith(str, suffix)*, *extract(str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *humanize_file_size(value)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *printf(format, X)*, *proper(str)*, *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])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *trim(str, [chars])*, *unicode(X)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.6.108 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 *char(X)*, *charindex(needle, haystack, [start])*, *endswith(str, suffix)*, *extract(str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *humanize_file_size(value)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *printf(format, X)*, *proper(str)*, *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])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *trim(str, [chars])*, *unicode(X)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.6.109 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 *char(X)*, *charindex(needle, haystack, [start])*, *endswith(str, suffix)*, *extract(str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *humanize_file_size(value)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *printf(format, X)*, *proper(str)*, *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])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *trim(str, [chars])*, *unicode(X)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.6.110 reverse(*str*)

Returns the reverse of the given string.

Parameters

- **str*** — The string to reverse.

Examples To reverse the string 'abc':

```
;SELECT reverse('abc')
cba
```

See Also *char(X)*, *charindex(needle, haystack, [start])*, *endswith(str, suffix)*, *extract(str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *humanize_file_size(value)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *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)*, *rightstr(str, N)*, *rtrim(str, [chars])*, *sparkline(value, [upper])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *trim(str, [chars])*, *unicode(X)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.6.111 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 *char(X)*, *charindex(needle, haystack, [start])*, *endswith(str, suffix)*, *extract(str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *humanize_file_size(value)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *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)*, *rtrim(str, [chars])*, *sparkline(value, [upper])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *trim(str, [chars])*, *unicode(X)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.6.112 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)
123.0
```

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)*

12.6.113 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()*

12.6.114 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 whitespace 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 *char(X)*, *charindex(needle, haystack, [start])*, *endswith(str, suffix)*, *extract(str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *humanize_file_size(value)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *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)*, *sparkline(value, [upper])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *trim(str, [chars])*, *unicode(X)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.6.115 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)*

12.6.116 sparkline(*value, [upper]*)

Converts a numeric value on a range to a bar chart character

Parameters

- **value*** — The numeric value to convert
- **upper** — The upper bound of the numeric range (default: 100)

Examples To get the unicode block element for the value 32 in the range of 0-128:

```
;SELECT sparkline(32, 128)
```

See Also *char(X)*, *charindex(needle, haystack, [start])*, *endswith(str, suffix)*, *extract(str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *humanize_file_size(value)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *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])*, *trim(str, [chars])*, *unicode(X)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.6.117 sparkline(*value, [upper]*)

An aggregate function to convert numeric values to a sparkline bar chart

Parameters

- **value*** — The numeric values to chart
- **upper** — The upper bound of the numeric range. If not provided, the default is derived from all of the provided values

Examples 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 *char(X)*, *charindex(needle, haystack, [start])*, *endswith(str, suffix)*, *extract(str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *humanize_file_size(value)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *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])*, *trim(str, [chars])*, *unicode(X)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.6.118 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 *char(X)*, *charindex(needle, haystack, [start])*, *endswith(str, suffix)*, *extract(str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *humanize_file_size(value)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *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])*, *sparkline(value, [upper])*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *trim(str, [chars])*, *unicode(X)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.6.119 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

12.6.120 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
```

12.6.121 sqlite_source_id()

Returns a string that identifies the specific version of the source code that was used to build the SQLite library.

12.6.122 sqlite_version()

Returns the version string for the SQLite library that is running.

12.6.123 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)*

12.6.124 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 *char(X)*, *charindex(needle, haystack, [start])*, *endswith(str, suffix)*, *extract(str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *humanize_file_size(value)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *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])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *strfilter(source, include)*, *substr(str, start, [size])*, *trim(str, [chars])*, *unicode(X)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.6.125 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 *char(X)*, *charindex(needle, haystack, [start])*, *endswith(str, suffix)*, *extract(str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *humanize_file_size(value)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *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])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *substr(str, start, [size])*, *trim(str, [chars])*, *unicode(X)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.6.126 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)*, *julianday(timestring, modifier)*, *time(timestring, modifier)*, *timediff(time1, time2)*, *timeslice(time, slice)*

12.6.127 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 *char(X)*, *charindex(needle, haystack, [start])*, *endswith(str, suffix)*, *extract(str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *humanize_file_size(value)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *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])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *trim(str, [chars])*, *unicode(X)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.6.128 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)*

12.6.129 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)*, *julianday(timestring, modifier)*, *strftime(format, timestring, modifier)*, *timediff(time1, time2)*, *timeslice(time, slice)*

12.6.130 timediff(*time1*, *time2*)

Compute the difference between two timestamps in seconds

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')
6.0
```

To get the difference between relative timestamps:

```
;SELECT timediff('today', 'yesterday')
86400.0
```

See Also *date(timestring, modifier)*, *datetime(timestring, modifier)*, *julianday(timestring, modifier)*, *strftime(format, timestring, modifier)*, *time(timestring, modifier)*, *timeslice(time, slice)*

12.6.131 timeslice(*time*, *slice*)

Return the start of the slice of time that the given timestamp falls in.

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 05:00:00.000
```

To group log messages into five minute buckets and count them:

```
;SELECT timeslice(log_time_msecs, '5m') AS slice, count(*) FROM lnav_
↪example_log GROUP BY slice
2017-01-01 05:00:00.000
```

See Also *date(timestring, modifier)*, *datetime(timestring, modifier)*, *julianday(timestring, modifier)*, *strftime(format, timestring, modifier)*, *time(timestring, modifier)*, *timediff(time1, time2)*

12.6.132 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
17.0
```

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)*

12.6.133 total_changes()

Returns the number of row changes caused by INSERT, UPDATE or DELETE statements since the current database connection was opened.

12.6.134 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 whitespace 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 *char(X)*, *charindex(needle, haystack, [start])*, *endswith(str, suffix)*, *extract(str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *humanize_file_size(value)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *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])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *unicode(X)*, *upper(str)*, *xpath(xpath, xmldoc)*

12.6.135 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
```

12.6.136 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 *char(X)*, *charindex(needle, haystack, [start])*, *endswith(str, suffix)*, *extract(str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *humanize_file_size(value)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *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])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *trim(str, [chars])*, *upper(str)*, *xpath(xpath, xmldoc)*

12.6.137 unlikely(*value*)

Short-hand for likelihood(X, 0.0625)

Parameters

- **value*** — The boolean value to return

12.6.138 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 *char(X)*, *charindex(needle, haystack, [start])*, *endswith(str, suffix)*, *extract(str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *humanize_file_size(value)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *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])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *trim(str, [chars])*, *unicode(X)*, *xpath(xpath, xmldoc)*

12.6.139 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="b">Hello</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 *char(X)*, *charindex(needle, haystack, [start])*, *endswith(str, suffix)*, *extract(str)*, *group_concat(X, [sep])*, *group_spooky_hash(str)*, *humanize_file_size(value)*, *instr(haystack, needle)*, *leftstr(str, N)*, *length(str)*, *lower(str)*, *ltrim(str, [chars])*, *padc(str, len)*, *padl(str, len)*, *padr(str, len)*, *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])*, *sparkline(value, [upper])*, *spooky_hash(str)*, *startswith(str, prefix)*, *strfilter(source, include)*, *substr(str, start, [size])*, *trim(str, [chars])*, *unicode(X)*, *upper(str)*

12.6.140 zeroblob(*N*)

Returns a BLOB consisting of N bytes of 0x00.

Parameters

- **N*** — The size of the BLOB.

CHAPTER

THIRTEEN

SQLITE TABLES REFERENCE

In addition to the tables generated for each log format, **lnav** includes the following tables/views:

- *environ*
- *lnav_file*
- *lnav_views*
- *lnav_view_stack*
- *lnav_view_filters*
- *lnav_view_filter_stats*
- *lnav_view_filters_and_stats*
- *all_logs*
- *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
```

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 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.

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.

13.3 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.

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.

13.4 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.

13.5 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.

13.6 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.

13.7 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.

13.8 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.

13.9 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.

13.10 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

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 messsage:

```
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
├ log_pid      = 2467
```

(continues on next page)

(continued from previous page)

```
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
     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
```

(continues on next page)

(continued from previous page)

```
    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
```

14.1 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

FIFTEEN

FREQUENTLY ASKED QUESTIONS

15.1 Q: How can I copy & paste without decorations?

Answer There are a few ways to do this:

- Use the *bookmark* hotkeys to mark lines and then press `c` to copy to the local system keyboard.
- 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.

15.2 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 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.

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.

15.3 Q: Why isn't my log file highlighted correctly?

TBD

15.4 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.

CHAPTER

SIXTEEN

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

Symbols

C

E

X