Inav Documentation

Release 0.8.5

Tim Stack

Aug 20, 2019

Contents

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

Contents:

CHAPTER 1

Introduction

The Log File Navigator, **lnav**, is an enhanced log file viewer that takes advantage of any semantic information that can be gleaned from the files being viewed, such as timestamps and log levels. Using this extra semantic information, lnav can do things like interleaving messages from different files, generate histograms of messages over time, and providing hotkeys for navigating through the file. It is hoped that these features will allow the user to quickly and efficiently zero in on problems.

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

1.2 Installation

Check the downloads page to see if there are packages for your operating system. Compiling from source is just a matter of doing:

```
$ ./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
```

CHAPTER 2

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.

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;

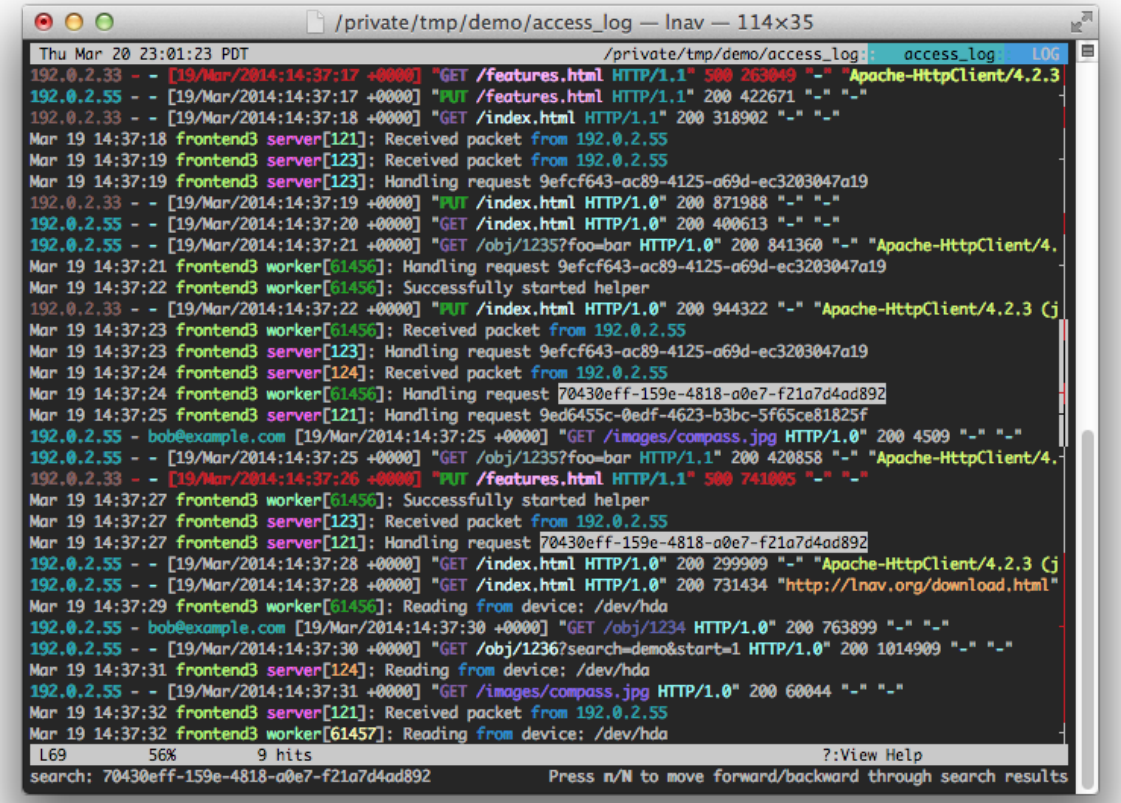

Fig. 1: Screenshot of **lnav** viewing syslog messages.

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 3

Command Line Interface

-h	Print these command-line options and exit.
-H	Start lnav and switch to the help view.
-C	Check the configuration for any errors and exit.
-c	Execute the given command. This option can be given multiple times.
-f	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 file	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

CHAPTER 4

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.

Continued on next page

Table 1 – continued from previous page

Name	Table Name	Description
OpenAM Debug Log	openamdb_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
SnapLogic Server Log	snaplogic_log	The SnapLogic server log format.
SSSD log format	sssd_log	Log format used by the System Security Services Daemon
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

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

4.1 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.

The contents of the format configuration should be a JSON object with a field for each format defined by the file. Each 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:

```
{
    "example_log" : {
        "title" : "Example Log Format",
        "description" : "Log format used in the documentation example.",
        "url" : "http://example.com/log-format.html",
        "regex" : {
            "basic" : {
                "pattern" : "^(?<timestamp>\\d{4}-\\d{2}-\\d{2}T\\d{2}:\\d{2}:\\d{2}
↪\\.\\d{3}Z)>>(?<level>\\w+)>>(?<component>\\w+)>>(?<body>.*)$"
```

(continues on next page)

(continued from previous page)

```
            }
        },
        "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!"
            }
        ]
    }
}
```

4.2 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:

```
{
    "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!"
            }
        ]
    }
}
```

4.3 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. Any arguments passed to a script can be referenced using '$N' where 'N' is the index of the argument. Remember that you need to use the ':eval' command (see *Miscellaneous*) 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.
```

4.4 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
```

4.5 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 5

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 **:cmd:'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
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 Extensions Reference* 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
```

(continues on next page)

(continued from previous page)

```
     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
    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

5.1 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 6

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.

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 7

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.

7.1 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/prevous SQL result
s	Shift + s		Next/prevous slow down in the log message rate
{	}		Previous/next location in history

7.2 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.

7.3 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

7.4 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	Cycle through colums to graph in the SQL result view
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.

7.5 Session

Keypress	Command
Ctrl + R	Reset current session.

7.6 Query

Keypress	Command
/	Search for lines matching a regular expression
;	Execute an SQL query
:	Execute an internal command, see *Command Reference* for more information
|	Execute an lnav script located in a format directory.
Ctrl +]	Abort the prompt

CHAPTER 8

Command Reference

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

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.

8.1 Filtering

The set of log messages that are displayed in the log view can be controlled with the following commands:

- filter-in <regex> - Only display log lines that match a regex.
- filter-out <regex> - Do not display log lines that match a regex.
- disable-filter <regex> - Disable the given filter.
- enable-filter <regex> - Enable the given filter.
- delete-filter <regex> - Delete the filter.
- set-min-log-level <level> - Only display log lines with the given log level or higher.
- hide-lines-before <abs-time|rel-time> - Hide lines before the given time.
- hide-lines-after <abs-time|rel-time> - Hide lines after the given time.
- show-lines-before-and-after - Show lines that were hidden by the "hide-lines" commands.

8.2 Bookmarks

- mark - Bookmark the top line in the view.
- partition-name <name> - Partition the log file around the top line in the log view and assign the given name. The top line and all that follow, up to the start of the next partition, will be included in the partition. The name

of the partition for a log line is visible in the top status bar to the right of the time stamp. The partition name for a log line can be retrieved via the *log_part* field in any log table.

- comment <text> - Attach a comment to the top line in the log view and bookmark that line.
- clear-comment - Clear the comment attached to the top line in the view.
- tag <tag1> [<tag2> ... [<tagN>]] - Attach one or more tags to a log line. A '#' will automatically be prepended to the tag name if it is not already there.
- untag <tag1> [<tag2> ... [<tagN>]] - Detach one or more tags from a log line.
- delete-tags <tag1> [<tag2> ... [<tagN>]] - Detach the given tags from all log lines.

8.3 Navigation

- goto <line#|N%|abs-time|relative-time> - Go to the given line number, N percent into the file, the given timestamp in the log view, or by the relative time (e.g. 'a minute ago').
- relative-goto <line#|N%> - Move the current view up or down by the given amount.
- next-mark error|warning|search|user|file|partition - Move to the next bookmark of the given type in the current view.
- prev-mark error|warning|search|user|file|partition - Move to the previous bookmark of the given type in the current view.
- prev-location - The previous location in the history.
- next-location - The next location in the history.

8.4 Time

- adjust-log-time <date|relative-time> - Change the timestamps for a log file.
- unix-time <secs-or-date> - Convert a unix-timestamp in seconds to a human-readable form or vice-versa.
- current-time - Print the current time in human-readable form and as a unix-timestamp.

8.5 Display

- help - Display the built-in help text.
- disable-word-wrap - Disable word wrapping in the log and text file views.
- enable-word-wrap - Enable word wrapping in the log and text file views.
- hide-fields <field-name> [<field-name2> ... <field-nameN>] - Hide large log message fields by replacing them with an ellipsis. You can quickly switching between showing and hiding hidden fields using the 'x' hotkey.
- show-fields <field-name> [<field-name2> ... <field-nameN>] - Show previously hidden log message fields.
- highlight <regex> - Colorize text that matches the given regex.
- clear-highlight <regex> - Clear a previous highlight.

- spectrogram <numeric-field> - Generate a spectrogram for a numeric log message field or SQL result column. The spectrogram view displays the range of possible values of the field on the horizontal axis and time on the vertical axis. The horizontal axis is split into buckets where each bucket counts how many log messages contained the field with a value in that range. The buckets are colored based on the count in the bucket: green means low, yellow means medium, and red means high. The exact ranges for the colors is computed automatically and displayed in the middle of the top line of the view. The minimum and maximum values for the field are displayed in the top left and right sides of the view, respectively.
- switch-to-view <name> - Switch to the given view name (e.g. log, text, ...)
- toggle-view <name> - Toggle the display of the given view (e.g. log, text, ...)
- zoom-to <zoom-level> - Set the zoom level for the histogram view.
- redraw - Redraw the window to correct any corruption.
- alt-msg <msg> - Set the message to be displayed on the bottom-right of the screen. This message is typically used for help text.

8.6 SQL

- create-logline-table <table-name> - Create an SQL table using the top line of the log view as a template. See the *Extracting Data* section for more information.
- delete-logline-table <table-name> - Delete a table created by create-logline-table.
- create-search-table <table-name> [regex] - Create an SQL table that extracts information from logs using the provided regular expression or the last search that was done. Any captures in the expression will be used as columns in the SQL table. If the capture is named, that name will be used as the column name, otherwise the column name will be of the form 'col_N'.
- delete-search-table <table-name> - Delete a table that was created with create-search-table.

8.7 Output

- append-to <file> - Append any bookmarked lines in the current view to the given file.
- write-to <file> - Overwrite the given file with any bookmarked lines in the current view. Use '-' to write the lines to the terminal and '/dev/clipboard' to write to the system clipboard.
- write-raw-to <file> - Overwrite the given file with all the lines in the current view. Use '-' to write the lines to the terminal and '/dev/clipboard' to write to the system clipboard.
- write-csv-to <file> - Write SQL query results to the given file in CSV format. Use '-' to write the lines to the terminal and '/dev/clipboard' to write to the system clipboard.
- write-json-to <file> - Write SQL query results to the given file in JSON format. Use '-' to write the lines to the terminal and '/dev/clipboard' to write to the system clipboard..
- pipe-to <shell-cmd> - Pipe the bookmarked lines in the current view to a shell command and open the output in lnav.
- pipe-line-to <shell-cmd> - Pipe the top line in the current view to a shell command and open the output in lnav.
- redirect-to [path] - If a path is given, all output from commands, like ":echo" and when writing to stdout (e.g. :write-to -), will be sent to the given file. If no path is specified, the current redirect will be cleared and output will be captured as it was before the redirect was done.

8.8 Miscellaneous

- echo [-n] <msg> - Display the given message in the command prompt. Useful for scripts to display messages to the user. The '-n' option leaves out the new line at the end of the message.
- eval <cmd> - Evaluate the given command or SQL query after performing environment variable substitution. The argument to *eval* must start with a colon, semi-colon, or pipe character to signify whether the argument is a command, SQL query, or a script to be executed, respectively.
- quit - Quit lnav. Alternatively, ':q' can be used as an alias for 'quit'.

8.9 Configuration

- config <option> - Get the current value of a configuration option.
- config <option> <value> - Set the value of a configuration option.
- reset-config <option> - Reset a configuration option to the default.
- save-config - Save the current configuration to ~/.lnav/config.json.

The following options are available:

- /ui/clock-format - Specifies the date-time format of the clock in the top-left corner of the UI. The format conversion specifiers are the same as in strftime(3).
- /ui/dim-text - Reduce the brightness of text. This setting can be useful when running in an xterm where the white color is very bright.
- /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.

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.

CHAPTER 9

SQLite Extensions Reference

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.

9.1 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.

9.2 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;
```

9.3 Math

Basic mathematical functions:

- cos(n)
- sin(n)
- tan(n)

- cot(n)
- cosh(n)
- sinh(n)
- coth(n)
- acos(n)
- asin(n)
- atan(r1,r2)
- atan2(r1,r2)
- exp(n)
- log(n)
- log10(n)
- power(x,y)
- sign(n) - Return one of 3 possibilities +1,0 or -1 when the argument is respectively positive, 0 or negative.
- sqrt(n)
- square(n)
- ceil(n)
- floor(n)
- pi()
- degrees - Convert radians to degrees
- radians - Convert degrees to radians

Aggregate functions:

- stddev
- variance
- mode
- median
- lower_quartile
- upper_quartile

9.4 String

Additional string comparison and manipulation functions:

- difference(s1,s2) - Computes the number of different characters between the soundex value fo 2 strings.
- replicate(s,n) - Given a string (s) in the first argument and an integer (n) in the second returns the string that constains s contatenated n times.
- proper(s) - Ensures that the words in the given string have their first letter capitalized and the following letters are lower case.

- charindex(s1,s2), charindex(s1,s2,n) - Given 2 input strings (s1,s2) and an integer (n) searches from the nth character for the string s1. Returns the position where the match occured. Characters are counted from 1. 0 is returned when no match occurs.
- leftstr(s,n) - Given a string (s) and an integer (n) returns the n leftmost (UTF-8) characters if the string has a length<=n or is NULL this function is NOP.
- rightstr(s,n) - Given a string (s) and an integer (n) returns the n rightmost (UTF-8) characters if the string has a length<=n or is NULL this function is NOP
- reverse(s) - Given a string returns the same string but with the characters in reverse order.
- padl(s,n) - Given an input string (s) and an integer (n) adds spaces at the beginning of (s) until it has a length of n characters. When s has a length >=n it's a NOP. padl(NULL) = NULL
- padr(s,n) - Given an input string (s) and an integer (n) appends spaces at the end of s until it has a length of n characters. When s has a length >=n it's a NOP. padr(NULL) = NULL
- padc(s,n) - Given an input string (s) and an integer (n) appends spaces at the end of s and adds spaces at the begining of s until it has a length of n characters. Tries to add has many characters at the left as at the right. When s has a length >=n it's a NOP. padc(NULL) = NULL
- strfilter(s1,s2) - Given 2 string (s1,s2) returns the string s1 with the characters NOT in s2 removed assumes strings are UTF-8 encoded.
- regexp(re,s) - Return 1 if the regular expression 're' matches the given string.
- regexp_replace(str, re, repl) - Replace the portions of the given string that match the regular expression with the replacement string. **NOTE**: The arguments for the string and the regular expression in this function are reversed from the plain regexp() function. This is to be somewhat compatible with functions in other database implementations.
- startswith(s1,prefix) - Given a string and prefix, return 1 if the string starts with the given prefix.
- endswith(s1,suffix) - Given a string and suffix, return 1 if the string ends with the given suffix.
- regexp_match(re,str) - Match and extract values from a string using a regular expression. The "re" argument should be a PCRE with captures. If there is a single capture, that captured value will be directly returned. If there is more than one capture, a JSON object will be returned with field names matching the named capture groups or 'col_N' where 'N' is the index of the capture. If the expression does not match the string, NULL is returned.
- extract(str) - Parse and extract values from a string using the same algorithm as the *logline* table (see *Extracting Data*). The discovered data is returned as a JSON-object that you can do further processing on.
- spooky_hash(str1, . . .) - Compute the hash value for the given arguments using the "spooky" hash algorithm.
- group_spooky_hash(str1, . . .) - An aggregate version of the "spooky_hash()" function.

9.5 File Paths

File path manipulation functions:

- basename(s) - Return the file name part of a path.
- dirname(s) - Return the directory part of a path.
- joinpath(s1,s2,. . .) - Return the arguments joined together into a path.

9.6 Networking

Network information functions:

- gethostbyname - Convert a host name into an IP address. The host name could not be resolved, the input value will be returned.
- gethostbyaddr - Convert an IPv4/IPv6 address into a host name. If the reverse lookup fails, the input value will be returned.

9.7 JSON

JSON functions:

- jget(json, json_ptr) - Get the value from the JSON-encoded string in first argument that is referred to by the JSON-Pointer in the second.
- json_group_object(key0, value0, ... keyN, valueN) - An aggregate function that creates a JSON-encoded object from the key value pairs given as arguments.
- json_group_array(value0, ... valueN) - An aggregate function that creates a JSON-encoded array from the values given as arguments.

9.8 Time

Time functions:

- timeslice(t, s) - Given a time stamp (t) and a time slice (s), return a timestamp for the bucket of time that the timestamp falls in. For example, with the timestamp "2015-03-01 11:02:00' and slice '5min' the returned value will be '2015-03-01 11:00:00'. This function can be useful when trying to group together log messages into buckets.

9.9 Internal State

The following functions can be used to access **lnav**'s internal state:

- log_top_line() - Return the line number at the top of the log view.
- log_top_datetime() - Return the timestamp of the line at the top of the log view.

9.10 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.

CHAPTER 10

SQLite Tables Reference

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

- environ
- lnav_views
- all_logs
- http_status_codes

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
```

10.1 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
```

10.2 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.

10.3 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.

10.4 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.

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.

10.5 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.

10.6 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.

10.7 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 11

Indices and tables

- genindex
- modindex
- search