## NAME

innotop - MySQL and InnoDB transaction/status monitor.

## SYNOPSIS

To monitor servers normally:

```
innotop
```

To monitor InnoDB status information from a file:

```
innotop /var/log/mysql/mysqld.err
```

To run innotop non-interactively in a pipe-and-filter configuration:

```
innotop --count 5 -d 1 -n
```

To monitor a database on another system using a particular username and password:

```
innotop -u <username> -p <password> -h <hostname>
```

## DESCRIPTION

innotop monitors MySQL servers. Each of its modes shows you a different aspect of what's happening in the server. For example, there's a mode for monitoring replication, one for queries, and one for transactions. innotop refreshes its data periodically, so you see an updating view.

innotop has lots of features for power users, but you can start and run it with virtually no configuration. If you're just getting started, see ''QUICK-START''. Press '?' at any time while running innotop for context-sensitive help.

## QUICK-START

To start innotop, open a terminal or command prompt. If you have installed innotop on your system, you should be able to just type ''innotop'' and press Enter; otherwise, you will need to change to innotop's directory and type ''perl innotop''.

With no options specified, innotop will attempt to connect to a MySQL server on localhost using mysql_read_default_group=client for other connection parameters. If you need to specify a different username and password, use the -u and -p options, respectively. To monitor a MySQL database on another host, use the -h option.

After you've connected, innotop should show you something like the following:

```
[RO] Query List (? for help) localhost, 01:11:19, 449.44 QPS, 14/7/163 con/run

CXN When Load QPS Slow QCacheHit KCacheHit BpsIn BpsOut
localhost Total 0.00 1.07k 697 0.00% 98.17% 476.83k 242.83k

CXN Cmd ID User Host DB Time Query
localhost Query 766446598 test 10.0.0.1 foo 00:02 INSERT INTO table (
```

(This sample is truncated at the right so it will fit on a terminal when running 'man innotop')

If your server is busy, you'll see more output. Notice the first line on the screen, which tells you that readonly is set to true ([RO]), what mode you're in and what server you're connected to. You can change to other modes with keystrokes; press 'T' to switch to a list of InnoDB transactions, for example.

Press the '?' key to see what keys are active in the current mode. You can press any of these keys and innotop will either take the requested action or prompt you for more input. If your system has Term::ReadLine support, you can use TAB and other keys to auto-complete and edit input.

To quit innotop, press the 'q' key.

## OPTIONS

innotop is mostly configured via its configuration file, but some of the configuration options can come from the command line. You can also specify a file to monitor for InnoDB status output; see ''MONITORING A FILE'' for more details.

You can negate some options by prefixing the option name with --no. For example, --noinc (or --no-inc) negates ''--inc''.

--color
Enable or disable terminal coloring. Corresponds to the ''color'' config file setting.

--config
Specifies a configuration file to read. This option is non-sticky, that is to say it does not persist to the configuration file itself.

--count
Refresh only the specified number of times (ticks) before exiting. Each refresh is a pause for ''interval'' seconds, followed by requesting data from MySQL connections and printing it to the terminal.

--delay
Specifies the amount of time to pause between ticks (refreshes). Corresponds to the configuration option ''interval''.

--help
Print a summary of command-line usage and exit.

--host
Host to connect to.

--inc
Specifies whether innotop should display absolute numbers or relative numbers (offsets from their previous values). Corresponds to the configuration option ''status_inc''.

--mode
Specifies the mode in which innotop should start. Corresponds to the configuration option ''mode''.

--nonint
Enable non-interactive operation. See ''NON-INTERACTIVE OPERATION'' for more.

--password
Password to use for connection.

--port
Port to use for connection.

--skipcentral
Don't read the central configuration file.

--user
User to use for connection.

--version
Output version information and exit.

--write
Sets the configuration option ''readonly'' to 0, making innotop write the running configuration to ~/.innotop/innotop.conf on exit, if no configuration file was loaded at start-up.

## HOTKEYS

innotop is interactive, and you control it with key-presses.

- Uppercase keys switch between modes.
- Lowercase keys initiate some action within the current mode.
- Other keys do something special like change configuration or show the innotop license.

Press '?' at any time to see the currently active keys and what they do.

## MODES

Each of innotop's modes retrieves and displays a particular type of data from the servers you're monitoring. You switch between modes with uppercase keys. The following is a brief description of each mode, in alphabetical order. To switch to the mode, press the key listed in front of its heading in the following list:

B: InnoDB Buffers

This mode displays information about the InnoDB buffer pool, page statistics, insert buffer, and adaptive hash index. The data comes from SHOW INNODB STATUS.

This mode contains the ''buffer_pool'', ''page_statistics'', ''insert_buffers'', and ''adaptive_hash_index'' tables by default.

C: Command Summary

This mode is similar to mytop's Command Summary mode. It shows the ''cmd_summary'' table, which looks something like the following:

```
Command Summary (? for help) localhost, 25+07:16:43, 2.45 QPS, 3 thd, 5.0.40
_____________________ Command Summary _____________________
Name Value Pct Last Incr Pct
Select_scan 3244858 69.89% 2 100.00%
Select_range 1354177 29.17% 0 0.00%
Select_full_join 39479 0.85% 0 0.00%
Select_full_range_join 4097 0.09% 0 0.00%
Select_range_check 0 0.00% 0 0.00%
```

The command summary table is built by extracting variables from ''STATUS_VARIABLES''. The variables must be numeric and must match the prefix given by the ''cmd_filter'' configuration variable. The variables are then sorted by value descending and compared to the last variable, as shown above. The percentage columns are percentage of the total of all variables in the table, so you can see the relative weight of the variables.

The example shows what you see if the prefix is ''Select_''. The default prefix is ''Com_''. You can choose a prefix with the 's' key.

It's rather like running SHOW VARIABLES LIKE ''prefix%'' with memory and nice formatting.

Values are aggregated across all servers. The Pct columns are not correctly aggregated across multiple servers. This is a known limitation of the grouping algorithm that may be fixed in the future.

D: InnoDB Deadlocks

This mode shows the transactions involved in the last InnoDB deadlock. A second table shows the locks each transaction held and waited for. A deadlock is caused by a cycle in the waits-for graph, so there should be two locks held and one waited for unless the deadlock information is truncated.

InnoDB puts deadlock information before some other information in the SHOW INNODB STATUS output. If there are a lot of locks, the deadlock information can grow very large, and there is a limit on the size of the SHOW INNODB STATUS output. A large deadlock can fill the entire output, or even be truncated, and prevent you from seeing other information at all. If you are running innotop in another mode, for example T mode, and suddenly you don't see anything, you might want to check and see if a deadlock has wiped out the data you need.

If it has, you can create a small deadlock to replace the large one. Use the 'w' key to 'wipe' the large deadlock with a small one. This will not work unless you have defined a deadlock table for the connection (see ''SERVER CONNECTIONS'').

You can also configure innotop to automatically detect when a large deadlock needs to be replaced with a small one (see ''auto_wipe_dl'').

This mode displays the ‘‘deadlock_transactions’’ and ‘‘deadlock_locks’’ tables by default.

F: InnoDB Foreign Key Errors
This mode shows the last InnoDB foreign key error information, such as the table where it happened, when and who and what query caused it, and so on.

InnoDB has a huge variety of foreign key error messages, and many of them are just hard to parse. innotop doesn't always do the best job here, but there's so much code devoted to parsing this messy, unparseable output that innotop is likely never to be perfect in this regard. If innotop doesn't show you what you need to see, just look at the status text directly.

This mode displays the ‘‘fk_error’’ table by default.

I: InnoDB I/O Info
This mode shows InnoDB's I/O statistics, including the I/O threads, pending I/O, file I/O miscellaneous, and log statistics. It displays the ‘‘io_threads’’, ‘‘pending_io’’, ‘‘file_io_misc’’, and ‘‘log_statistics’’ tables by default.

L: Locks
This mode shows information about current locks. At the moment only InnoDB locks are supported, and by default you'll only see locks for which transactions are waiting. This information comes from the TRANSACTIONS section of the InnoDB status text. If you have a very busy server, you may have frequent lock waits; it helps to be able to see which tables and indexes are the ‘‘hot spot’’ for locks. If your server is running pretty well, this mode should show nothing.

You can configure MySQL and innotop to monitor not only locks for which a transaction is waiting, but those currently held, too. You can do this with the InnoDB Lock Monitor (<http://dev.mysql.com/doc/en/innodb-monitor.html>).
It's not documented in the MySQL manual, but creating the lock monitor with the following statement also affects the output of SHOW INNODB STATUS, which innotop uses:

```
CREATE TABLE innodb_lock_monitor(a int) ENGINE=INNODB;
```

This causes InnoDB to print its output to the MySQL file every 16 seconds or so, as stated in the manual, but it also makes the normal SHOW INNODB STATUS output include lock information, which innotop can parse and display (that's the undocumented feature).

This means you can do what may have seemed impossible: to a limited extent (InnoDB truncates some information in the output), you can see which transaction holds the locks something else is waiting for. You can also enable and disable the InnoDB Lock Monitor with the key mappings in this mode.

This mode displays the ‘‘innodb_locks’’ table by default. Here's a sample of the screen when one connection is waiting for locks another connection holds:

```
_____________________________________ InnoDB Locks _____________________________
CXN ID Type Waiting Wait Active Mode DB Table Index
localhost 12 RECORD 1 00:10 00:10 X test t1 PRIMARY
localhost 12 TABLE 0 00:10 00:10 IX test t1
localhost 12 RECORD 1 00:10 00:10 X test t1 PRIMARY
localhost 11 TABLE 0 00:00 00:25 IX test t1
localhost 11 RECORD 0 00:00 00:25 X test t1 PRIMARY
```

You can see the first connection, ID 12, is waiting for a lock on the PRIMARY key on test.t1, and has been waiting for 10 seconds. The second connection isn't waiting, because the Waiting column is 0, but it holds locks on the same index. That tells you connection 11 is blocking connection 12.

M: Master/Slave Replication Status
: This mode shows the output of SHOW SLAVE STATUS and SHOW MASTER STATUS in three tables. The first two divide the slave's status into SQL and I/O thread status, and the last shows master status. Filters are applied to eliminate non-slave servers from the slave tables, and non-master servers from the master table.

  This mode displays the ‘‘slave_sql_status’’, ‘‘slave_io_status’’, and ‘‘master_status’’ tables by default.

O: Open Tables
: This section comes from MySQL's SHOW OPEN TABLES command. By default it is filtered to show tables which are in use by one or more queries, so you can get a quick look at which tables are 'hot'. You can use this to guess which tables might be locked implicitly.

  This mode displays the ‘‘open_tables’’ mode by default.

Q: Query List
: This mode displays the output from SHOW FULL PROCESSLIST, much like **mytop**'s query list mode. This mode does **not** show InnoDB-related information. This is probably one of the most useful modes for general usage.

  There is an informative header that shows general status information about your server. You can toggle it on and off with the 'h' key. By default, innotop hides inactive processes and its own process. You can toggle these on and off with the 'i' and 'a' keys.

  You can EXPLAIN a query from this mode with the 'e' key. This displays the query's full text, the results of EXPLAIN, and in newer MySQL versions, even the optimized query resulting from EXPLAIN EXTENDED. innotop also tries to rewrite certain queries to make them EXPLAIN-able. For example, INSERT/SELECT statements are rewritable.

  This mode displays the ‘‘q_header’’ and ‘‘processlist’’ tables by default.

R: InnoDB Row Operations and Semaphores
: This mode shows InnoDB row operations, row operation miscellaneous, semaphores, and information from the wait array. It displays the ‘‘row_operations’’, ‘‘row_operation_misc’’, ‘‘semaphores’’, and ‘‘wait_array’’ tables by default.

S: Variables & Status
: This mode calculates statistics, such as queries per second, and prints them out in several different styles. You can show absolute values, or incremental values between ticks.

  You can switch between the views by pressing a key. The 's' key prints a single line each time the screen updates, in the style of **vmstat**. The 'g' key changes the view to a graph of the same numbers, sort of like **tload**. The 'v' key changes the view to a pivoted table of variable names on the left, with successive updates scrolling across the screen from left to right. You can choose how many updates to put on the screen with the ‘‘num_status_sets’’ configuration variable.

  Headers may be abbreviated to fit on the screen in interactive operation. You choose which variables to display with the 'c' key, which selects from predefined sets, or lets you create your own sets. You can edit the current set with the 'e' key.

  This mode doesn't really display any tables like other modes. Instead, it uses a table definition to extract and format the data, but it then transforms the result in special ways before outputting it. It uses the ‘‘var_status’’ table definition for this.

T: InnoDB Transactions
: This mode shows transactions from the InnoDB monitor's output, in **top**-like format. This mode is the reason I wrote innotop.

  You can kill queries or processes with the 'k' and 'x' keys, and EXPLAIN a query with the 'e' or 'f' keys. InnoDB doesn't print the full query in transactions, so explaining may not work

right if the query is truncated.

The informational header can be toggled on and off with the 'h' key. By default, innotop hides inactive transactions and its own transaction. You can toggle this on and off with the 'i' and 'a' keys.

This mode displays the ''t_header'' and ''innodb_transactions'' tables by default.

## INNOTOP STATUS

The first line innotop displays is a ''status bar'' of sorts. What it contains depends on the mode you're in, and what servers you're monitoring. The first few words are always [RO] (if readonly is set to 1), the innotop mode, such as ''InnoDB Txns'' for T mode, followed by a reminder to press '?' for help at any time.

### ONE SERVER

The simplest case is when you're monitoring a single server. In this case, the name of the connection is next on the status line. This is the name you gave when you created the connection — most likely the MySQL server's hostname. This is followed by the server's uptime.

If you're in an InnoDB mode, such as T or B, the next word is ''InnoDB'' followed by some information about the SHOW INNODB STATUS output used to render the screen. The first word is the number of seconds since the last SHOW INNODB STATUS, which InnoDB uses to calculate some per-second statistics. The next is a smiley face indicating whether the InnoDB output is truncated. If the smiley face is a :-), all is well; there is no truncation. A :^| means the transaction list is so long, InnoDB has only printed out some of the transactions. Finally, a frown :-( means the output is incomplete, which is probably due to a deadlock printing too much lock information (see ''D: InnoDB Deadlocks'').

The next two words indicate the server's queries per second (QPS) and how many threads (connections) exist. Finally, the server's version number is the last thing on the line.

### MULTIPLE SERVERS

If you are monitoring multiple servers (see ''SERVER CONNECTIONS''), the status line does not show any details about individual servers. Instead, it shows the names of the connections that are active. Again, these are connection names you specified, which are likely to be the server's hostname. A connection that has an error is prefixed with an exclamation point.

If you are monitoring a group of servers (see ''SERVER GROUPS''), the status line shows the name of the group. If any connection in the group has an error, the group's name is followed by the fraction of the connections that don't have errors.

See ''ERROR HANDLING'' for more details about innotop's error handling.

### MONITORING A FILE

If you give a filename on the command line, innotop will not connect to ANY servers at all. It will watch the specified file for InnoDB status output and use that as its data source. It will always show a single connection called 'file'. And since it can't connect to a server, it can't determine how long the server it's monitoring has been up; so it calculates the server's uptime as time since innotop started running.

## SERVER ADMINISTRATION

While innotop is primarily a monitor that lets you watch and analyze your servers, it can also send commands to servers. The most frequently useful commands are killing queries and stopping or starting slaves.

You can kill a connection, or in newer versions of MySQL kill a query but not a connection, from ''Q: Query List'' and ''T: InnoDB Transactions'' modes. Press 'k' to issue a KILL command, or 'x' to issue a KILL QUERY command. innotop will prompt you for the server and/or connection ID to kill (innotop does not prompt you if there is only one possible choice for any input). innotop pre-selects the longest-running query, or the oldest connection. Confirm the command with 'y'.

In ‘‘Slave Replication Status’’‘‘ in ’’M: Master mode, you can start and stop slaves with the ’a’ and ’o’ keys, respectively. You can send these commands to many slaves at once. innotop fills in a default command of START SLAVE or STOP SLAVE for you, but you can actually edit the command and send anything you wish, such as SET GLOBAL SQL_SLAVE_SKIP_COUNTER=1 to make the slave skip one binlog event when it starts.

You can also ask innotop to calculate the earliest binlog in use by any slave and issue a PURGE MASTER LOGS on the master. Use the ’b’ key for this. innotop will prompt you for a master to run the command on, then prompt you for the connection names of that master’s slaves (there is no way for innotop to determine this reliably itself). innotop will find the minimum binlog in use by these slave connections and suggest it as the argument to PURGE MASTER LOGS.

## SERVER CONNECTIONS

When you create a server connection using ’@’, innotop asks you for a series of inputs, as follows:

DSN

A DSN is a Data Source Name, which is the initial argument passed to the DBI module for connecting to a server. It is usually of the form

```
DBI:mysql:;mysql_read_default_group=mysql;host=HOSTNAME
```

Since this DSN is passed to the DBD::mysql driver, you should read the driver’s documentation at ‘‘/search.cpan.org/dist/DBD-mysql/lib/DBD/mysql.pm’’‘‘ in ’’http: for the exact details on all the options you can pass the driver in the DSN. You can read more about DBI at <http://dbi.perl.org/docs/>, and especially at <http://search.cpan.org/~timb/DBI/DBI.pm>.

The mysql_read_default_group=mysql option lets the DBD driver read your MySQL options files, such as ~/.my.cnf on UNIX-ish systems. You can use this to avoid specifying a username or password for the connection.

InnoDB Deadlock Table

This optional item tells innotop a table name it can use to deliberately create a small deadlock (see ‘‘D: InnoDB Deadlocks’’). If you specify this option, you just need to be sure the table doesn’t exist, and that innotop can create and drop the table with the InnoDB storage engine. You can safely omit or just accept the default if you don’t intend to use this.

Username

innotop will ask you if you want to specify a username. If you say ’y’, it will then prompt you for a user name. If you have a MySQL option file that specifies your username, you don’t have to specify a username.

The username defaults to your login name on the system you’re running innotop on.

Password

innotop will ask you if you want to specify a password. Like the username, the password is optional, but there’s an additional prompt that asks if you want to save the password in the innotop configuration file. If you don’t save it in the configuration file, innotop will prompt you for a password each time it starts. Passwords in the innotop configuration file are saved in plain text, not encrypted in any way.

Once you finish answering these questions, you should be connected to a server. But innotop isn’t limited to monitoring a single server; you can define many server connections and switch between them by pressing the ’@’ key. See ‘‘SWITCHING BETWEEN CONNECTIONS’’.

## SERVER GROUPS

If you have multiple MySQL instances, you can put them into named groups, such as ’all’, ’masters’, and ’slaves’, which innotop can monitor all together.

You can choose which group to monitor with the ’#’ key, and you can press the TAB key to switch to the next group. If you’re not currently monitoring a group, pressing TAB selects the first group.

To create a group, press the '#' key and type the name of your new group, then type the names of the connections you want the group to contain.

## SWITCHING BETWEEN CONNECTIONS

innotop lets you quickly switch which servers you're monitoring. The most basic way is by pressing the '@' key and typing the name(s) of the connection(s) you want to use. This setting is per-mode, so you can monitor different connections in each mode, and innotop remembers which connections you choose.

You can quickly switch to the 'next' connection in alphabetical order with the 'n' key. If you're monitoring a server group (see "SERVER GROUPS") this will switch to the first connection.

You can also type many connection names, and innotop will fetch and display data from them all. Just separate the connection names with spaces, for example "server1 server2." Again, if you type the name of a connection that doesn't exist, innotop will prompt you for connection information and create the connection.

Another way to monitor multiple connections at once is with server groups. You can use the TAB key to switch to the 'next' group in alphabetical order, or if you're not monitoring any groups, TAB will switch to the first group.

innotop does not fetch data in parallel from connections, so if you are monitoring a large group or many connections, you may notice increased delay between ticks.

When you monitor more than one connection, innotop's status bar changes. See "INNOTOP STATUS".

## ERROR HANDLING

Error handling is not that important when monitoring a single connection, but is crucial when you have many active connections. A crashed server or lost connection should not crash innotop. As a result, innotop will continue to run even when there is an error; it just won't display any information from the connection that had an error. Because of this, innotop's behavior might confuse you. It's a feature, not a bug!

innotop does not continue to query connections that have errors, because they may slow innotop and make it hard to use, especially if the error is a problem connecting and causes a long time-out. Instead, innotop retries the connection occasionally to see if the error still exists. If so, it will wait until some point in the future. The wait time increases in ticks as the Fibonacci series, so it tries less frequently as time passes.

Since errors might only happen in certain modes because of the SQL commands issued in those modes, innotop keeps track of which mode caused the error. If you switch to a different mode, innotop will retry the connection instead of waiting.

By default innotop will display the problem in red text at the bottom of the first table on the screen. You can disable this behavior with the "show_cxn_errors_in_tbl" configuration option, which is enabled by default. If the "debug" option is enabled, innotop will display the error at the bottom of every table, not just the first. And if "show_cxn_errors" is enabled, innotop will print the error text to STDOUT as well. Error messages might only display in the mode that caused the error, depending on the mode and whether innotop is avoiding querying that connection.

## NON-INTERACTIVE OPERATION

You can run innotop in non-interactive mode, in which case it is entirely controlled from the configuration file and command-line options. To start innotop in non-interactive mode, give the L"<--nonint"> command-line option. This changes innotop's behavior in the following ways:

- Certain Perl modules are not loaded. Term::Readline is not loaded, since innotop doesn't prompt interactively. Term::ANSIColor and Win32::Console::ANSI modules are not loaded. Term::ReadKey is still used, since innotop may have to prompt for connection passwords when starting up.

- innotop does not clear the screen after each tick.
- innotop does not persist any changes to the configuration file.
- If ''--count'' is given and innotop is in incremental mode (see ''status_inc'' and ''--inc''), innotop actually refreshes one more time than specified so it can print incremental statistics. This suppresses output during the first tick, so innotop may appear to hang.
- innotop only displays the first table in each mode. This is so the output can be easily processed with other command-line utilities such as awk and sed. To change which tables display in each mode, see ''TABLES''. Since ''Q: Query List'' mode is so important, innotop automatically disables the ''q_header'' table. This ensures you'll see the ''processlist'' table, even if you have innotop configured to show the q_header table during interactive operation. Similarly, in ''T: InnoDB Transactions'' mode, the ''t_header'' table is suppressed so you see only the ''innodb_transactions'' table.
- All output is tab-separated instead of being column-aligned with whitespace, and innotop prints the full contents of each table instead of only printing one screenful at a time.
- innotop only prints column headers once instead of every tick (see ''hide_hdr''). innotop does not print table captions (see ''display_table_captions''). innotop ensures there are no empty lines in the output.
- innotop does not honor the ''shorten'' transformation, which normally shortens some numbers to human-readable formats.
- innotop does not print a status line (see ''INNOTOP STATUS'').

## CONFIGURING

Nearly everything about innotop is configurable. Most things are possible to change with built-in commands, but you can also edit the configuration file.

While running innotop, press the '$' key to bring up the configuration editing dialog. Press another key to select the type of data you want to edit:

S: Statement Sleep Times
: Edits SQL statement sleep delays, which make innotop pause for the specified amount of time after executing a statement. See ''SQL STATEMENTS'' for a definition of each statement and what it does. By default innotop does not delay after any statements.

  This feature is included so you can customize the side-effects caused by monitoring your server. You may not see any effects, but some innotop users have noticed that certain MySQL versions under very high load with InnoDB enabled take longer than usual to execute SHOW GLOBAL STATUS. If innotop calls SHOW FULL PROCESSLIST immediately afterward, the processlist contains more queries than the machine actually averages at any given moment. Configuring innotop to pause briefly after calling SHOW GLOBAL STATUS alleviates this effect.

  Sleep times are stored in the ''stmt_sleep_times'' section of the configuration file. Fractional-second sleeps are supported, subject to your hardware's limitations.

c: Edit Columns
: Starts the table editor on one of the displayed tables. See ''TABLE EDITOR''. An alternative way to start the table editor without entering the configuration dialog is with the '^' key.

g: General Configuration
: Starts the configuration editor to edit global and mode-specific configuration variables (see ''MODES''). innotop prompts you to choose a variable from among the global and mode-specific ones depending on the current mode.

k: Row-Coloring Rules
: Starts the row-coloring rules editor on one of the displayed table(s). See ''COLORS'' for details.

p: Manage Plugins
Starts the plugin configuration editor. See ''PLUGINS'' for details.

s: Server Groups
Lets you create and edit server groups. See ''SERVER GROUPS''.

t: Choose Displayed Tables
Lets you choose which tables to display in this mode. See ''MODES'' and ''TABLES''.

## CONFIGURATION FILE

innotop's default configuration file locations are `$HOME`/.innotop and /etc/innotop/innotop.conf, and they are looked for in that order. If the first configuration file exists, the second will not be processed. Those can be overridden with the ''--config'' command-line option. You can edit it by hand safely, however innotop reads the configuration file when it starts, and, if readonly is set to 0, writes it out again when it exits. Thus, if readonly is set to 0, any changes you make by hand while innotop is running will be lost.

innotop doesn't store its entire configuration in the configuration file. It has a huge set of default configuration values that it holds only in memory, and the configuration file only overrides these defaults. When you customize a default setting, innotop notices, and then stores the customizations into the file. This keeps the file size down, makes it easier to edit, and makes upgrades easier.

A configuration file is read-only be default. You can override that with ''--write''. See ''readonly''.

The configuration file is arranged into sections like an INI file. Each section begins with [section-name] and ends with [/section-name]. Each section's entries have a different syntax depending on the data they need to store. You can put comments in the file; any line that begins with a # character is a comment. innotop will not read the comments, so it won't write them back out to the file when it exits. Comments in read-only configuration files are still useful, though.

The first line in the file is innotop's version number. This lets innotop notice when the file format is not backwards-compatible, and upgrade smoothly without destroying your customized configuration.

The following list describes each section of the configuration file and the data it contains:

general
The 'general' section contains global configuration variables and variables that may be mode-specific, but don't belong in any other section. The syntax is a simple key=value list. innotop writes a comment above each value to help you edit the file by hand.

S_func
Controls S mode presentation (see ''S: Variables & Status''). If g, values are graphed; if s, values are like vmstat; if p, values are in a pivoted table.

S_set
Specifies which set of variables to display in ''S: Variables & Status'' mode. See ''VARIABLE SETS''.

auto_wipe_dl
Instructs innotop to automatically wipe large deadlocks when it notices them. When this happens you may notice a slight delay. At the next tick, you will usually see the information that was being truncated by the large deadlock.

charset
Specifies what kind of characters to allow through the ''no_ctrl_char'' transformation. This keeps non-printable characters from confusing a terminal when you monitor queries that contain binary data, such as images.

The default is 'ascii', which considers anything outside normal ASCII to be a control character. The other allowable values are 'unicode' and 'none'. 'none' considers every character a control character, which can be useful for collapsing ALL text fields in

queries.

cmd_filter
: This is the prefix that filters variables in ‘‘C: Command Summary’’ mode.

color
: Whether terminal coloring is permitted.

cxn_timeout
: On MySQL versions 4.0.3 and newer, this variable is used to set the connection's timeout, so MySQL doesn't close the connection if it is not used for a while. This might happen because a connection isn't monitored in a particular mode, for example.

debug
: This option enables more verbose errors and makes innotop more strict in some places. It can help in debugging filters and other user-defined code. It also makes innotop write a lot of information to ‘‘debugfile’’ when there is a crash.

debugfile
: A file to which innotop will write information when there is a crash. See ‘‘FILES’’.

display_table_captions
: innotop displays a table caption above most tables. This variable suppresses or shows captions on all tables globally. Some tables are configured with the hide_caption property, which overrides this.

global
: Whether to show GLOBAL variables and status. innotop only tries to do this on servers which support the GLOBAL option to SHOW VARIABLES and SHOW STATUS. In some MySQL versions, you need certain privileges to do this; if you don't have them, innotop will not be able to fetch any variable and status data. This configuration variable lets you run innotop and fetch what data you can even without the elevated privileges.

  I can no longer find or reproduce the situation where GLOBAL wasn't allowed, but I know there was one.

graph_char
: Defines the character to use when drawing graphs in ‘‘S: Variables & Status’’ mode.

header_highlight
: Defines how to highlight column headers. This only works if Term::ANSIColor is available. Valid values are 'bold' and 'underline'.

hide_hdr
: Hides column headers globally.

interval
: The interval at which innotop will refresh its data (ticks). The interval is implemented as a sleep time between ticks, so the true interval will vary depending on how long it takes innotop to fetch and render data.

  This variable accepts fractions of a second.

mode
: The mode in which innotop should start. Allowable arguments are the same as the key presses that select a mode interactively. See ‘‘MODES’’.

num_digits
: How many digits to show in fractional numbers and percents. This variable's range is between 0 and 9 and can be set directly from ‘‘S: Variables & Status’’ mode with the '+' and '-' keys. It is used in the ‘‘set_precision’’, ‘‘shorten’’, and ‘‘percent’’ transformations.

num_status_sets
: Controls how many sets of status variables to display in pivoted ''S: Variables & Status'' mode. It also controls the number of old sets of variables innotop keeps in its memory, so the larger this variable is, the more memory innotop uses.

plugin_dir
: Specifies where plugins can be found. By default, innotop stores plugins in the 'plugins' subdirectory of your innotop configuration directory.

readonly
: Whether the configuration file is readonly. This cannot be set interactively.

show_cxn_errors
: Makes innotop print connection errors to STDOUT. See ''ERROR HANDLING''.

show_cxn_errors_in_tbl
: Makes innotop display connection errors as rows in the first table on screen. See ''ERROR HANDLING''.

show_percent
: Adds a '%' character after the value returned by the ''percent'' transformation.

show_statusbar
: Controls whether to show the status bar in the display. See ''INNOTOP STATUS''.

skip_innodb
: Disables fetching SHOW INNODB STATUS, in case your server(s) do not have InnoDB enabled and you don't want innotop to try to fetch it. This can also be useful when you don't have the SUPER privilege, required to run SHOW INNODB STATUS.

status_inc
: Whether to show absolute or incremental values for status variables. Incremental values are calculated as an offset from the last value innotop saw for that variable. This is a global setting, but will probably become mode-specific at some point. Right now it is honored a bit inconsistently; some modes don't pay attention to it.

plugins
: This section holds a list of package names of active plugins. If the plugin exists, innotop will activate it. See ''PLUGINS'' for more information.

filters
: This section holds user-defined filters (see ''FILTERS''). Each line is in the format filter_name=text='filter text' tbls='table list'.

  The filter text is the text of the subroutine's code. The table list is a list of tables to which the filter can apply. By default, user-defined filters apply to the table for which they were created, but you can manually override that by editing the definition in the configuration file.

active_filters
: This section stores which filters are active on each table. Each line is in the format table_name=filter_list.

tbl_meta
: This section stores user-defined or user-customized columns (see ''COLUMNS''). Each line is in the format col_name=properties, where the properties are a name=quoted-value list.

connections
: This section holds the server connections you have defined. Each line is in the format name=properties, where the properties are a name=value list. The properties are self-explanatory, and the only one that is treated specially is 'pass' which is only present if 'savepass' is set. This section of the configuration file will be skipped if any DSN, username, or password command-line options are used. See ''SERVER CONNECTIONS''.

active_connections
: This section holds a list of which connections are active in each mode. Each line is in the format mode_name=connection_list.

server_groups
: This section holds server groups. Each line is in the format name=connection_list. See ''SERVER GROUPS''.

active_server_groups
: This section holds a list of which server group is active in each mode. Each line is in the format mode_name=server_group.

max_values_seen
: This section holds the maximum values seen for variables. This is used to scale the graphs in ''S: Variables & Status'' mode. Each line is in the format name=value.

active_columns
: This section holds table column lists. Each line is in the format tbl_name=column_list. See ''COLUMNS''.

sort_cols
: This section holds the sort definition. Each line is in the format tbl_name=column_list. If a column is prefixed with '-', that column sorts descending. See ''SORTING''.

visible_tables
: This section defines which tables are visible in each mode. Each line is in the format mode_name=table_list. See ''TABLES''.

varsets
: This section defines variable sets for use in ''S: Status & Variables'' mode. Each line is in the format name=variable_list. See ''VARIABLE SETS''.

colors
: This section defines colorization rules. Each line is in the format tbl_name=property_list. See ''COLORS''.

stmt_sleep_times
: This section contains statement sleep times. Each line is in the format statement_name=sleep_time. See ''S: Statement Sleep Times''.

group_by
: This section contains column lists for table group_by expressions. Each line is in the format tbl_name=column_list. See ''GROUPING''.

## CUSTOMIZING

You can customize innotop a great deal. For example, you can:

- Choose which tables to display, and in what order.
- Choose which columns are in those tables, and create new columns.
- Filter which rows display with built-in filters, user-defined filters, and quick-filters.
- Sort the rows to put important data first or group together related rows.
- Highlight rows with color.
- Customize the alignment, width, and formatting of columns, and apply transformations to columns to extract parts of their values or format the values as you wish (for example, shortening large numbers to familiar units).
- Design your own expressions to extract and combine data as you need. This gives you unlimited flexibility.

All these and more are explained in the following sections.

**TABLES**

A table is what you'd expect: a collection of columns. It also has some other properties, such as a caption. Filters, sorting rules, and colorization rules belong to tables and are covered in later sections.

Internally, table meta-data is defined in a data structure called `%tbl_meta`. This hash holds all built-in table definitions, which contain a lot of default instructions to innotop. The meta-data includes the caption, a list of columns the user has customized, a list of columns, a list of visible columns, a list of filters, color rules, a sort-column list, sort direction, and some information about the table's data sources. Most of this is customizable via the table editor (see ''TABLE EDITOR'').

You can choose which tables to show by pressing the '$' key. See ''MODES'' and ''TABLES''.

The table life-cycle is as follows:

- Each table begins with a data source, which is an array of hashes. See below for details on data sources.
- Each element of the data source becomes a row in the final table.
- For each element in the data source, innotop extracts values from the source and creates a row. This row is another hash, which later steps will refer to as `$set`. The values innotop extracts are determined by the table's columns. Each column has an extraction subroutine, compiled from an expression (see ''EXPRESSIONS''). The resulting row is a hash whose keys are named the same as the column name.
- innotop filters the rows, removing those that don't need to be displayed. See ''FILTERS''.
- innotop sorts the rows. See ''SORTING''.
- innotop groups the rows together, if specified. See ''GROUPING''.
- innotop colorizes the rows. See ''COLORS''.
- innotop transforms the column values in each row. See ''TRANSFORMATIONS''.
- innotop optionally pivots the rows (see ''PIVOTING''), then filters and sorts them.
- innotop formats and justifies the rows as a table. During this step, innotop applies further formatting to the column values, including alignment, maximum and minimum widths. innotop also does final error checking to ensure there are no crashes due to undefined values. innotop then adds a caption if specified, and the table is ready to print.

The lifecycle is slightly different if the table is pivoted, as noted above. To clarify, if the table is pivoted, the process is extract, group, transform, pivot, filter, sort, create. If it's not pivoted, the process is extract, filter, sort, group, color, transform, create. This slightly convoluted process doesn't map all that well to SQL, but pivoting complicates things pretty thoroughly. Roughly speaking, filtering and sorting happen as late as needed to effect the final result as you might expect, but as early as possible for efficiency.

Each built-in table is described below:

adaptive_hash_index
: Displays data about InnoDB's adaptive hash index. Data source: ''STATUS_VARIABLES''.

buffer_pool
: Displays data about InnoDB's buffer pool. Data source: ''STATUS_VARIABLES''.

cmd_summary
: Displays weighted status variables. Data source: ''STATUS_VARIABLES''.

deadlock_locks
: Shows which locks were held and waited for by the last detected deadlock. Data source: ''DEADLOCK_LOCKS''.

deadlock_transactions
: Shows transactions involved in the last detected deadlock. Data source: ‘‘DEADLOCK_TRANSACTIONS’’.

explain
: Shows the output of EXPLAIN. Data source: ‘‘EXPLAIN’’.

file_io_misc
: Displays data about InnoDB’s file and I/O operations. Data source: ‘‘STATUS_VARIABLES’’.

fk_error
: Displays various data about InnoDB’s last foreign key error. Data source: ‘‘STATUS_VARIABLES’’.

innodb_locks
: Displays InnoDB locks. Data source: ‘‘INNODB_LOCKS’’.

innodb_transactions
: Displays data about InnoDB’s current transactions. Data source: ‘‘INNODB_TRANSACTIONS’’.

insert_buffers
: Displays data about InnoDB’s insert buffer. Data source: ‘‘STATUS_VARIABLES’’.

io_threads
: Displays data about InnoDB’s I/O threads. Data source: ‘‘IO_THREADS’’.

log_statistics
: Displays data about InnoDB’s logging system. Data source: ‘‘STATUS_VARIABLES’’.

master_status
: Displays replication master status. Data source: ‘‘STATUS_VARIABLES’’.

open_tables
: Displays open tables. Data source: ‘‘OPEN_TABLES’’.

page_statistics
: Displays InnoDB page statistics. Data source: ‘‘STATUS_VARIABLES’’.

pending_io
: Displays InnoDB pending I/O operations. Data source: ‘‘STATUS_VARIABLES’’.

processlist
: Displays current MySQL processes (threads/connections). Data source: ‘‘PROCESSLIST’’.

q_header
: Displays various status values. Data source: ‘‘STATUS_VARIABLES’’.

row_operation_misc
: Displays data about InnoDB’s row operations. Data source: ‘‘STATUS_VARIABLES’’.

row_operations
: Displays data about InnoDB’s row operations. Data source: ‘‘STATUS_VARIABLES’’.

semaphores
: Displays data about InnoDB’s semaphores and mutexes. Data source: ‘‘STATUS_VARIABLES’’.

slave_io_status
: Displays data about the slave I/O thread. Data source: ‘‘STATUS_VARIABLES’’.

slave_sql_status
: Displays data about the slave SQL thread. Data source: ‘‘STATUS_VARIABLES’’.

t_header
Displays various InnoDB status values. Data source: ''STATUS_VARIABLES''.

var_status
Displays user-configurable data. Data source: ''STATUS_VARIABLES''.

wait_array
Displays data about InnoDB's OS wait array. Data source: ''OS_WAIT_ARRAY''.

**COLUMNS**

Columns belong to tables. You can choose a table's columns by pressing the '^' key, which starts the ''TABLE EDITOR'' and lets you choose and edit columns. Pressing 'e' from within the table editor lets you edit the column's properties:

- hdr: a column header. This appears in the first row of the table.
- just: justification. '-' means left-justified and '' means right-justified, just as with printf formatting codes (not a coincidence).
- dec: whether to further align the column on the decimal point.
- num: whether the column is numeric. This affects how values are sorted (lexically or numerically).
- label: a small note about the column, which appears in dialogs that help the user choose columns.
- src: an expression that innotop uses to extract the column's data from its source (see ''DATA SOURCES''). See ''EXPRESSIONS'' for more on expressions.
- minw: specifies a minimum display width. This helps stabilize the display, which makes it easier to read if the data is changing frequently.
- maxw: similar to minw.
- trans: a list of column transformations. See ''TRANSFORMATIONS''.
- agg: an aggregate function. See ''GROUPING''. The default is ''first''.
- aggonly: controls whether the column only shows when grouping is enabled on the table (see ''GROUPING''). By default, this is disabled. This means columns will always be shown by default, whether grouping is enabled or not. If a column's aggonly is set true, the column will appear when you toggle grouping on the table. Several columns are set this way, such as the count column on ''processlist'' and ''innodb_transactions'', so you don't see a count when the grouping isn't enabled, but you do when it is.

**FILTERS**

Filters remove rows from the display. They behave much like a WHERE clause in SQL. innotop has several built-in filters, which remove irrelevant information like inactive queries, but you can define your own as well. innotop also lets you create quick-filters, which do not get saved to the configuration file, and are just an easy way to quickly view only some rows.

You can enable or disable a filter on any table. Press the '%' key (mnemonic: % looks kind of like a line being filtered between two circles) and choose which table you want to filter, if asked. You'll then see a list of possible filters and a list of filters currently enabled for that table. Type the names of filters you want to apply and press Enter.

*USER-DEFINED FILTERS*

If you type a name that doesn't exist, innotop will prompt you to create the filter. Filters are easy to create if you know Perl, and not hard if you don't. What you're doing is creating a subroutine that returns true if the row should be displayed. The row is a hash reference passed to your subroutine as `$set`.

For example, imagine you want to filter the processlist table so you only see queries that have been running more than five minutes. Type a new name for your filter, and when prompted for

the subroutine body, press TAB to initiate your terminal's auto-completion. You'll see the names of the columns in the ''processlist'' table (innotop generally tries to help you with auto-completion lists). You want to filter on the 'time' column. Type the text ''$set->{time} > 300'' to return true when the query is more than five minutes old. That's all you need to do.

In other words, the code you're typing is surrounded by an implicit context, which looks like this:

```
sub filter {
my ( $set ) = @_;
# YOUR CODE HERE
}
```

If your filter doesn't work, or if something else suddenly behaves differently, you might have made an error in your filter, and innotop is silently catching the error. Try enabling ''debug'' to make innotop throw an error instead.

*QUICK-FILTERS*

innotop's quick-filters are a shortcut to create a temporary filter that doesn't persist when you restart innotop. To create a quick-filter, press the '/' key. innotop will prompt you for the column name and filter text. Again, you can use auto-completion on column names. The filter text can be just the text you want to ''search for.'' For example, to filter the ''processlist'' table on queries that refer to the products table, type '/' and then 'info product'.

The filter text can actually be any Perl regular expression, but of course a literal string like 'product' works fine as a regular expression.

Behind the scenes innotop compiles the quick-filter into a specially tagged filter that is otherwise like any other filter. It just isn't saved to the configuration file.

To clear quick-filters, press the '' key and innotop will clear them all at once.

**SORTING**

innotop has sensible built-in defaults to sort the most important rows to the top of the table. Like anything else in innotop, you can customize how any table is sorted.

To start the sort dialog, start the ''TABLE EDITOR'' with the '^' key, choose a table if necessary, and press the 's' key. You'll see a list of columns you can use in the sort expression and the current sort expression, if any. Enter a list of columns by which you want to sort and press Enter. If you want to reverse sort, prefix the column name with a minus sign. For example, if you want to sort by column a ascending, then column b descending, type 'a -b'. You can also explicitly add a + in front of columns you want to sort ascending, but it's not required.

Some modes have keys mapped to open this dialog directly, and to quickly reverse sort direction. Press '?' as usual to see which keys are mapped in any mode.

**GROUPING**

innotop can group, or aggregate, rows together (the terms are used interchangeably). This is quite similar to an SQL GROUP BY clause. You can specify to group on certain columns, or if you don't specify any, the entire set of rows is treated as one group. This is quite like SQL so far, but unlike SQL, you can also select un-grouped columns. innotop actually aggregates every column. If you don't explicitly specify a grouping function, the default is 'first'. This is basically a convenience so you don't have to specify an aggregate function for every column you want in the result.

You can quickly toggle grouping on a table with the '=' key, which toggles its aggregate property. This property doesn't persist to the config file.

The columns by which the table is grouped are specified in its group_by property. When you turn grouping on, innotop places the group_by columns at the far left of the table, even if they're not supposed to be visible. The rest of the visible columns appear in order after them.

Two tables have default group_by lists and a count column built in: ''processlist'' and ''innodb_transactions''. The grouping is by connection and status, so you can quickly see how

many queries or transactions are in a given status on each server you're monitoring. The time columns are aggregated as a sum; other columns are left at the default 'first' aggregation.

By default, the table shown in "S: Variables & Status" mode also uses grouping so you can monitor variables and status across many servers. The default aggregation function in this mode is 'avg'.

Valid grouping functions are defined in the `%agg_funcs` hash. They include

first
: Returns the first element in the group.

count
: Returns the number of elements in the group, including undefined elements, much like SQL's COUNT(*).

avg
: Returns the average of defined elements in the group.

sum
: Returns the sum of elements in the group.

Here's an example of grouping at work. Suppose you have a very busy server with hundreds of open connections, and you want to see how many connections are in what status. Using the built-in grouping rules, you can press 'Q' to enter "Q: Query List" mode. Press '=' to toggle grouping (if necessary, select the "processlist" table when prompted).

Your display might now look like the following:

```
Query List (? for help) localhost, 32:33, 0.11 QPS, 1 thd, 5.0.38-log

CXN Cmd Cnt ID User Host Time Query
localhost Query 49 12933 webusr localhost 19:38 SELECT * FROM
localhost Sending Da 23 2383 webusr localhost 12:43 SELECT col1,
localhost Sleep 120 140 webusr localhost 5:18:12
localhost Statistics 12 19213 webusr localhost 01:19 SELECT * FROM
```

That's actually quite a worrisome picture. You've got a lot of idle connections (Sleep), and some connections executing queries (Query and Sending Data). That's okay, but you also have a lot in Statistics status, collectively spending over a minute. That means the query optimizer is having a really hard time optimizing your statements. Something is wrong; it should normally take milliseconds to optimize queries. You might not have seen this pattern if you didn't look at your connections in aggregate. (This is a made-up example, but it can happen in real life).

**PIVOTING**

innotop can pivot a table for more compact display, similar to a Pivot Table in a spreadsheet (also known as a crosstab). Pivoting a table makes columns into rows. Assume you start with this table:

```
foo bar
=== ===
1 3
2 4
```

After pivoting, the table will look like this:

```
name set0 set1
==== ==== ====
foo 1 2
bar 3 4
```

To get reasonable results, you might need to group as well as pivoting. innotop currently does this for "S: Variables & Status" mode.

## COLORS

By default, innotop highlights rows with color so you can see at a glance which rows are more important. You can customize the colorization rules and add your own to any table. Open the table editor with the '^' key, choose a table if needed, and press 'o' to open the color editor dialog.

The color editor dialog displays the rules applied to the table, in the order they are evaluated. Each row is evaluated against each rule to see if the rule matches the row; if it does, the row gets the specified color, and no further rules are evaluated. The rules look like the following:

```
state eq Locked black on_red
cmd eq Sleep white
user eq system user white
cmd eq Connect white
cmd eq Binlog Dump white
time > 600 red
time > 120 yellow
time > 60 green
time > 30 cyan
```

This is the default rule set for the ''processlist'' table. In order of priority, these rules make locked queries black on a red background, ''gray out'' connections from replication and sleeping queries, and make queries turn from cyan to red as they run longer.

(For some reason, the ANSI color code ''white'' is actually a light gray. Your terminal's display may vary; experiment to find colors you like).

You can use keystrokes to move the rules up and down, which re-orders their priority. You can also delete rules and add new ones. If you add a new rule, innotop prompts you for the column, an operator for the comparison, a value against which to compare the column, and a color to assign if the rule matches. There is auto-completion and prompting at each step.

The value in the third step needs to be correctly quoted. innotop does not try to quote the value because it doesn't know whether it should treat the value as a string or a number. If you want to compare the column against a string, as for example in the first rule above, you should enter 'Locked' surrounded by quotes. If you get an error message about a bareword, you probably should have quoted something.

## EXPRESSIONS

Expressions are at the core of how innotop works, and are what enables you to extend innotop as you wish. Recall the table lifecycle explained in ''TABLES''. Expressions are used in the earliest step, where it extracts values from a data source to form rows.

It does this by calling a subroutine for each column, passing it the source data set, a set of current values, and a set of previous values. These are all needed so the subroutine can calculate things like the difference between this tick and the previous tick.

The subroutines that extract the data from the set are compiled from expressions. This gives significantly more power than just naming the values to fill the columns, because it allows the column's value to be calculated from whatever data is necessary, but avoids the need to write complicated and lengthy Perl code.

innotop begins with a string of text that can look as simple as a value's name or as complicated as a full-fledged Perl expression. It looks at each 'bareword' token in the string and decides whether it's supposed to be a key into the `$set` hash. A bareword is an unquoted value that isn't already surrounded by code-ish things like dollar signs or curly brackets. If innotop decides that the bareword isn't a function or other valid Perl code, it converts it into a hash access. After the whole string is processed, innotop compiles a subroutine, like this:

```
sub compute_column_value {
my ( $set, $cur, $pre ) = @_;
my $val = # EXPANDED STRING GOES HERE
return $val;
}
```

Here's a concrete example, taken from the header table ''q_header'' in ''Q: Query List'' mode. This expression calculates the qps, or Queries Per Second, column's values, from the values returned by SHOW STATUS:

```
Questions/Uptime_hires
```

innotop decides both words are barewords, and transforms this expression into the following Perl code:

```
$set->{Questions}/$set->{Uptime_hires}
```

When surrounded by the rest of the subroutine's code, this is executable Perl that calculates a high-resolution queries-per-second value.

The arguments to the subroutine are named `$set`, `$cur`, and `$pre`. In most cases, `$set` and `$cur` will be the same values. However, if ''status_inc'' is set, `$cur` will not be the same as `$set`, because `$set` will already contain values that are the incremental difference between `$cur` and `$pre`.

Every column in innotop is computed by subroutines compiled in the same fashion. There is no difference between innotop's built-in columns and user-defined columns. This keeps things consistent and predictable.

## TRANSFORMATIONS

Transformations change how a value is rendered. For example, they can take a number of seconds and display it in H:M:S format. The following transformations are defined:

commify
: Adds commas to large numbers every three decimal places.

dulint_to_int
: Accepts two unsigned integers and converts them into a single longlong. This is useful for certain operations with InnoDB, which uses two integers as transaction identifiers, for example.

no_ctrl_char
: Removes quoted control characters from the value. This is affected by the ''charset'' configuration variable.

  This transformation only operates within quoted strings, for example, values to a SET clause in an UPDATE statement. It will not alter the UPDATE statement, but will collapse the quoted string to [BINARY] or [TEXT], depending on the charset.

percent
: Converts a number to a percentage by multiplying it by two, formatting it with ''num_digits'' digits after the decimal point, and optionally adding a percent sign (see ''show_percent'').

secs_to_time
: Formats a number of seconds as time in days+hours:minutes:seconds format.

set_precision
: Formats numbers with ''num_digits'' number of digits after the decimal point.

shorten
: Formats a number as a unit of 1024 (k/M/G/T) and with ''num_digits'' number of digits after the decimal point.

**TABLE EDITOR**

The innotop table editor lets you customize tables with keystrokes. You start the table editor with the '^' key. If there's more than one table on the screen, it will prompt you to choose one of them. Once you do, innotop will show you something like this:

```
Editing table definition for Buffer Pool. Press ? for help, q to quit.

name hdr label src
cxn CXN Connection from which cxn
buf_pool_size Size Buffer pool size IB_bp_buf_poo
buf_free Free Bufs Buffers free in the b IB_bp_buf_fre
pages_total Pages Pages total IB_bp_pages_t
pages_modified Dirty Pages Pages modified (dirty IB_bp_pages_m
buf_pool_hit_rate Hit Rate Buffer pool hit rate IB_bp_buf_poo
total_mem_alloc Memory Total memory allocate IB_bp_total_m
add_pool_alloc Add'l Pool Additonal pool alloca IB_bp_add_poo
```

The first line shows which table you're editing, and reminds you again to press '?' for a list of key mappings. The rest is a tabular representation of the table's columns, because that's likely what you're trying to edit. However, you can edit more than just the table's columns; this screen can start the filter editor, color rule editor, and more.

Each row in the display shows a single column in the table you're editing, along with a couple of its properties such as its header and source expression (see ''EXPRESSIONS'').

The key mappings are Vim-style, as in many other places. Pressing 'j' and 'k' moves the highlight up or down. You can then (d)elete or (e)dit the highlighted column. You can also (a)dd a column to the table. This actually just activates one of the columns already defined for the table; it prompts you to choose from among the columns available but not currently displayed. Finally, you can re-order the columns with the '+' and '-' keys.

You can do more than just edit the columns with the table editor, you can also edit other properties, such as the table's sort expression and group-by expression. Press '?' to see the full list, of course.

If you want to really customize and create your own column, as opposed to just activating a built-in one that's not currently displayed, press the (n)ew key, and innotop will prompt you for the information it needs:

- The column name: this needs to be a word without any funny characters, e.g. just letters, numbers and underscores.
- The column header: this is the label that appears at the top of the column, in the table header. This can have spaces and funny characters, but be careful not to make it too wide and waste space on-screen.
- The column's data source: this is an expression that determines what data from the source (see ''TABLES'') innotop will put into the column. This can just be the name of an item in the source, or it can be a more complex expression, as described in ''EXPRESSIONS''.

Once you've entered the required data, your table has a new column. There is no difference between this column and the built-in ones; it can have all the same properties and behaviors. innotop will write the column's definition to the configuration file, so it will persist across sessions.

Here's an example: suppose you want to track how many times your slaves have retried transactions. According to the MySQL manual, the Slave_retried_transactions status variable gives you that data: ''The total number of times since startup that the replication slave SQL thread has retried transactions. This variable was added in version 5.0.4.'' This is appropriate to add to the ''slave_sql_status'' table.

To add the column, switch to the replication-monitoring mode with the 'M' key, and press the '^'

key to start the table editor. When prompted, choose slave_sql_status as the table, then press 'n' to create the column. Type 'retries' as the column name, 'Retries' as the column header, and 'Slave_retried_transactions' as the source. Now the column is created, and you see the table editor screen again. Press 'q' to exit the table editor, and you'll see your column at the end of the table.

## VARIABLE SETS

Variable sets are used in ''S: Variables & Status'' mode to define more easily what variables you want to monitor. Behind the scenes they are compiled to a list of expressions, and then into a column list so they can be treated just like columns in any other table, in terms of data extraction and transformations. However, you're protected from the tedious details by a syntax that ought to feel very natural to you: a SQL SELECT list.

The data source for variable sets, and indeed the entire S mode, is the combination of SHOW STATUS, SHOW VARIABLES, and SHOW INNODB STATUS. Imagine that you had a huge table with one column per variable returned from those statements. That's the data source for variable sets. You can now query this data source just like you'd expect. For example:

```
Questions, Uptime, Questions/Uptime as QPS
```

Behind the scenes innotop will split that variable set into three expressions, compile them and turn them into a table definition, then extract as usual. This becomes a ''variable set,'' or a ''list of variables you want to monitor.''

innotop lets you name and save your variable sets, and writes them to the configuration file. You can choose which variable set you want to see with the 'c' key, or activate the next and previous sets with the '>' and '<' keys. There are many built-in variable sets as well, which should give you a good start for creating your own. Press 'e' to edit the current variable set, or just to see how it's defined. To create a new one, just press 'c' and type its name.

You may want to use some of the functions listed in ''TRANSFORMATIONS'' to help format the results. In particular, ''set_precision'' is often useful to limit the number of digits you see. Extending the above example, here's how:

```
Questions, Uptime, set_precision(Questions/Uptime) as QPS
```

Actually, this still needs a little more work. If your ''interval'' is less than one second, you might be dividing by zero because Uptime is incremental in this mode by default. Instead, use Uptime_hires:

```
Questions, Uptime, set_precision(Questions/Uptime_hires) as QPS
```

This example is simple, but it shows how easy it is to choose which variables you want to monitor.

## PLUGINS

innotop has a simple but powerful plugin mechanism by which you can extend or modify its existing functionality, and add new functionality. innotop's plugin functionality is event-based: plugins register themselves to be called when events happen. They then have a chance to influence the event.

An innotop plugin is a Perl module placed in innotop's ''plugin_dir'' directory. On UNIX systems, you can place a symbolic link to the module instead of putting the actual file there. innotop automatically discovers the file. If there is a corresponding entry in the ''plugins'' configuration file section, innotop loads and activates the plugin.

The module must conform to innotop's plugin interface. Additionally, the source code of the module must be written in such a way that innotop can inspect the file and determine the package name and description.

### Package Source Convention

innotop inspects the plugin module's source to determine the Perl package name. It looks for a line of the form ''package Foo;'' and if found, considers the plugin's package name to be Foo. Of course the package name can be a valid Perl package name, with double semicolons and so on.

It also looks for a description in the source code, to make the plugin editor more human-friendly. The description is a comment line of the form ''# description: Foo'', where ''Foo'' is the text innotop will consider to be the plugin's description.

**Plugin Interface**

The innotop plugin interface is quite simple: innotop expects the plugin to be an object-oriented module it can call certain methods on. The methods are

new(%variables)

This is the plugin's constructor. It is passed a hash of innotop's variables, which it can manipulate (see ''Plugin Variables''). It must return a reference to the newly created plugin object.

At construction time, innotop has only loaded the general configuration and created the default built-in variables with their default contents (which is quite a lot). Therefore, the state of the program is exactly as in the innotop source code, plus the configuration variables from the ''general'' section in the config file.

If your plugin manipulates the variables, it is changing global data, which is shared by innotop and all plugins. Plugins are loaded in the order they're listed in the config file. Your plugin may load before or after another plugin, so there is a potential for conflict or interaction between plugins if they modify data other plugins use or modify.

*register_for_events()*

This method must return a list of events in which the plugin is interested, if any. See ''Plugin Events'' for the defined events. If the plugin returns an event that's not defined, the event is ignored.

event handlers

The plugin must implement a method named the same as each event for which it has registered. In other words, if the plugin returns qw(foo bar) from *register_for_events()*, it must have *foo()* and *bar()* methods. These methods are callbacks for the events. See ''Plugin Events'' for more details about each event.

**Plugin Variables**

The plugin's constructor is passed a hash of innotop's variables, which it can manipulate. It is probably a good idea if the plugin object saves a copy of it for later use. The variables are defined in the innotop variable `%pluggable_vars`, and are as follows:

action_for

A hashref of key mappings. These are innotop's global hot-keys.

agg_funcs

A hashref of functions that can be used for grouping. See ''GROUPING''.

config

The global configuration hash.

connections

A hashref of connection specifications. These are just specifications of how to connect to a server.

dbhs

A hashref of innotop's database connections. These are actual DBI connection objects.

filters

A hashref of filters applied to table rows. See ''FILTERS'' for more.

modes

A hashref of modes. See ''MODES'' for more.

server_groups
: A hashref of server groups. See ‘‘SERVER GROUPS’’.

tbl_meta
: A hashref of innotop's table meta-data, with one entry per table (see ‘‘TABLES’’ for more information).

trans_funcs
: A hashref of transformation functions. See ‘‘TRANSFORMATIONS’’.

var_sets
: A hashref of variable sets. See ‘‘VARIABLE SETS’’.

**Plugin Events**

Each event is defined somewhere in the innotop source code. When innotop runs that code, it executes the callback function for each plugin that expressed its interest in the event. innotop passes some data for each event. The events are defined in the `%event_listener_for` variable, and are as follows:

extract_values($set, `$cur`, `$pre`, `$tbl`)
: This event occurs inside the function that extracts values from a data source. The arguments are the set of values, the current values, the previous values, and the table name.

set_to_tbl
: Events are defined at many places in this subroutine, which is responsible for turning an arrayref of hashrefs into an arrayref of lines that can be printed to the screen. The events all pass the same data: an arrayref of rows and the name of the table being created. The events are set_to_tbl_pre_filter, set_to_tbl_pre_sort,set_to_tbl_pre_group, set_to_tbl_pre_colorize, set_to_tbl_pre_transform, set_to_tbl_pre_pivot, set_to_tbl_pre_create, set_to_tbl_post_create.

draw_screen($lines)
: This event occurs inside the subroutine that prints the lines to the screen. `$lines` is an arrayref of strings.

**Simple Plugin Example**

The easiest way to explain the plugin functionality is probably with a simple example. The following module adds a column to the beginning of every table and sets its value to 1.

```
use strict;
use warnings FATAL => 'all';

package Innotop::Plugin::Example;
# description: Adds an 'example' column to every table

sub new {
my ( $class, %vars ) = @_;
# Store reference to innotop's variables in $self
my $self = bless { %vars }, $class;

# Design the example column
my $col = {
hdr => 'Example',
just => '',
dec => 0,
num => 1,
label => 'Example',
src => 'example', # Get data from this column in the data source
tbl => '',
trans => [],
};
```

```
# Add the column to every table.
my $tbl_meta = $vars{tbl_meta};
foreach my $tbl ( values %$tbl_meta ) {
# Add the column to the list of defined columns
$tbl->{cols}->{example} = $col;
# Add the column to the list of visible columns
unshift @{$tbl->{visible}}, 'example';
}

# Be sure to return a reference to the object.
return $self;
}

# I'd like to be called when a data set is being rendered into a table, please.
sub register_for_events {
my ( $self ) = @_;
return qw(set_to_tbl_pre_filter);
}

# This method will be called when the event fires.
sub set_to_tbl_pre_filter {
my ( $self, $rows, $tbl ) = @_;
# Set the example column's data source to the value 1.
foreach my $row ( @$rows ) {
$row->{example} = 1;
}
}

1;
```

**Plugin Editor**

The plugin editor lets you view the plugins innotop discovered and activate or deactivate them. Start the editor by pressing $ to start the configuration editor from any mode. Press the 'p' key to start the plugin editor. You'll see a list of plugins innotop discovered. You can use the 'j' and 'k' keys to move the highlight to the desired one, then press the * key to toggle it active or inactive. Exit the editor and restart innotop for the changes to take effect.

## SQL STATEMENTS

innotop uses a limited set of SQL statements to retrieve data from MySQL for display. The statements are customized depending on the server version against which they are executed; for example, on MySQL 5 and newer, INNODB_STATUS executes ''SHOW ENGINE INNODB STATUS'', while on earlier versions it executes ''SHOW INNODB STATUS''. The statements are as follows:

```
Statement SQL executed
================== ==============================
INNODB_STATUS SHOW [ENGINE] INNODB STATUS
KILL_CONNECTION KILL
KILL_QUERY KILL QUERY
OPEN_TABLES SHOW OPEN TABLES
PROCESSLIST SHOW FULL PROCESSLIST
SHOW_MASTER_LOGS SHOW MASTER LOGS
SHOW_MASTER_STATUS SHOW MASTER STATUS
SHOW_SLAVE_STATUS SHOW SLAVE STATUS
SHOW_STATUS SHOW [GLOBAL] STATUS
SHOW_VARIABLES SHOW [GLOBAL] VARIABLES
```

## DATA SOURCES

Each time innotop extracts values to create a table (see ''EXPRESSIONS'' and ''TABLES''), it does so from a particular data source. Largely because of the complex data extracted from SHOW INNODB STATUS, this is slightly messy. SHOW INNODB STATUS contains a mixture of single values and repeated values that form nested data sets.

Whenever innotop fetches data from MySQL, it adds two extra bits to each set: cxn and Uptime_hires. cxn is the name of the connection from which the data came. Uptime_hires is a high-resolution version of the server's Uptime status variable, which is important if your ''interval'' setting is sub-second.

Here are the kinds of data sources from which data is extracted:

STATUS_VARIABLES
: This is the broadest category, into which the most kinds of data fall. It begins with the combination of SHOW STATUS and SHOW VARIABLES, but other sources may be included as needed, for example, SHOW MASTER STATUS and SHOW SLAVE STATUS, as well as many of the non-repeated values from SHOW INNODB STATUS.

DEADLOCK_LOCKS
: This data is extracted from the transaction list in the LATEST DETECTED DEADLOCK section of SHOW INNODB STATUS. It is nested two levels deep: transactions, then locks.

DEADLOCK_TRANSACTIONS
: This data is from the transaction list in the LATEST DETECTED DEADLOCK section of SHOW INNODB STATUS. It is nested one level deep.

EXPLAIN
: This data is from the result set returned by EXPLAIN.

INNODB_TRANSACTIONS
: This data is from the TRANSACTIONS section of SHOW INNODB STATUS.

IO_THREADS
: This data is from the list of threads in the the FILE I/O section of SHOW INNODB STATUS.

INNODB_LOCKS
: This data is from the TRANSACTIONS section of SHOW INNODB STATUS and is nested two levels deep.

OPEN_TABLES
: This data is from SHOW OPEN TABLES.

PROCESSLIST
: This data is from SHOW FULL PROCESSLIST.

OS_WAIT_ARRAY
: This data is from the SEMAPHORES section of SHOW INNODB STATUS and is nested one level deep. It comes from the lines that look like this:

```
--Thread 1568861104 has waited at btr0cur.c line 424 ....
```

## MYSQL PRIVILEGES

- You must connect to MySQL as a user who has the SUPER privilege for many of the functions.
- If you don't have the SUPER privilege, you can still run some functions, but you won't necessarily see all the same data.
- You need the PROCESS privilege to see the list of currently running queries in Q mode.
- You need special privileges to start and stop slave servers.
- You need appropriate privileges to create and drop the deadlock tables if needed (see "SERVER CONNECTIONS").

## SYSTEM REQUIREMENTS

You need Perl to run innotop, of course. You also need a few Perl modules: DBI, DBD::mysql, Term::ReadKey, and Time::HiRes. These should be included with most Perl distributions, but in case they are not, I recommend using versions distributed with your operating system or Perl distribution, not from CPAN. Term::ReadKey in particular has been known to cause problems if installed from CPAN.

If you have Term::ANSIColor, innotop will use it to format headers more readably and compactly. (Under Microsoft Windows, you also need Win32::Console::ANSI for terminal formatting codes to be honored). If you install Term::ReadLine, preferably Term::ReadLine::Gnu, you'll get nice auto-completion support.

I run innotop on Gentoo GNU/Linux, Debian and Ubuntu, and I've had feedback from people successfully running it on Red Hat, CentOS, Solaris, and Mac OSX. I don't see any reason why it won't work on other UNIX-ish operating systems, but I don't know for sure. It also runs on Windows under ActivePerl without problem.

innotop has been used on MySQL versions 3.23.58, 4.0.27, 4.1.0, 4.1.22, 5.0.26, 5.1.15, and 5.2.3. If it doesn't run correctly for you, that is a bug that should be reported.

## FILES

`$HOMEDIR`/.innotop and/or /etc/innotop are used to store configuration information. Files include the configuration file innotop.conf, the core_dump file which contains verbose error messages if "debug" is enabled, and the plugins/ subdirectory.

## GLOSSARY OF TERMS

tick

A tick is a refresh event, when innotop re-fetches data from connections and displays it.

## ACKNOWLEDGEMENTS

The following people and organizations are acknowledged for various reasons. Hopefully no one has been forgotten.

Allen K. Smith, Aurimas Mikalauskas, Bartosz Fenski, Brian Miezejewski, Christian Hammers, Cyril Scetbon, Dane Miller, David Multer, Dr. Frank Ullrich, Giuseppe Maxia, Google.com Site Reliability Engineers, Google Code, Jan Pieter Kunst, Jari Aalto, Jay Pipes, Jeremy Zawodny, Johan Idren, Kristian Kohntopp, Lenz Grimmer, Maciej Dobrzanski, Michiel Betel, MySQL AB, Paul McCullagh, Sebastien Estienne, Sourceforge.net, Steven Kreuzer, The Gentoo MySQL Team, Trevor Price, Yaar Schnitman, and probably more people that have not been included.

(If your name has been misspelled, it's probably out of fear of putting international characters into this documentation; earlier versions of Perl might not be able to compile it then).

## COPYRIGHT, LICENSE AND WARRANTY

Execute innotop and press '!' to see this information at any time.

## AUTHOR

Originally written by Baron Schwartz; currently maintained by Aaron Racine.

## BUGS

You can report bugs, ask for improvements, and get other help and support at <http://code.google.com/p/innotop/>.

There are mailing lists, a source code browser, a bug tracker, etc. Please use these instead of contacting the maintainer or author directly, as it makes our job easier and benefits others if the discussions are permanent and public. Of course, if you need to contact us in private, please do.