Lograptor Documentation

Release 1.2.3

Davide Brunato

Sep 04, 2018

CONTENTS

Lograptor is a GREP-like tool which provides a command-line interface for processing system logs.

Regular expression searches can be performed together with filtering rules and scope delimitation options. Each search run can be sent to an output channel (stdout, e-mail, file) and can produces a customizable report.

The program can parse logs written in RFC 3164 and RFC 5424 formats. Lograptor requires Python >= 2.7, and is provided with a base configuration for a set of well known applications. You can easily add new applications or new rules to match other unparsed logs.

The project uses parts of Epylog under LGPL terms with author's permission.

CHAPTER

ONE

INSTALLATION

1.1 Installing from package

Lograptor is packaged with Python's *wheel* format on PyPI (RPM/DEB packages formats are not maintained anymore) so you can install it using *pip*. If your have root access you can do a system wide installation:

```
sudo pip install lograptor
```

In this case the sources are installed under Python's packages directory (eg. */usr/lib/python3.6/site-packages/*) and the data files (man, docs and configuration files) are installed under standard POSIX paths (*/usr/share* and */etc*).

For an installation at user level run:

```
pip install --user lograptor
```

In this case the files are written into *~/.local/* directory.

You can also install the package into a virtual environment (using *virtualenv* or *pyvenv*). In this case the configuration file have to be referenced explicitly, using *–conf* option, or the configuration files have to be copied to one of the program's default locations, that are in order:

```
./lograptor.conf
~/.config/lograptor/lograptor.conf
~/.local/etc/lograptor/lograptor.conf
/etc/lograptor/lograptor.conf
<package source location>/config/lograptor.conf
```

1.2 Installing from source

For installing from the source you also need Python's setuptools, that is generically available on almost all Linux distributions or however is packaged on PyPI.

With *setuptools* installed clone the git repository, choosing one of those commands:

```
git clone https://github.com/brunato/lograptor
git clone git://github.com/brunato/lograptor.git
```

or download the zip archive from the site and extract the content to a folder. Then go into the lograptor's source base directory and type:

```
python setup.py install
```

To install also the configuration and documentation files run:

```
python setup.py install_data
```

CHAPTER

TWO

THE LOGRAPTOR COMMAND

2.1 SYNOPSIS

```
lograptor [options] PATTERN [FILE ...]
lograptor [options] [-e PATTERN | -f PATTERNS_FILE ] [FILE ...]
```

2.2 DESCRIPTION

lograptor is a search tool for system logs saved with legacy BSD syslog format (RFC 3164) or IETF syslog format (RFC 5424).

It's developed as a compact and configurable GREP-like tool, usable for raw or refined searches and to create customizable reports on system logs. The application mixes regex pattern matching search with scope delimiters and a configurable set of filters. You can configure additional application pattern rules using the classical regexp syntax. lograptor can also produce and publish reports in various formats. Reporting can be automated using cron.

For lograptor's configuration see lograptor-conf(5).

For more information on adding and configuring applications see lograptor-apps(5).

2.3 OPTIONS

2.3.1 Positional Arguments

[FILE ...]
Input files. Each argument can be a file path or a glob pathname. A "-" stands for standard input. If no arguments are given then processes all the files included within the scope of the selected applications.

2.3.2 General Options

--conf FILE
Use a specific configuration file. For default try to find and use a *lograptor.conf* file located in the current directory, in the *~/.config/lograptor/* directory, in the *~/.local/etc/lograptor/* directory or in the */etc/lograptor/* directory. If none of them exist then uses the default configuration provided within the package into the subdirectory *config/*. If you call the program from the command line without other options and arguments a summary of configuration settings is dumped to stdout.

-d [0-4]
Logging level (default is 2, use 4 for debug). A level of 0 suppress also error messages about nonexistent or unreadable files.

-V, --version
Show program's version number and exit.

--help
Show an help page about program options and exit.

2.3.3 Scope Selection

-a APP[,APP...], **--apps** APP[,APP...]
Process the log lines related to an application. An app name is valid when a configuration file is defined. For default all apps defined and enabled are processed.

--hosts HOSTNAME/IP[,HOSTNAME/IP...]
Process the log lines related to a comma separated list of hostnames and/or IP addresses. File path wildcards can be used for hostnames.

-F FIELD=PATTERN[,FIELD=PATTERN...], **--filter** FIELD=PATTERN[,FIELD=PATTERN...]
Process the log lines that match all the conditions for pattern rule's field values. The filters within a single option are applied with logical conjunction (AND). Multiple -F options are used with logical disjunction (OR).

--time HH:MM,HH:MM
Process the log lines related to a time range.

--date [YYYY]MMDD[,[YYYY]MMDD]
Restrict the search scope to a date or a date interval.

--last [hour|day|week|month|Nh|Nd|Nw|Nm]
Restrict the search scope to a previous time period.

2.3.4 Matcher Selection

-G, --ruled
Use patterns and application rules matching. This is the default.

-X, --unruled
Use patterns only. Application pattern rules are skipped. This option is incompatible with report and filtering options.

-U, --unparsed
Match the patterns but select the lines that don't match any application rule. This option is useful for finding anomalies and for application's rules debugging. This option is incompatible with filters (option -F).

2.3.5 Matching Control

-e PATTERN, **--regexp**=PATTERN
The search pattern. Use the option more times to specify multiple search patterns. Empty patterns are skipped.

-f FILE, **--file**=FILE
Obtain patterns from FILE, one per line. Blank lines are skipped. If this option is used multiple times or is combined with the -e (–regexp) option, search for all patterns given. An empty file contains zero patterns, and therefore matches nothing.

-i, --ignore-case
Ignore case distinctions in matching, so that characters that differ only in case match each other.

-v, --invert-match
Invert the sense of matching, to select non-matching lines.

-w, --word-regexp
Force PATTERN to match only whole words. The matching substring must either be at the beginning of the line, or preceded by a non-word constituent character. Similarly, it must be either at the end of the line

or followed by a non-word constituent character. Word-constituent characters are letters, digits, and the underscore.

2.3.6 General Output Control

--output CHANNEL[,CHANNEL...]
Send output to a comma separated list of channels. Channels have to be defined in the configuration file. For default the output is sent to *stdout* channel.

-c, --count
Suppress normal output; instead print a count of matching lines for each input file. With the -v/–invert-match option count non-matching lines.

--color [(auto|always|never)]
Use markers to highlight the matching strings. The colors are defined by the environment variable LOGRAPTOR_COLORS.

-L, --files-without-match
Print only names of FILEs containing no match.

-l, --files-with-match
Print only names of FILEs containing matches. The scanning will stop on the first match.

-m NUM, **--max-count** NUM
Stop reading a file after NUM matching lines. When -c/–count option is also used, lograptor does not output a count greater than NUM. When using -t/–thread option the limit is related to the number of threads and not to the number of lines matched.

-o, --only-matching
Print only the matched (non-empty) parts of a matching line, with each such part on a separate output line.

-q, --quiet
Quiet; do not write anything to standard output. Exit immediately with zero status if any match is found, even if an error was detected.

-s, --no-messages
Suppress error messages about nonexistent or unreadable files. Equivalent to -d 0.

2.3.7 Output Data Control

--report [NAME]
Produce a report at the end of processing. If NAME is omitted that use the *default* report defined in the lograptor configuration file.

--ip-lookup
Translate IP addresses to DNS names. Use a DNS local cache to improve the speed of the lookups and reduce the network service's load.

--uid-lookup
Translate UIDs to usernames. The configured local system authentication is used for lookups, so it must be inherent to the UIDs that have to be resolved.

--anonymize
Anonymize defined application rule's fields value. Translation tables are built in volatile memory for each run. The anonymous tokens have the format FILTER_NNN. This option overrides –ip-lookup and –uid-lookup options. WARNING: this is an experimental feature.

2.3.8 Output Line Prefix Control

-n, --line-number
Prefix each line of output with the line number within its input file.

-H, --with-filename
Print the file name for each match. This is the default when there is more than one file to search.

-h, --no-filename
Suppress the prefixing of file names on output. This is the default when there is only one file (or only standard input) to search.

2.3.9 Context Line Control

-T, --thread
The context is the log thread of the application. The thread rules defined in application configuration files are used.

-A NUM, **--after-context** NUM
Print NUM lines of trailing context after matching lines. Places a line containing a group separator (described under –group-separator option) between contiguous groups of matches. With the -o or –only-matching option, this has no effect and a warning is given.

-B NUM, **--before-context** NUM
Print NUM lines of leading context before matching lines. Places a line containing a group separator (described under –group-separator) between contiguous groups of matches. With the -o or –only-matching option, this has no effect and a warning is given.

-C NUM, **--context** NUM
Print NUM lines of output context. Places a line containing a group separator (described under –group-separator) between contiguous groups of matches. With the -o or –only-matching option, this has no effect and a warning is given.

--group-separator SEP
Use SEP as a group separator. By default SEP is double hyphen (–).

--no-group-separator
Use empty string as a group separator.

2.3.10 File and Directory Selection

-r, --recursive
Read all files under each directory, recursively, following symbolic links only if they are on the command line.

-R, --dereference-recursive
Read all files under each directory, recursively. Follow all symbolic links, unlike -r.

--exclude GLOB
Skip any file with a name suffix that matches the pattern GLOB, using wildcard matching; a name suffix is either the whole name, or any suffix starting after a / and before a +non-/. When searching recursively, skip any subfile whose base name matches GLOB; the base name is the part after the last /. A pattern can use *, ?, and [...] as wildcards, and to quote a wildcard or backslash character literally.

--exclude-from FILE
Skip files whose base name matches any of the file-name globs read from FILE (using wildcard matching as described under –exclude).

--exclude-dir DIR
Skip any command-line directory with a name suffix that matches the pattern GLOB. When searching recursively, skip any subdirectory whose base name matches GLOB. Ignore any redundant trailing slashes in GLOB.

--include GLOB
Search only files whose base name matches GLOB (using wildcard matching as described under –exclude).

2.4 FILES

```
/etc/lograptor/lograptor.conf
/etc/lograptor/conf.d/*.conf
/usr/bin/lograptor
```

2.5 AUTHORS

Davide Brunato <brunato@sissa.it>

2.6 SEE ALSO

lograptor.conf(5), lograptor-apps(5), lograptor-examples(5),

CHAPTER

THREE

LOGRAPTOR CONFIGURATION

3.1 CONFIGURATION FILE

lograptor.conf

lograptor looks at *./lograptor.conf*, *~/.config/lograptor/lograptor.conf*, *~/.local/etc/lograptor/lograptor.conf* or */etc/lograptor/lograptor.conf* for a configuration file, using the first file found, but you can use an specific configuration file using the `--conf` command line option.

3.2 DESCRIPTION

A lograptor configuration file uses the Python's ConfigParser format which provides a structure similar to Microsoft Windows INI files. A configuration file consists of sections and option entries. A section start with a ''[section]" header. Each section can have different `name=value` (`name: value` is also accepted) option entries, with continuations in the style of RFC 822 (see section 3.1.1, "LONG HEADER FIELDS"). Note that leading and trailing whitespaces are removed from values.

A configuration file for lograptor includes three fixed-named sections (*main*, *patterns* and *fields*) and at least one section for the default report (*default_report*). Other sections can be added in order to configure additional output channels or reports.

3.3 [main] SECTION

`confdir`
: This is where lograptor should look for apps configuration information, most notably, *conf.d* directory. See lograptor-apps(5) for more info on apps configuration.

`logdir`
: Where the system logs are located. Useful to shortening log path specification in application's configuration files.

`tmpdir`
: Where to create temporary directories and put temporary files. Note that log files can grow VERY big and lograptor might need similar space for processing purposes. Make sure there is no danger of filling up that partition. A good place is /var/tmp, since that is usually a separate partition dedicated entirely for logs.

`from_address`
: Use a specific sender address when sending reports or notifications. Defaults to address *root@<HOST_FQDN>*.

`smtp_server`
: Use this smtp server when sending notifications. Can be either a hostname of an SMTP server to use, or the location of a sendmail binary. If the value starts with a "/" is considered a path. E.g. valid entries:

```
smtp_server = mail.example.com

smtp_server = /usr/sbin/sendmail -t
```

encodings
A comma-separated list of standard encodings's codecs to use for accessing the log resources. For default its value is 'uft-8, latin1, latin2'.

mapexp
The dimension of translation tables for –anonymize option. The number is the power of 10 that represents the maximum extension of each table (default is 4).

3.4 [patterns] SECTION

This section includes these basic pattern rules:

DNSNAME
Regular expression pattern for DNS names matching.

IPV4_ADDRESS
Regular expression pattern for IPv4 addresses matching.

IPV6_ADDRESS
Regular expression pattern for IPv6 addresses matching.

EMAIL
Regular expression pattern for RFC824 e-mail address matching.

USERNAME
Regular expression pattern for username matching.

ID
Regular expression pattern for numerical ID matching.

ASCII
Regular expression pattern for ASCII characters matching.

These rules are essential for a correct program execution. You don't need to add basic pattern rules to you configuration files because are embedded in program defaults. You can redefine the basic patterns pattern rules but you have to make sure the new patterns are conform with regexp syntax to avoid execution errors. Basic pattern customization is useful to match non-ortodox log elements or if you want to simplify the patterns to slightly speed-up the processing.

Declare additional pattern options if you want to define also additional fields in your configuration. All the pattern options maybe declared using name with uppercase letters, for clarity and for avoiding collisions with field names.

Defined pattern can be used as template strings in the pattern rules of the applications.

3.5 [fields] SECTION

This section contains the fields that can be included in lograptor filters (command option -F) and in application's pattern rules.

Each field declaration maybe a template regex pattern, that uses the declared patterns as template variables. A string interpolation is then used to create the effective regexp patterns during lograptor execution.

The default configuration includes 8 predefined fields:

user
Field for usernames (defaults to `(|${USERNAME})`).

mail
Field for email addresses (defaults to `${EMAIL}`).

from
Field for sender email addresses (defaults to `${EMAIL}`).

rcpt
Field for recipient email addresses (defaults to `$${EMAIL}`).

client
Field for client IP/name (defaults to `(${DNSNAME}|${IPV4_ADDRESS}|${DNSNAME}\[${IPV4_ADDRESS}\])`).

pid
Field for process IDs (defaults to `${ID}`).

uid
Field for user IDs (defaults to `${ID}`).

msgid
Field for message IDs (defaults to `${ASCII}`).

Those filters are usually skipped in the configuration files because are embedded in the lograptor's defaults.

3.6 OUTPUT CHANNEL SECTIONS

The default output channel is *stdout* that is the standard output terminal channel (*TermChannel*). Other types of channels can be defined, currently you can choose either a *Mail Channel* or a *File Channel*.

Channel types have two common options and some characteristic options. Other options are ignored. A channel section has a name of format <*channel-name*>*_channel*. The defined channels are usable within the option –output option.

type
The channel type. Type must be set to "tty" for a terminal channel (*TermChannel*), "mail" for *MailChannel* and "file" for a *FileChannel".

formats
Can be one a comma-separated list of the following: *text*, *html*, or *csv*.

3.6.1 Mail Channel SECTIONS

These are the custom options used by *MailChannel* declaration sections:

mailto
The list of email addresses where to mail the report. Separate multiple entries by a comma. If omitted, "root@localhost" will be used.

include_rawlogs
Whether to include the gzipped raw logs with the message. If set to "yes", it will attach the file with all processed logs with the message. If you use a file publisher in addition to the mail publisher, this may be a tad too paranoid.

rawlogs_limit
If the size of rawlogs.gz is more than this setting (in kilobytes), then raw logs will not be attached. Useful if you have a 50Mb log and check your mail over a slow uplink.

gpg_encrypt
Logs routinely contain sensitive information, so you may want to encrypt the email report to ensure that nobody can read it other than designated administrators. Set to "yes" to enable gpg-encryption of the mail report. You will need to install mygpgme (installed by default on all yum-managed systems).

gpg_keyringdir
If you don't want to use the default keyring (usually /root/.gnupg), you can set up a separate keyring directory for lograptor's use. E.g.:

```
> mkdir -m 0700 /etc/lograptor/gpg
```

gpg_recipients
List of PGP key id's to use when encrypting the report. The keys must be in the pubring specified in gpg_keyringdir. If this option is omitted, lograptor will encrypt to all keys found in the pubring. To add a public key to a keyring, you can use the following command:

```
> gpg [--homedir=/etc/lograptor/gpg] --import pubkey.gpg
```

You can generate the pubkey.gpg file by running "gpg –export KEYID" on your workstation, or you can use "gpg –search" to import the public keys from the keyserver.

gpg_signers
To use the signing option, you will first need to generate a private key:

```
> gpg [--homedir=/etc/lograptor/gpg] --gen-key
```

Create a *sign-only RSA key* and leave the passphrase empty. You can then use "`gpg --export`" to export the key you have generated and import it on the workstation where you read mail. If gpg_signers is not set, the report will not be signed.

3.6.2 File Channel SECTIONS

These are the custom options used by *FileChannel* declaration sections:

method
Method must be set to "file" for this config to work as a file publisher.

path
Where to place the directories with reports. A sensible location would be in `/var/www/html/lograptor`. Note that the reports may contain sensitive information, so make sure you place a .htaccess in that directory and require a password, or limit by host.

dirmask, filemask
These are the masks to be used for the created directories and files. For format values look at strftime documentation here: https://docs.python.org/2/library/time.html

save_rawlogs
Whether to save the raw logs in a file in the same directory as the report. The default is off, since you can easily look in the original log sources.

expire_in
A digit specifying the number of days after which the old directories should be removed. Default is 7.

notify
Optionally send notifications to these email addresses when new reports become available. Comment out if no notification is desired. This is definitely redundant if you also use the mail publisher.

pubroot
When generating a notification message, use this as publication root to make a link. E.g.:

```
pubroot = http://www.example.com/lograptor
```

will make a link: http://www.example.com/lograptor/dirname/filename.html

3.7 REPORT SECTIONS

A report section has a name of format *<report-name>_report*. The defined reports are usable within the option –report option.

These are the entries that can be declared within a report section:

title
: What should be the title of the report. For mailed reports, this is the subject of the message. For the ones published on the web, this is the title of the page (as in <title></title>) for html reports, or the main header for plain text reports.

html_template
: Which template should be used for the final html reports. The default value is `$cfgdir/report_template.html`.

text_template
: Which template should be used for the final plain text reports. The default value is `$cfgdir/report_template.txt`.

The *subreport options* define the report logical divisions. The subreports are inserted in the report using the interpolation of variable string "${subreport}". You can declare a subreport option using an option name thas has a "_subreport" suffix. The order of subreports's declaration is preserved in report composition. In the default report configuration there are 4 subreports defined:

logins_subreport
: User's "logins" subreport.

email_subreport
: E-mail ("email") subreport.

commands_subreport
: System "commands" subreport.

databases_subreport
: Databases lookups subreport.

You could add your own subreports: this can be a needs when you expand the applications configurations provided. To composite the report the subreports are then referred in application's "report data" sections. See lograptor-apps(5) for more details on app's report rules.

3.8 COMMENTS

Lines starting with "#" or ';' are ignored and may be used to provide comments.

3.9 AUTHORS

Davide Brunato <brunato@sissa.it>

3.10 SEE ALSO

lograptor(8), lograptor-apps(5), lograptor-examples(5),

CHAPTER

FOUR

CONFIGURE LOGRAPTOR'S APPLICATIONS

4.1 CONFIGURATION FILES

${confdir}/*.conf

Lograptor defines its applications by configuration files. An application configuration filename is the name of the application followed by the suffix `.conf`. Each file that is located in the configuration directory that has this suffix has to be an application configuration file for lograptor.

An application's configuration file uses the Python's ConfigParser format which provides a structure similar to Microsoft Windows INI files. A configuration file consists of sections and option entries. A section start with a ''[section]" header. Each section can have different `name=value` (`name: value` is also accepted) option entries, with continuations in the style of RFC 822 (see section 3.1.1, "LONG HEADER FIELDS"). Note that leading and trailing whitespaces are removed from values.

4.2 DESCRIPTION

An application configuration file for *lograptor* must contains two sections:

main Contains the parameters of the application. Includes log app-tags, log files locations, priority and enabling status.

rules This section contains the *pattern rules* for the analysis of application's logs. Those regexp rules are used by the engines of *lograptor*.

Optional additional sections can be defined to define report data composition.

4.3 [main] SECTION

desc A fully comprehensive description of the application.

files Log files of the application. You can specify multiple entries separated by commas. Entries can be GLOB filename patterns, so you can use the wildcard characters *?*, *, + in filenames. String interpolation is done on entries just before processing, so you can use obtain the effective list of files to be included in the run. Typically the string `$logdir` (or `${logdir}`) is used to shorten paths that have the same common root. You can also use other variables related to program options, such as `$hostname`, that is linked to the option –hosts.

Finally you can also use some wildcards related to dates:

%Y specifies the year

%m specifies the month as a number with 2 digits (01..12)

%d specifies the day with 2 digits (01..)

Currently only these formats are supported to specify the dates. Filenames that include variables related to dates are expanded by the program according to the date range provided (options –last or –date).

enabled It can be either "yes" or "no." If "no", the program ignores the app. If the application is invoked explicitly using the option -a/–app then the value of this parameter is ignored. This allows you to schedule reports with a favorite set of applications and still be able to use the program for analyze logs of all the applications defined.

priority It's an unsigned integer that indicates the priority of the application, commonly a value from 0 to 10. A lower value indicates an higher priority in the composition of the final report, ie report data elements produced by the application will appear before those of other applications with an higher value. The priority also conditions the processing order of the log files.

4.4 [rules] SECTION

This section contains pattern rules written as regular expressions, according to the syntax of Python's re module. Those rules are used by the program to analyze application's log lines and to extract information from matched events. Each rule is identified with the option name, so must be unique within application. Don't use names already used by other options of the program for defining a pattern rule, in order to avoid ambiguities.

4.4.1 Symbolic Groups

Lograptor makes use of Python's regex symbolic groups to extract information from logs. A pattern rule must contain at least one symbolic group in order to be accepted by the program. For example if a rule is:

```
SMTPD_Warning = ": warning: (?P<reason>.+)"
```

the program extract information about group *"reason"* and is able to use those information during reporting stage. You can use more symbolic groups within a rule for detailing the structure of extracted data:

```
Mail_Resent = ": (?P<thread>[A-Z,0-9]{9,14}): resent-message-id=<(?P<reason>.+)>"
```

The "thread" symbolic group is used to extract thread information from log lines, in order to perform thread matching (see option -T/–thread).

4.4.2 Pattern Rules and Filters

An app pattern rule can also contain variables ($VARNAME or ${VARNAME}) related to a lograptor's filter. At the run each variable is substituted with the corresponding filter's pattern. This feature has sense when you pair a variable with a symbolic group, as in this example:

```
Mail_Client = ": (?P<thread>[A-Z,0-9]{9,14}): client=(?P<client>${client})"
```

If you use filter options the program discards the rules logically excluded by filters (*unused rules*).

4.4.3 Dictionary of Results

Each rule produces a table of results as a Python dictionary. This dictionary has tuples as keys and integers as values. The values record the number of events associated with each tuple. For example with the following rule:

```
Mail_Received = ": (?P<thread>[A-Z,0-9]{9,14}): from=<(?P<from>${from})>, size=(?P
↪<size>\d+)"
```

the tuple key consists of three elements, positionally related to fields <*hostname*>, <*from*> and <*size*>:

```
('smtp.example.com', 'postmaster@example.com', '4827')
```

Of course inserting more symbolic groups increase the complexity of the results and the number of elements of the dictionary. So if you don't need details you could simplify the default pattern rules.

4.4.4 Order of Pattern Rules

The sequence of the rules in the configuration also determines the order of execution during the process of log analysis. The order are important to reduce execution total time. Generally is better to put first the rules corresponding to more numerous log lines.

4.4.5 Writing Pattern Rules

A simple method to write new pattern rules is to use the lograptor unparsed engine for each application, in order to verify which lines are not matched by any pattern rule, e.g.:

```
# lograptor -a dovecot --unparsed -m 1 /var/log/dovecot.log
...
...
```

If the search is not empty start to write a new detailed rule until the match is done and the line disappear from the above search command. Repeat these steps until lograptor doesn't found any unparsed string in your file.

With this technique you can easily write down all the report rules for an application in some minutes.

4.5 REPORT DATA SECTIONS

Additional configuration sections define the data elements for composing the report. These sections have some mandatory options and one or more options that define the usage of application's pattern rules.

4.5.1 Mandatory Options

subreport Indicates in which subreport insert the element. It has to match the name of one of the subreports specified in the main configuration file.

title Header to be included in the report.

color Color to be used for the header (use the names or the codes defined for HTML and CSS specifications).

function Function to apply on the results extracted from the pattern rules of the application. There are three different functions definable, each one lead to a different representation of the results:

`total(), total` Creates lists with total values from the results.

`top(<num>, <header>)` Creates a ranking of maximum values.

The <num> parameter is a positive integer that indicating how many maximum values to be taken into account. The third parameter is a description for the field, which will appear on the right column of a two-column table.

`table(<header 1>, .. <header K>)` Create a table from a result set.

The arguments are the descriptions that have to be included in the headers of the table. The number of arguments determines the number of columns of the table. These tables, also when generated from logs of different applications, are compacted into a single table under specific conditions. For this topic read the REPORT OPTIMIZATION paragraph.

4.5.2 Pattern Rules Related Options

A report data section must includes at least an option that refers to a pattern rule of the application. For doing this simply add the name of a pattern rule as option of the report data section. If you need to refer twice to a pattern rule in the same section you can use a numeric suffix for differentiate the options names. The order of those additional options is important because it is maintained when composing the report.

The syntax of a report rule depends by the function type specified in the "function" option.

Report data sections with function "total"

In case of defining a report data section that uses the *total* function the syntax of an additional option must be:

```
<pattern_rule_name> = (<filter>, "<description>"[:[+]<counter_field>[<unit>])
```

The parameter <*filter*> can have the following values:

* Computes the total on all results.

<field>=<pattern> Consider only the tuples of results for which the specified field satisfies the constraint described by <*pattern*>. The value <*field*> must be the name of a symbolic group and must be defined in all the pattern rules provided for the section.

<field>!=<pattern> Consider only the results that don't satisfy the constraint specified by <*pattern*>. The value <*field*> must be the name of a symbolic group present in all the pattern rules provided for the section.

The <*description*> will be the header of the column of the results.

The optional <*counter_field*> is used to calculate the total value from result values. For default, the count is done on the value associated with the tuple-key of the dictionary of results, ie the number of events extracted for the particular combination of values. If you specify a <*counter_field*> the count is computed using tuple's values related to the field. Fill <*counter_field*> with the name of the symbolic group that you want to use for calculate the total value. If <*counter_field*> is preceded by a "+" the total sum is calculated using field values times the number of events.

The <*counter_field*> can be followed by a measurement <*unit*> specification of bits or bytes. This specification have to be enclosed between square brackets and can have one of the metric prefixes K, M, G, or T. The value is calculated according to the JEDEC specification, ie 1Kbit = 1024 bits. For example "[Kb]" or "[Kbits]" means kilobits and "[GB]" or "[Gbytes]" means gigabytes. The numerical results in bytes or bits are then normalized to the multiple unit best suited for report presentation.

As a full example, having the pattern rule:

```
Mail_Received = ": (?P<thread>[A-Z,0-9]{9,14}): from=<(?P<from>${from})>, size=(?P
↪<size>\d+)"
```

and defining the corresponding report rule:

```
Mail_Received = (*, "Total Messages Processed")
```

you will produce a report that contains the count of total messages received. Instead, using the following option:

```
Mail_Received = (*, "Total Transferred Size":+size)
```

a count of the total number of bytes received will be made. Adding a memory measurement unit specification:

```
Mail_Received = (*, "Total Transferred Size":+size[B])
```

you can afford a better understanding of the results.

Report data section with function "top"

In case of function *top* the syntax of an additional option must be:

```
<pattern_rule_name> = (<filter>, <field>[:[+]<counter_field>[<unit>])
```

All the parameters except <*field*> have the same syntax and meaning as have for the function *total*. The <*field*> parameter can be *hostname* or the name of a symbolic group belonging to the pattern rule associated, with the exception of the *thread* symbolic group that is reserved.

For example, having this pattern rule:

```
Mail_Received = ": (?P<thread>[A-Z,0-9]{9,14}): from=<(?P<from>${from})>, size=(?P
↪<size>\d+)"
```

you can define a report data option that creates the list of servers that have sent more mail:

```
Mail_Received = (*, hostname)
```

Instead, with the following report data option:

```
Mail_Received = (*, from)
```

a ranking of email accounts that have sent more messages is created.

As in the case of the *total* function, you can specify a <*counter_field*> for count alternative values. For example with this report rule:

```
Mail_Received = (*, from:size[B])
```

you obtain the ranking of the largest e-mails sent during the period: Instead, inserting the prefix "+":

```
Mail_Received = (*, from:+size[B])
```

the program computes the list of senders that had the most high traffic during the period.

Report rules with function "table"

In case of function *table* the syntax of an additional option must be:

```
<report_rule> = (<filter>, <field>, ... <field>)
```

The <*filter*> parameter has the same syntax and effect as that has in the case of functions "total" and "top".

The <*field*> parameters are literal strings enclosed in double quotes, or *hostname* (without quotes) or in alternative the name of a symbolic group belonging to the associated pattern rule (except *thread* that is a reserved).

The number of <*field*> parameters cannot be less than the number of columns of the table, that is defined by the section's option "function". When the number of parameters of the report rule is greater than the number of columns of the table, the program collapses the remaining values in the last column of the table, forming a comma-separated list.

If <*field*> is a string enclosed between double quotes it will be used as fixed value in the corresponding column, in order to decorate the data and distinguish results from those extracted by other rules or different applications.

The first <*field*> parameter is used for sorting the table, so is usually better if you use for this a reference to a symbolic group instead of a quoted string.

When multiple report data options are configured the results are merged in a single table, so use multiple report data options only if mixing these results is significant.

4.5.3 Report Optimization

The program automatically merge tables produced from logs of different applications when the tables belong to the same subreport. Table merging is done when if there is an exact matching between titles and headers. The correspondence of the headers is performed on names, total number and position. This feature is useful for example if you want to produce a single table with all user logins. The resulting reports are smaller and more readable.

4.6 COMMENTS

Lines starting with "#" or ';' are ignored and may be used to provide comments.

4.7 AUTHORS

Davide Brunato <brunato@sissa.it>

4.8 SEE ALSO

lograptor(8), lograptor.conf(5), lograptor-examples(5),

CHAPTER

FIVE

LOGRAPTOR USAGE EXAMPLES

5.1 DESCRIPTION

This chapter describes simple cases usage and some advanced ones.

5.2 BASIC PATTERN SEARCH

Search a pattern in specific log file:

```
lograptor -e 'hello' /var/log/messages
```

Same search but ignoring characters case:

```
lograptor -i -e 'hello' /var/log/messages
```

Search a string in Postfix's log files of the last 3 days:

```
lograptor --last=3d -a postfix -e 'example.com'
```

5.3 SEARCHING WITH FILTERS

Search of e-mails sent by an address, with match at connection thread level:

```
lograptor -T -F from=user@example.com -e '' /var/log/maillog
```

Search of e-mail messages sent by a domain:

```
lograptor -F from=.*@example.com /var/log/maillog
```

Search of e-mail messages sent by a domain to another domain:

```
lograptor -T -F from=.*@example.com -e 'to=<.*@example2.org>' /var/log/maillog
```

Search of e-mail messages sent by our domain to external domains:

```
lograptor -T -F from=.*@example.com -e 'to=<.*@(?!example.org>)' /var/log/maillog
```

5.4 GENERATING REPORTS

Produce a default report on console for application *crond*:

```
lograptor --report -a crond -e '' /var/log/cron
```

On a custom report saved on a file:

```
lograptor --report my_report --output file -a crond -e '' /var/log/cron
```

5.5 SCRIPTING AND CRON

lograptor can be easily called by a script and put in a cron execution. For example you can run a daily batch to all logs at midnight:

```
# crontab -l
0 0 * * * lograptor --output=mail,file
```

Running as a batch makes sense if you send the output to not-stdout channels.

5.6 DEFINING APP RULES

When you need to define a new application or to update the configuration of an already defined application the main problem is generally the definition of app's rules. An app rule is essentially a regular expression template, that is transformed into one or several regular expressions at runtime.

To define rules for an application use those steps:

1. Use the *unparsed* matcher to find the first unparsed line in your log:

   ```
   # lograptor -U -s -a dovecot -m 1 -e '' /var/log/dovecot.log
   Sep 22 00:00:04 ockham dovecot: imap-login: Login: user=<brunato>, PID=23892,
   method=PLAIN, rip=192.168.107.132, lip=192.168.1.174, secured
   ```

2. Build a regex pattern and put it in the "rules" section of your application configuration (eg. /etc/lograptor/conf.d/dovecot.conf):

   ```
   IMAP_Logins = dovecot: imap-login: Login: user=<(?P<user>${user})>,\s
                 PID=(?P<thread>(?P<pid>${pid})),\s(\S+),\srip=(?P<client>$
   ↪{client})
   ```

3. Repeat steps 1 and 2 until there are no more unparsed lines.

Into an app's pattern rule you have to define some named groups to retrieve the relevant information and to permit to some program features to works (eg. filters, report, anonymization).

INDEX

Symbols

C

E

F

G

I

M

N

P

R

S

T