Lograptor Documentation

Release 1.0

Davide Brunato

October 07, 2014

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

Pattern matching's searches can be performed together with filtering rules and scope delimitation options. Each run can produce a report that can be easily sent by e-mail or saved in a file system directory. The program can parse logs written in RFC 3164 and RFC 5424 formats. Lograptor require Python >= 2.6, and is provided with a base configuration for a set of well known applications. You can easily extend this set adding new applications with specific pattern search rules.

Part I

Contents:

CHAPTER

ONE

LOGRAPTOR COMMAND

1.1 SYNOPSIS

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

1.2 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 CLI tool, usable for raw or refined searches and to create specific or periodic reports on system logs. The application mix classical pattern matching features with scope parameters and a configurable set of filters. You can define application filtering rule using the classical regexp syntax. Lograptor can also produce and publish reports in various formats. Reporting could be automated using cron.

To understand Lograptor's configuration see lograptor-conf(5).

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

1.3 OPTIONS

`--version`

Show program's version number and exit.

`-h, --help`

Show this help message and exit.

1.3.1 General Options

`--conf``=<CONFIGFILE>`

Provide a different configuration to Lograptor (default is /etc/lograptor/lograptor.conf). If you call the program from the command line without options and arguments, or with only this option, a summary of configuration settings is dumped to stdout and then the process exit successfully.

`-d` `[0..4]`

Logging level. The default is 2 (warning). Level 0 log only critical errors, higher levels show more informations.

1.3.2 Scope Options:

-H `HOST[,HOST...]`, **--hosts**=`HOST[,HOST...]`
Restrict the analysis to log lines related to comma separated list of hostnames and/or IP addresses. File path wildcards can be used for hostnames.

-a `APP[,APP...]`, **--apps**=`APP[,APP...]`
Analyze only log lines related to a comma separate list of applications. An app is valid when a configuration file is defined. As default all apps defined and enabled are processed.

-A
Skip applications processing and works only with pattern(s) matching. This option is incompatible with report and filtering options.

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

--date=`[YYYY]MMDD[,[YYYY]MMDD]`
Restrict search scope to a date or an interval of dates.

--time=`HH:MM,HH:MM`
Restrict search scope to a time range.

1.3.3 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. Empty patterns are skipped.

-i, --ignore-case
Ignore case distinctions in matching.

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

-F `FILTER=PATTERN[,FILTER=PATTERN...]`
Apply a list of filters to the search. The filters specified within a single option are applied with logical conjunction (AND): only the app's rules that contain all the filters, as Python regex's named groups, are considered. Multiple -F options are used with logical disjunction (OR).

-t, --thread
Perform matching at application's thread level. The thread rules are defined in app's configuration file.

-u, --unparsed
Match only lines that are unparsable by app's rules. This option is useful for finding anomalies and for application's rules debugging. This option is incompatible with filters (option -F).

--anonymize
Anonymize output for values connected to provided filters. Translation tables are built in volatile memory for each run. The anonymous tokens have the format FILTER_NN. This option overrides –ip, –uid.

1.3.4 Output Control:

-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.

-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.

`-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. Also see the -s or –no-messages option.

`-s, --no-messages`
Suppress final run summary and error messages about nonexistent or unreadable files.

`-o, --with-filename`
Print the filename for each matching line.

`-O, --no-filename`
Suppress the default headers with filenames on output. This is the default behaviour for output also when searching in a single file.

`--ip`
Do a reverse lookup translation for the IP addresses. Use a DNS local caching to improve the speed of the lookups and reduce the network service's load.

`--uid`
Map numeric 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.

1.3.5 Report Control:

`-r, --report`
Make a formatted text report at the end of processing and display on console.

`--publish``=PUBLISHER[,PUBLISHER...]`
Make a report and publish it using a comma separated list of publishers. You have to define your publishers in the main configuration file to use this option.

1.4 FILES

`/etc/lograptor/lograptor.conf`

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

`/usr/bin/lograptor`

1.5 AUTHORS

Davide Brunato <brunato@sissa.it>

1.6 SEE ALSO

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

CHAPTER

TWO

LOGRAPTOR CONFIGURATION

2.1 CONFIGURATION FILE

/etc/lograptor/lograptor.conf

Lograptor will look for */etc/lograptor/lograptor.conf* as default configuration file, but you can override that by passing `--conf` switch on the command line.

2.2 DESCRIPTION

Lograptor configuration file use 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 whitespace is removed from values.

The configuration file include five fixed-named sections. Extra sections can be added in order to define report's publishers. A publisher can be of two types: *Mail Publisher* or *File Publisher*. The publisher type is defined with the option **method**. The names of the publishers are then used in –publish option. Other sections are ignored.

2.2.1 [main] SECTION

`cfgdir`

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 on a designated loghost is /var/tmp, since that is usually a separate partition dedicated entirely for logs.

`fromaddr`

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

`smtpserv`

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:

```
smtpserv = mail.example.com

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

`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).

2.2.2 [patterns] SECTION

Basic pattern rules. Those rules are essential for correct program execution. All the patterns could be commented out because are also defined in Lograptor's code. It's possible to customize patterns, but you have to make sure the new patterns are conform with regexp syntax to avoid execution errors. Pattern customization is useful to match non-ortodox log elements or if you want to simplify the patterns to slightly speed-up the processing.

`rfc3164_pattern`
This is the path for legacy BSD log header searches, compliant to RFC 3164 specifications.

`rfc5424_pattern`
This is the path for IETF log header searches, compliant to RFC 5424 specifications.

`ipaddr_pattern`
This is the pattern for IP addresses matching.

`dnsname_pattern`
This is the pattern for DNS names matching.

`email_pattern`
This is the pattern for RFC824 e-mail address matching.

`username_pattern`
This is the pattern for username matching.

`id_pattern`
This is the pattern for numerical ID matching.

2.2.3 [filters] SECTION

This section contains default pattern rules for Lograptor filters (command option -F). Each pattern rule is usually referred as a composition of basic patterns. Variable related strings's interpolation is then used to define the effective regexp pattern during execution. You could add your own filter or customize patterns, but in this case you have to make sure that the changes do not exclude valid log lines.

In default configuration 8 filters are defined. Those filters could be commented out because are also defined with it's default in Lograptor code.

`user`
Filter for usernames (defaults to `${username_pattern}`).

`mail`
Filter for email addresses (defaults to `${email_pattern}`).

`from`
Filter for sender email addresses (defaults to `${email_pattern}`).

`rcpt`
Filter for recipient email addresses (defaults to `${email_pattern}`).

`client`
Filter for client IP/name (defaults to `(${dnsname_pattern}|${ipv4_pattern}|${dnsname_pattern}\[${ip`

`pid`
Filter for process IDs (defaults to `${id_pattern}`).

`uid`
Filter for user numerical IDs (defaults to `${id_pattern}`).

`msgid`
Filter for message IDs (defaults to `${ascii_pattern}`).

2.2.4 [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`.

2.2.5 [subreports] SECTION

The *subreports* section define the report logical divisions. The subreports are inserted in the report using the interpolation of variable string "$subreport". The order of subreports's definition is preserved in report composition. In default configuration there are 4 subreports defined:

`logins`
User's logins subreport.

`email`
E-mail subreport.

`commands`
System commands subreport.

`databases`
Databases lookups subreport.

You could add your own subreports: this should be needed when add new apps to configuration. To composite the report the subreports are then referred in application's report rules. See lograptor-apps(5) for more details on app's report rules.

2.2.6 MAIL PUBLISHER SECTIONS

`method`
Method must be set to "mail" for this publisher to be considered a mail publisher.

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

`format`
Can be one of the following: *html*, *plain*, or *csv*. If you use a mail client that doesn't support html mail, then you better use "plain" or "both", though you will miss out on visual cueing that lograptor uses to notify of important events.

`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.

2.2.7 FILE PUBLISHER 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: http://www.python.org/doc/current/lib/module-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

2.3 COMMENTS

Lines starting with “#” or ‘;’ are ignored and may be used to provide comments.

2.4 AUTHORS

Davide Brunato <brunato@sissa.it>

2.5 SEE ALSO

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

CHAPTER

THREE

CONFIGURE LOGRAPTOR'S APPLICATIONS

3.1 CONFIGURATION FILES

/etc/lograptor/conf.d/*.conf

Lograptor uses regexp rules sets, defined in apps configuration files, to extract informations from system logs. Each application is defined with a configuration file placed in the configuration subdirectory `conf.d`, usually in `/etc/lograptor/conf.d`. A configuration file name coincides with the name of the application followed by the suffix `.conf`. Files without `.conf` suffix are ignored.

3.2 DESCRIPTION

An application's configuration file use 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 whitespace is removed from values.

The configuration file must contain two sections:

main This section include general parameters for the application.

rules This section contains rules for application's logs analisys. Those rules are called *report rules* for our scope.

Other sections can be defined to describe rules for report composition. Those sections will be referred hereafter as "report sections".

3.2.1 [main] SECTION

desc Put here a fully comprehensive description of the application.

files Log files for the application. You can specify multiple entries separated by commas. Lograptor use configuration variable string interpolation to obtain the effective list of files to be included in the run. Typically use the string `$logdir` (or `${logdir}`) to shorten the paths that have the same common root. You can also use other variables related to program options, such as `$hostname`, that is linked to option -H/–hosts.

On filenames, you can use wildcard characters typical of command line (eg. ?, *, +), in order to include also the log file rotated periodically.

You can also use variable strings 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..)

At now, only these formats are supported to specify the dates, but others ought to be included in future versions. Filenames that include variables related to dates are expanded by the program according to the date range required (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 is a value from 0 to 10. Lower values indicate higher priority in the composition of the final report, ie report elements produced by the application will appear before those of other applications. The priority also relates to the processing order of log files.

3.2.2 [rules] SECTION

This section contains rules written as regular expressions, according to the syntax of Python’s re module, and so will be called *“pattern rules”*. 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. You should not use names already used by other options of the program for naming a rule, in order to avoid ambiguity or incompatibility.

Symbolic groups

Lograptor makes use of symbolic groups to extract informations from logs. A pattern rule must contain at least one symbolic group 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 informations during reporting stage. Of course you are free to use more symbolic groups within a rule:

```
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).

Variables related to 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 makes 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 discard the rules logically excluded by filters.

Dictionary of results

Each rule produces a table of results as a Python dictionary. This dictionary have 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+)"
```

a tuple key will consists of three elements, positionally related to fields <hostname>, <from> and <size>. In this example a tuple maybe:

```
('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 semplify default pattern rules.

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.

3.2.3 REPORT SECTIONS

These optional sections defines elements for composing the report. For brevity we will refer to these sections as "report sections". These sections have some fixed options and one or more options that describe the usage of application's pattern rules, hereafter referred as "report rules".

Fixed options

subreport Indicates in which subreport insert the element. It must be the name of one of the subreports specified in the main configuration file.

title Header to be included in the report.

color Optional alternative color for the header (names or codes defined in the specifications of HTML and CSS).

function Function to be applied on results extracted from the pattern rules of the application. There are 3 different functions definable, each one for a different representation of results:

`total(), total` A function that allows you to create lists with total values from the results.

`top(<num>, <header>)` A function that allows you to create 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 in the report on the right column of the table.

`table(<header 1>, ..  <header K>)` A function that allows you to create a table from a result set. The parameters are the descriptions that have to be included in the headers of the table. The number of descriptions determines the number of columns of the table. Report tables, also when generated from logs of different applications, can be compacted into a single table under specific conditions. For this topic read REPORT OPTIMIZATION paragraph.

Report rules

A report section must include at least a rule to The remaining options of a report section must all be report rules. These options must be named identical to one of the pattern rules defined in the section [rules] of the configuration. 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 options is important because it is maintained in composition of the report.

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

Report rules with function "total"

In case of function *total* the syntax of the report rules is:

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

Where the parameter <filter> can have the following values:

- `*` Computes the total on all results.

`<field>=<pattern>` Considers only the tuples of results for which the specified field satisfies the constraint described by the pattern. The value <field> must be the name of a symbolic group present in all the report rules specified below for the section.

`<field>!=<pattern>` Consider only the results that do not satisfy the constraint specified by the pattern. The value <field> must be the name of a symbolic group present in all the report rules specified below for the section.

The description is associated to columns of the results.

The optional *<counter_field>* is used to calculate the total value. 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 counting is done using tuple's values related to the field. The <counter_field> must take only numeric values, otherwise it will generate a configuration error.

If <counter_field> is preceded by a "+" the total sum is calculated using field values times the number of events.

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

For 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 rules with function "top"

In case of function *top* the syntax the report rules is:

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

All the parameters except <field> have the same syntax and meaning as in the case of 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 *thread* that is a reserved group.

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 rule to create the list of servers that have sent more mail:

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

Instead, with the following report rule:

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

you create the ranking of email accounts that have sent more messages.

As in the case of "total", you can specify a <counter_field> for counting 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 have high traffic during the period.

Report rules with function "table"

In case of function *table* the syntax of a report rule is:

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

The <filter> parameter has the same syntax and effect as that of the report rules of functions "total" and "top".

The <field> parameters are 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, as defined in 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 other applications.

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

When multiple report rules are provided the results are merged in a single table, so use multiple report rule in the same report section only when these have sense.

3.3 WRITING PATTERN RULES

A simple method to write new pattern rules is make some Lograptor runs limited to extract unparsed strings for a single application, e.g.:

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

Then write down the new rule:

Repeat the steps until lograptor doesn't found any unparsed strings in your file. Take a significantly long log file as input file.

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

3.4 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 result is a smaller and more readable reports.

3.5 COMMENTS

Lines starting with “#” or ‘;’ are ignored and may be used to provide comments.

3.6 AUTHORS

Davide Brunato <brunato@sissa.it>

3.7 SEE ALSO

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

CHAPTER
FOUR

LOGRAPTOR USAGE EXAMPLES

4.1 DESCRIPTION

Lograptor usage examples describe the simplicity.

4.2 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"
```

4.3 SEARCHING WITH FILTERS

Search of mail sent by an address, with match at connection thread level:

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

4.4 GENERATING REPORTS

Produce a report on console for application "crond":

```
lograptor -ra crond /var/log/cron
```

The same but produce an HTML report and publish it with default publishers, including unparsed logs:

```
lograptor -R html -ua crond "" /var/log/cron
```

4.5 SCRIPTING AND CRON

Lograptor can be easily called by a script and put in cron execution.

Part II

Indices and tables

- *genindex*