PyLogAnalyser v0.5.1

A Python multiplatform tool to filter, colorise and analyse logs

1.Introduction

One of the most common activities for developers and testers is reading logs, which includes understanding and analyzing them. Since logs are normally written in black and white, this activity becomes slow, difficult and wearying, reducing the progress and velocity of the sprint time.

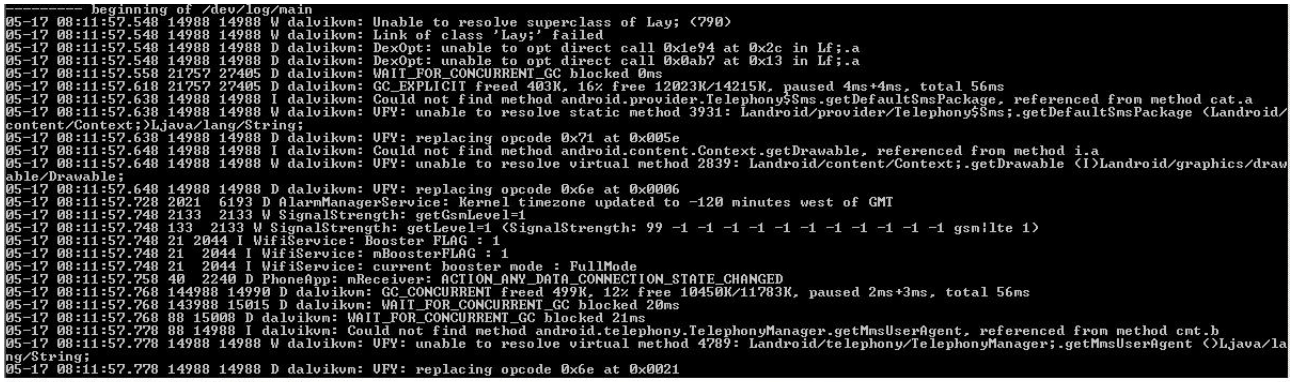
It would be **highly desirable** some tool which receives a log in black and white, parses it, filters it (discarding unnecessary lines) and **colorizes it**, so that it's **much easier** detecting the **useful information**, **improving** the **analysis** and **performance** of the **work**.

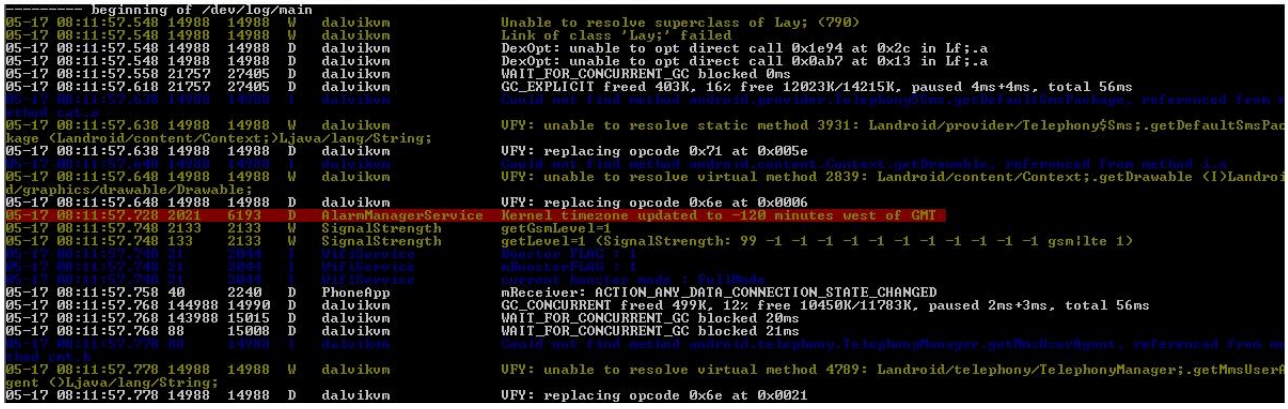
In order to modify the log, it's necessary to provide a configuration file which contains rules which implement the processing to obtain the desired result.

For this purpose, **PyLogAnalyser** has been developed.

I hope you will find this tool very useful, practical and ready to speed up your performance analyzing logs.

As a brief example, see the effects of PyLogAnalyser over an Android log:





Above, android logcat is displayed without any process.

Below, the same log processed by PyLogAnalyser, colorized, columnized and filtered to display just the lines desired by the final user.

The advantage is clear for any developer or tester who needs to analyze logs.

2.Features

- It processes an **input** file descriptor (**file** or **stdin**) according to a **configuration file**, which contains a **set of rules**, and whose **output** is dumped into another file descriptor (**file** or **stdout**).

- The **configuration file** (an INI file using ConfigParser Python package) contains the **rules** which permit operations as **filtering**, **colorizing** (both, **foreground** and **background**), **columnizing** or modifying lines from **regex**, **line number** or **regex groups**.
- **Multi-platform** tool (it works in Windows, Linux and Cygwin) developed in **Python** and usable as a Python package from command-line.
- Easy, Powerful and user-friendly.

3.Installation

Inside the installation file, there is a 'README.txt' file where all the installation process is explained.

4.Basic Usage

For getting started, some basic examples on how to use PyLogAnalyser.

The full syntax is provided in next section.

Three good examples are:

- Colorizing logs
- Filtering unnecessary lines
- Columnize logs to make it more easy to read

Remember that basic examples, both the input files and the configuration files, are stored in the 'android' folder inside the PyLogAnalyser package. In order to see the specific location of this folder, just write the following command from the prompt (after having installed the package):

```
$> python -m loganalyser --demo
```

Also, there are a couple of android logcat logs which are very long stored in the folder extra_logs.

In the message, the full path of the 'android' folder will be printed. Just copy it and use the files contained within.

Colorizing logs

To see the effects of colorizing the logs, take the files 'Android_logcat_brief.conf' and 'Android_logcat_brief.log' from the 'android' folder.

Assuming that current directory in command prompt contains those two files, just run the following command:

```
$> python -m loganalyser -i Android_logcat_brief_short.log -c
   Android_logcat_brief.conf --stdout
```

This way, the log will be displayed in colours.

In case that the log comes from the standard output of some process, it's also possible to colorize by using a pipe and the `--stdin` option:

```
$> python cat.py Android_logcat_brief_short.log | python -m
   loganalyser --stdin -c Android_logcat_brief.conf --stdout
```

The Python script cat.py is contained also in the android folder as a substitute of the same prompt command if it's not available.

Although fully explained in next section, the configuration file renders all the information for colorizing the logs. Inside it, there are seven rules, five for colorizing (red for errors, yellow for warning, blue for info, etc.), one for establishing the regular expression to parse and one for filtering the remaining lines.

Below the result of the log is displayed:

```
W/InputDispatcher( 2021): channel ~ Consumer closed input channel or an error occurred.  events=0x9
E/InputDispatcher( 2021): channel ~ Channel is unrecoverably broken and will be disposed!
W/InputDispatcher( 2021): Attempted to unregister already unregistered input channel
D/KeyguardViewMediator( 2021): setHidden false
D/PhoneApp( 2240): mReceiver: ACTION_ANY_DATA_CONNECTION_STATE_CHANGED
D/PhoneApp( 2240): mReceiver: ACTION_ANY_DATA_CONNECTION_STATE_CHANGED
D/PhoneApp( 2240): mReceiver: ACTION_ANY_DATA_CONNECTION_STATE_CHANGED
D/PhoneApp( 2240): mReceiver: ACTION_ANY_DATA_CONNECTION_STATE_CHANGED
D/PhoneApp( 2240): mReceiver: ACTION_ANY_DATA_CONNECTION_STATE_CHANGED
D/PhoneApp( 2240): mReceiver: ACTION_ANY_DATA_CONNECTION_STATE_CHANGED
D/PhoneApp( 2240): mReceiver: ACTION_ANY_DATA_CONNECTION_STATE_CHANGED
D/PhoneApp( 2240): mReceiver: ACTION_ANY_DATA_CONNECTION_STATE_CHANGED
D/PhoneApp( 2240): mReceiver: ACTION_ANY_DATA_CONNECTION_STATE_CHANGED
D/AlarmManagerService( 2021): Kernel timezone updated to -120 minutes west of GMT
W/SignalStrength( 2133): getGsmLevel=1
W/SignalStrength( 2133): getLevel=1 (SignalStrength: 99 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 gsm|lte 1)
I/WifiService( 2021): Booster FLAG : 1
I/WifiService( 2021): mBoosterFLAG : 1
I/WifiService( 2021): current booster mode : FullMode
D/PhoneApp( 2240): mReceiver: ACTION_ANY_DATA_CONNECTION_STATE_CHANGED
D/AlarmManagerService( 2021): Kernel timezone updated to -120 minutes west of GMT
W/SignalStrength( 2133): getGsmLevel=1
W/SignalStrength( 2133): getLevel=1 (SignalStrength: 99 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 gsm|lte 1)
I/WifiService( 2021): Booster FLAG : 1
I/WifiService( 2021): mBoosterFLAG : 1
I/WifiService( 2021): current booster mode : FullMode
D/PhoneApp( 2240): mReceiver: ACTION_ANY_DATA_CONNECTION_STATE_CHANGED
D/Tethering( 2021): interfaceLinkStateChanged rmnet0, true
D/Tethering( 2021): interfaceStatusChanged rmnet0, true
I/WifiService( 2021): Booster FLAG : 1
I/WifiService( 2021): mBoosterFLAG : 1
I/WifiService( 2021): current booster mode : FullMode
D/PhoneApp( 2240): mReceiver: ACTION_ANY_DATA_CONNECTION_STATE_CHANGED
```

As also, the configuration file is written below:

```
[tag ERROR]
# template takes the key-value pairs defined in 'template regex', see section below.
template = template regex
# level is a regex_id defined in the regex template. E indicates ERROR
level = E
hl_fg = red

[tag WARNING]
template = template regex
level = W
hl_fg = yellow

[tag INFO]
template = template regex
level = I
hl_fg = blue

[tag DEBUG]
template = template regex
level = D
hl_fg = grey

[tag VERBOSE]
level = V
template = template regex
hl_fg = grey

[template regex]
regex =(?P<level>C|W|E|I|D|V)/(?P<modulo>.*?)\s{0,5}\(\s{0,5}(?P<PID>\d{2,4})\): (?P<msg>.*)

[DEFAULT_RULE]
regex = .*
action = filter
```

Filtering unnecessary lines

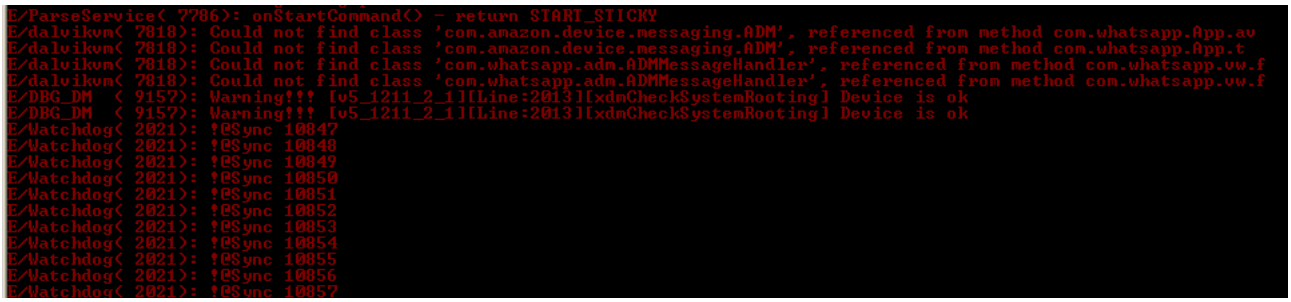
Imagine that you only want to display error messages from previous log, 'Android_logcat_brief.log', a very common situation when an issue has come up or when try to start an investigation.

To do so, just launch from the prompt the following command:

```
$> python -m loganalyser -i Android_logcat_brief_short.log -c
   Android_logcat_brief_just_errors.conf --stdout
```

Again, you will see just the error messages, colorized as previously.

In the configuration file, all the colorizing rules but the error rule are set to filter the lines, this is the reason why only error lines are displayed.



And the configuration file in this case:

```
[tag ERROR]
# For pre-loading the values of this section, See template regex section below
template = template regex
level = E
hl_fg = red

[tag INFO]
template = template regex
level = I
hl_fg = blue
active = no

[template regex]
regex =(?P<level>C|W|E|I|D|V)/(?P<modulo>.*?)\s{0,5}\(\s{0,5}(?P<PID>\d{2,4})\): (?
P<msg>.*)

[DEFAULT_RULE]
regex = .*
action = filter
```

The only rule that is really important in this case is the first one.

About the second one, 'tag INFO', it's set as inactive by the field 'active = no'.

Columnize and colorize

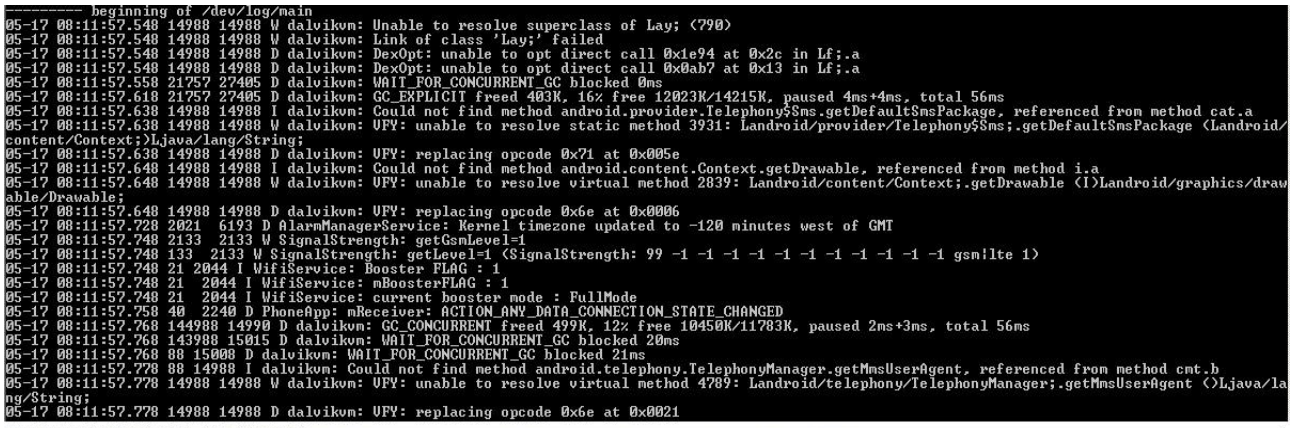
As a final example, it's possible to columnize fields, so that all the fields appear with the same indentation and colours, as done previously.

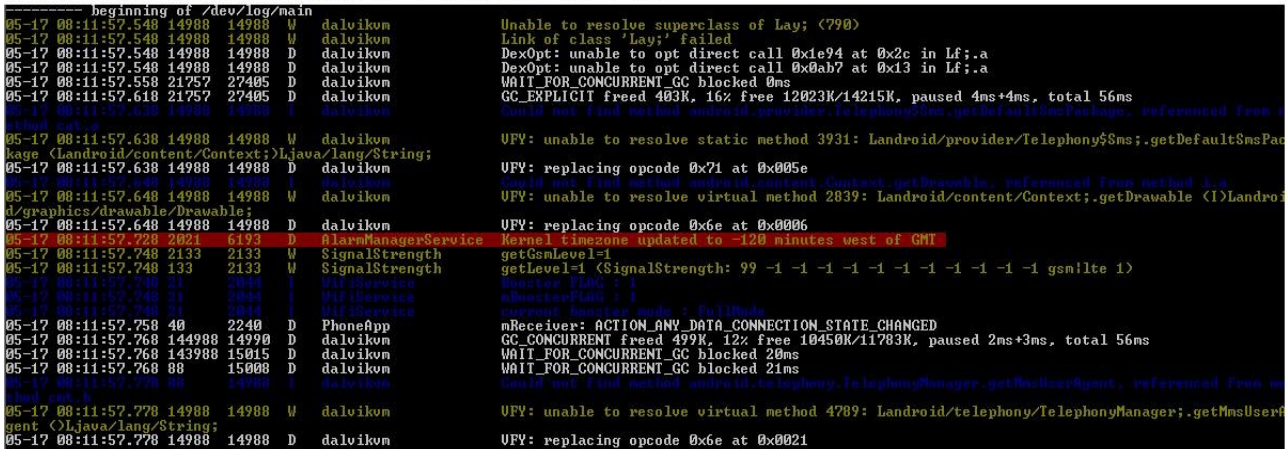
In this case, launch from the prompt the following command:

```
$> python -m loganalyser -i Android_logcat_threadtime_short.log -c
   Android_logcat_threadtime.conf --stdout
```

For the configuration file in this case, just the columnize field has been included in the template rule. This columnize field contains the group id of the regular expression (1, 2, 3...) and the width of the field to be considered.

The figure below is the same as shown at the section Introduction.





And the configuration file can be seen below:

```
[tag ERROR]
# template takes the key-value pairs defined in 'template regex', see section below.
template = template regex
# level is a regex_id defined in the regex template, see below.
level = E
hl_fg = red

[tag WARNING]
template = template regex
level = W
hl_fg = yellow

[tag INFO]
template = template regex
level = I
hl_fg = blue

[tag DEBUG]
template = template regex
level = D
hl_fg = grey

[tag VERBOSE]
level = V
template = template regex
hl_fg = grey
```

```
[template regex]
regex = (?P<date>\d{2}-\d{2} \d{2}:\d{2}:\d{2}\.\d{3})\s{1,5}(?P<PID>\d{2,6})\s{1,5}(?
P<TID>\d{2,6})\s{1,3}(?P<level>C|W|E|I|D|V)\s{1,3}(?P<modulo>.*?):\s{1,}(?P<msg>.*)
columnize = [(1, 8), (2, 6), (3, 6), (4, 3), (5, 20), (6, 50)]

[DEFAULT_RULE]
regex = .*
action = filter
```

For understanding the columnized field, the following table explains it.

columnize = [(1, 8), (2, 6), (3, 6), (4, 3), (5, 20), (6, 50)]

Regex_id	Group no.	Width
date	1	8
PID	2	6
TID	3	6
level	4	3
modulo	5	20
msg	6	50

Hence, by indicating the group number and the character-width in a key-value list, the columnization can be done.

5.Syntax

As explained, the user needs to provide at least and input descriptor, a configuration file and an output descriptor.

Below, the full syntax for using PyLogAnalyser is provided.

```
Usage: python -m loganalyser [-i] [-o] [-c] [-t] [-d|--
demo] [-r|--report] [-b|--about] [-n|--stdin] [-u|--stdout]

Options:
   --version       show program's version number and exit
   -h, --help      show this help message and exit
   -v [log|console|both]  Three possible values:'log'
   --verbose=[log|console|both] saves logs in file;
                                'console' shows in
                                 console;'both' does both;
   -i FILE, --input=FILE   Input file to parse (LOGANA_IN
by default)
   -o FILE, --output=FILE   output file where to write
(LOGANA_OUT by default)
   -c FILE, --conf=FILE     configuration file containing
rules to apply (LOGANA_CONF by default)
   -b, --about      Displays information about the tool
   -s, --demo       Shows instructions on how to use
PyLogAnalyser
   -r, --report     Displays information about how to
report an issue
   -n, --stdin      Uses stdin providing line inputs.
   -u, --stdout     Uses stdout for printing results.
```

Input stream: file or standard input

As input, either a txt file or standard input can be used, but not both. If both input are introduced, the system prevents it, prints an error message and stops execution.

For using input, the `-i` parameter is used followed by the full path to file.

For using the standard input, the `--stdin` option (or `-n` option) is used and some pipe or equivalent system should be employed.

In case that the input file dynamically changes, it's not possible to use the `-i` option, but it's mandatory to use some `tail`-equivalent command, a pipe and standard input.

In the section Colorizing logs (page 2) you can see both examples, using input file and standard input. The same example is shown below,

```
$> python cat.py Android_logcat_threadtime_short.log | python -m
   loganalyser -c Android_logcat_threadtime.conf --stdout
```

The screenshot is also shown above.

In future versions of PyLogAnalyser, an HTML input will be possible, because HTML output is currently possible and it would be interesting to process this same output as input.

Output stream: file and/or standard output

As output, both standard output and a file can be generated. As output file, either txt files, HTML files or both can be used, depending on the user choice.

To select the standard output, the `--stdout` option (or `-u` option) should be written. There are several screenshots above which show the standard output result.

For printing the output, the user should use the `-o` option followed by the full path to file. Depending on the extension of the file, the output will be a txt file, an HTML file or both files:

- If the output file extension is 'txt' (case insensitive), the generated file will be a text file.

  ```
  -o "C:\...\android\Android_logcat_filtered.txt"
  ```

- If the output file extension is HTML (case insensitive), the generated file will be HTML.

  ```
  -o Android_logcat_filtered.html
  ```

- If the output file extension is both (case insensitive), there will be two files, one 'txt' and one HTML.

  ```
  -o ../Android_logcat_filtered.both
  ```

  In this case, the files will keep the file name, the location of the original one and they will use the txt and HTML extension, respectively. For example, previous file would generate Android_logcat_filtered.txt and Android_logcat_filtered.html in the parent directory where the prompt is located in that moment.

As a reminder, take into account the codification and platform-dependency of input txt files for working, since carry-return symbol is different in Windows and Linux.

About the structure of the configuration file, generating rules and such, it is explained below.

Environment variables

Instead of indicating directly the input, output or configuration file, it's possible to use environment variables, so the options can be omitted in the command-line if the variables have been previously created.

For the input file, LOGANA_IN variable is used (%LOGANA_IN% in Windows, $LOGANA_IN in Linux); for the output file, LOGANA_OUT is used.

Finally, for the configuration file, LOGANA_CONF is used.

In case of double-existence, command-line option and environment variable, command-line is considered.

As an example, two standard variables are defined (under Windows environment, in Linux it would be equivalent) below and after the PyLogAnalyser package is instantiated without normal parameters.

```
$> set LOGANA_IN=Android_logcat_brief_short.log
$> set LOGANA_CONF=Android_logcat_brief.conf
$> python -m loganalyser --stdout
```

The result is the same as in the first example in page 2.

Configuration file

This is the file which contains the rules for which all the log processing happens: how to colorize, filter or pass lines and how to columnize the log lines.

Hence, this is the most important file that the user has to create in order to process the input file.

Its structure is just an INI file, where each section is a rule to process and each rule contains key-value pairs which establish the behaviour of the rule.

There are four types of rules:

1. Standard rules
2. ExcludeOnly rule and IncludeOnly rule
3. Template rules

The goal and fields of each rule are different, although template rules and standard rules share most of the key-value pairs. The only key-value pair shared by all the rules is the 'active' key to indicate if the rule is active (value yes) or inactive (value no).

The way how PyLogAnalyser differentiates them is by the rule title,

1. Standard rules don't follow any pattern, just they can't collide with the rest to not create confusion.
2. ExcludeOnly and IncludeOnly rules must contain this title in its rule, though there is no case sensitivity and an space is permitted, so 'Exclude Only', 'excludeonly' or 'exclude only' are permitted (equivalent for IncludeOnly).
3. Template rules must follow the regular expression pattern `^template .*$` with no case sensitivity, which means that the rule should start by the word template (Template, tEmPlaTe, etc. are acceptable also) and some other title after.

Standard rule is a rule intended to receive an input line and process it. So, it is the most common rule to appear in the configuration file.

The structure of a Standard rule is described below:

```
[Standard Rule title]
regex = (?P<date>\d{2}-\d{2}
\d{2}:\d{2}:\d{2}\.\d{3})\s{1,5}(?P<PID>\d{2,6})\s{1,5}
(?P<TID>\d{2,6})\s{1,3}(?P<level>C|W|E|I|D|V)\s{1,3}
(?P<mod>.*?):\s{1,}(?P<msg>.*)

match_search = match|search
regex_flags =  re.DEBUG|re.I|re.L|re.M|re.S|re.U|re.X

(regex_id) =
level = I
mod = dalvikvm
```

```
[DEFAULT_RULE]
regex = .*
match_search = match
regex_flags = 0

action = pass

active = yes
```

msg = VFY: .*
 	
date_min = 05-17 08:11:57.768
date_max = 05-17 08:11:57.778
date_format = %m-%d %H:%M:%S.%f
 	
line_min = 22
line_max = 34
 	
template = template regex
 	
columnize = [(1, 8), (2, 6), (3, 6), (4, 3), (5, 20), (6, 50)]
 	
#Linux: (BLACK, RED, GREEN, YELLOW, BLUE, MAGENTA, CYAN, WHITE)
# bold, bright, dim 	
#Windows: (BLACK, BLUE, GREEN, CYAN, RED, MAGENTA, YELLOW, GREY)
# INTENSITY 	
hl_fg = GREEN
hl_bg = GREY
 	
active = yes|no|true|false|True|False
action = pass|filter	

# Not yet implemented lines_after = lines_before = hl_flags = bold|bright|dim	

The values have been filled as an example from Android_logcat_threadtime.conf and Android_logcat_threadtime_short.log.

Field Title

The title can be anyone which does not collide with the other rules, either they are Template, IncludeOnly, ExcludeOnly or Standard. The title DEFAULT can't be used for sections in Python package ConfigParser as explained in it's official documentation, see link [1] below.

[1] https://docs.python.org/2/library/configparser.html

The default values are indicated in the right column of the table above.

The minimum information contained in a standard rule should be how to match an input line, either by regex or line number (`line_min` or `line_max`) and how to process the rule, either by filtering it or by passing it. The rest of the parameters are optionals and they depend on the desired effect.

Field regex, match_search, regex_flags, date_min/date_max/date_format and regex_ids

This field contains a regular expression string (regex) as processed by Python `re` module.

The regex field should be written as indicated in previous examples, using no backslash symbol, no scape character.

It is highly recommendable to use group identifiers, whose syntax is `(?P<regex_id>)`, which makes easier to recognize components in the regex and it permits the use of regex_ids, as explained after.

By default, the Python re module will use the match function to compare the regex field with the input line; but if the user wants to use the re module search function, it's possible by indicating search in the match_search field.

Moreover, it's possible to use the Python regex flags to indicate special conditions in the application of functions like compile, match or search. Those flags are indicated as displayed above, i.e., exactly as they should be introduced in Python.

About the regex_ids, they consist of key-value pairs where the key corresponds to group identifier used in the regex field, for example, date, level, TID, PID, mod and msg, and the value is set by the user with a regular expression also. For example, in the configuration above there are three regex_ids: level, mod and msg. All of them are processed as regular expressions (see msg as an example which uses special characters).

```
regex = (?P<date>\d{2}-\d{2} \d{2}:\d{2}:\d{2}\.\d{3})\s{1,5}(?P<PID>\d{2,6})\s{1,5}(?P<TID>\d{2,6})\s{1,3}(?P<level>C|W|E|I|D|V)\s{1,3}(?P<mod>.*?):\s{1,}(?P<msg>.*)

match_search = match|search
regex_flags = re.DEBUG|re.I|re.L|re.M|re.S|re.U|re.X

level = I
mod = dalvikvm
msg = VFY: .*

date_min = 05-17 08:11:57.768
date_max = 05-17 08:11:57.778
date_format = %m-%d %H:%M:%S.%f
```

The final goal of regex_ids is to specify the rule to process more deeply than just a regex or a line number. Also, it's possible to use one common regex for all the rules, but different regex_ids for each rule, something very useful in a real situation.

Regarding the regex_id date, it's an special identifier which needs additional fields, not only the key-value pair as the others. In this case, it's possible to select an initial and final date to process the input lines, but it's mandatory to include the date format so that PyLogAnalyser recognizes the digits for year, month, day, etc.

The three parameters date_min, date_max and date_format, are shown above as an example. Both date_min and date_max follow the same format as in the input lines and date_format follows the rules of strptime function in Python datetime module (see link [2] below).

[2] https://docs.python.org/2/library/datetime.html#strftime-strptime-behavior

Those format rules have been written in the configuration files Android_logcat_threadtime.conf and Android_logcat_threadtime_alt.conf as an assistance text in case of need.

Identifier	Example
%a	Sun, Mon, ... (en_US)
%A	Sunday, Monday, ... (en_US)
%w	0, 1, ..., 6 (0 = Sunday, 6 = Saturday)
%d	01, 02, 03, ..., 31
%b	Jan, Feb, Mar, ... (en_US)
%B	January, February, Mars, ... (en_US)
%m	01, 02, 03, ..., 12
%Y	1970, ...
%y	70, 80
%H	Hour 24-h: 00 01 02 03
%h	Hour 12-h: 01, ...
%p	AM, PM (en_US)
%M	00 01 02 ... 59

%S	00 01 02 ... 59
%f	us 000000

Fields line_min/line_max

In case that the selection of input lines should be done by the line number, the keys line_min and line_max are intended for that. Any or both are admissible also.

The key line_min indicates the minimum cardinal number (e.g., 1, 2, 300, etc.) which should be considered for processing the rule.

For the key line_max, it's the maximum cardinal number to consider to process the rule.

```
line_min = 22
line_max = 34
```

Take into account that the first (or last) processed line includes the number written by the user, for example, in the example above, the 22th line for line_min and the 34th line for line_max.

Field template

The template field references the title of a template rule that should be present in the configuration file. This template rule, as explained below, contains key-value pairs which, like a template, fill the values of current rule.

For example, the example above or the three initial examples reference to a template rule name template regex. This template rule sets at least the regex key.

```
[Standard Rule title]
level = I
mod = dalvikvm
msg = VFY: .*

date_min = 05-17 08:11:57.768
date_max = 05-17 08:11:57.778
date_format = %m-%d %H:%M:%S.%f

line_min = 22
line_max = 34

template = template regex
```

```
[Template regex]
regex = (?P<date>\d{2}-\d{2} \d{2}:\d{2}:\d{2}\.\d{3})\
s{1,5}(?P<PID>\d{2,6})\s{1,5}(?P<TID>\d{2,6})\s{1,3}
(?P<level>C|W|E|I|D|V)\s{1,3}(?P<mod>.*?):\s{1,}(?
P<msg>.*)

columnize =  [(1, 8), (2, 6), (3, 6), (4, 3), (5, 20), (6, 50)]
```

This way it's much easier to write some fields in a template just once and make a reference in the rest of the rules.

Format options: 'hl_bg', 'hl_fg', 'columnize'

There are three options which change the format: **hl_bg** highlights the background colour, **hl_fg** highlights the foreground color and **columnize** indicates the width of all the group ids so that the line appears columnized.

The three of them have been presented in the previous section, Basic Usage, so no more explanations are necessary.

```
[Standard Rule title]

#Linux: (BLACK, RED, GREEN, YELLOW, BLUE,
MAGENTA, CYAN, WHITE)
# bold, bright, dim
#Windows: (BLACK, BLUE, GREEN, CYAN, RED,
MAGENTA, YELLOW, GREY)
# INTENSITY
hl_fg = green
```

```
[Template regex]
regex = (?P<date>\d{2}-\d{2} \d{2}:\d{2}:\d{2}\.\d{3})\
s{1,5}(?P<PID>\d{2,6})\s{1,5}(?P<TID>\d{2,6})\s{1,3}
(?P<level>C|W|E|I|D|V)\s{1,3}(?P<mod>.*?):\s{1,}(?
P<msg>.*)

columnize =  [(1, 8), (2, 6), (3, 6), (4, 3), (5, 20), (6, 50)]
```

```
hl_bg = grey
```

By default, if the are not present, the colors of the lines printed will be the same as in the command prompt and the same column width as it contained originally.

Field 'active'

This field indicates if the rule is active or not, i.e., whether it should be considered for processing rules or not.

By default, if this field is not present, the rule is active.

Field 'action':

This field indicates whether the line should be displayed or discarded.

By default, if this field is not present, the rule action will display the line.

One special type of standard rule is the **DEFAULT_RULE**, as shown above, which is used by PyLogAnalyser in case that no other Standard rule written by the user matches the input line.

By definition, PyLogAnalyser needs that every input line match some rule, so in case that the existing rules don't match, the DEFAULT_RULE is used.

```
[DEFAULT_RULE]
regex = .*
match_search = match
regex_flags = 0

action = pass
active = yes
```

If the user wishes to modify the DEFAULT_RULE, to filter the lines by default instead of passing them, it's possible also.

IncludeOnly rule and **ExcludeOnly rule** have the same structure but different behaviour.

Both contain a list of key-value pairs in the form of 'rule1', 'rule2', etc. whose values contains the rule to exclude or include.

Any or both can be used, but in case that both appear, only IncludeOnly will be considered, even though both rules are active.

IncludeOnly is used in case that some only some rules are needed to be included and the rest of them should be considered as inactive, even though their active field is enabled.

ExcludeOnly is used in case that some specific rules should not be considered for parsing the input lines, again, even though their active field is enabled.

This two rules are interesting in case of root-causing a difficult log or very long configuration files, where it's necessary to enable or disable logs quickly.

Both, IncludeOnly and ExcludeOnly can be active or inactive exactly as the Standard rules.

As an example, taken from Android_logcat_brief.conf:

```
[IncludeOnly]
rule1 = tag VFY
active = no

[ExcludeOnly]
rule1 = pass
#active = no

[tag VFY]
template = template regex
```

```
msg = VFY: .*
hl_fg = magenta
hl_bg = grey
```

Template rules are rules not intended to processed, but to be referenced from Standard rules to automatically fill their fields. This is very common in cases where all the rules share the same regex or the same date format, for example.

Template rules are filled with the same fields as Standard rules.

After, Standard rules reference to the template rule from the 'template' field, whose value is the title of the template rule, which always begin with the word 'template', e.g. 'template regex', as shown in file 'Android_logcat_threadtime_two.conf'.

In case that the Standard rule references a template whose field is also present inside, the Standard rule field overwrites the template one.

```
[Template regex]
regex = (?P<date>\d{2}-\d{2} \d{2}:\d{2}:\d{2}\.\d{3})\s{1,5}(?
P<PID>\d{2,6})\s{1,5}(?P<TID>\d{2,6})\s{1,3}(?P<level>C|W|E|I|D|V)\s{1,3}
(?P<mod>.*?):\s{1,}(?P<msg>.*)

columnize =  [(1, 8), (2, 6), (3, 6), (4, 3), (5, 20), (6, 50)]
```

However, Template rules can't reference to other template rules, although it's a good suggestion for next versions of the tool.

Finally, for the **configuration file**, take into account that Standard rules are processed in sequential order, where the first is the most important rule and the last is the less important. Consider it when you write the rules in the configuration file.

Verbose (option -v [log|console|both])

In case that some user discovers an issue in PyLogAnalyser or just to save some results on the execution of the package, it's possible to activate internal logs, which can be displayed from console, into a log file or both, depending on what does the user wishes.

If the option chosen is 'log' or 'both', the log files are stored in current directory where the prompt is launched, inside a '.debug' folder.

Show demo (option --demo)

The best way to see how PyLogAnalyser works is using this option.

This option prints a message for the user on how to quickly use the tool.

Report (option --report)

If you find any issue in PyLogAnalyser, use this option to show instructions on how to provide a full report, so that the developers will be able to solve it ASAP.

About (option –about) and version (option --version)

This option just prints information about the author and the tool.

6. Recommendations

Just as general recommendation, consider the following items during the use of PyLogAnalyser:

1. Use some regex online service for testing the regex.

If the regular expression that you need is difficult, maybe you need to test and try in some regex online service like http://www.regexplanet.com/.

2. Try short logs first

During the generation of the configuration file or general tests done by the users, maybe it's better to use short logs first instead of long logs, because it's easier to debug and solve.

7.Contact

If you detect any issue or just want to contact me, you can do it at my Sourceforge address imoren2x@users.sourceforge.com.

Thank you and I hope you will profit this tool as I did.