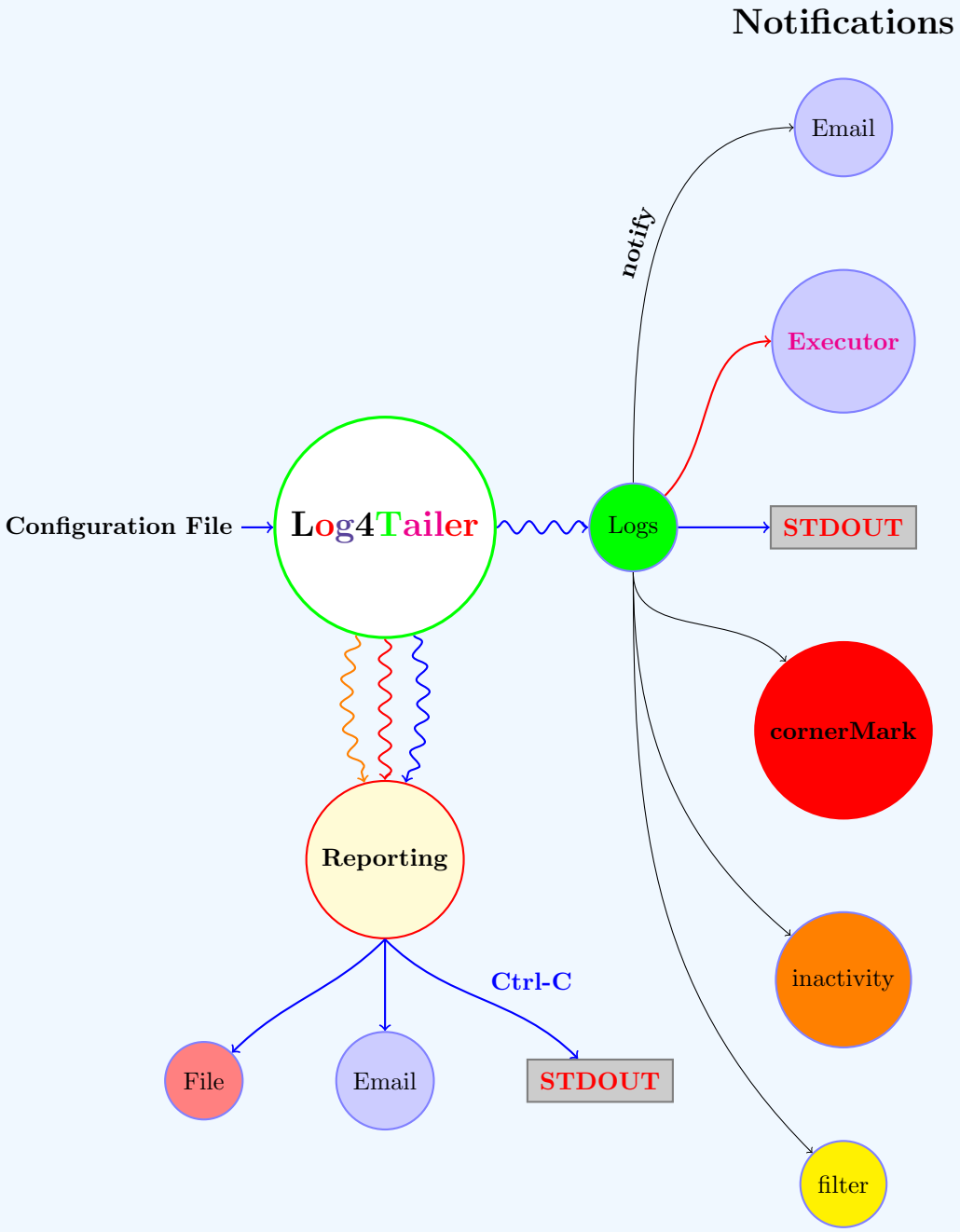
Alerta Project

A log monitoring application



Colors, like features, follow the changes of the emotions.

Pablo Picasso

Alerta User's Guide

Version 3.0.9

http://bitbucket.org/jordilin/alerta

Jordi Carrillo

November 18, 2012

Alerta's Contents

Part I
The Basics

1 Introduction

Alerta aims to provide a different approach to traditional log tailers. While the famous tail linux command line tool does its job just well, **Alerta** adds some more advanced features, like throttling, user defined colors and email notification just to name a few. **Alerta** project's future is to become a very good choice for monitoring application logs. List of features:

- Multitailing capability. It can tail multiple logs at a time
- Colors for every level: warn, info, debug, error and fatal
- Emphasize multiple targets (log traces) given regular expressions
- Follow log upon truncation by default
- User defined colors for each level
- Silent (daemonized) mode for full automatic log monitorization
- Throttling mode. Slow down the information being printed in the terminal
- Inactivity log monitoring
- mail notification
- Every log with its own color scheme
- Freeze output upon specific and well defined conditions
- Cornermark notification, visual alert box in your terminal when you are out of your desktop
- Log reporting by email, to a file or standard output
- Tailing remote logs by means of ssh

Most people use tail -F to tail and monitor the logs these days. When debugging enterprise class applications you cannot just follow (in many situations) what is going on unless you go to the log, less it and check if something was wrong, or just Ctrl-C tail program and scroll back. Human eye cannot distinguish or grab a line out of thousands when that information is showed incredibly fast in the screen. **Alerta** is just more than a tailer, it is becoming a full featured monitoring application.

The *tail* command line Unix/Linux tool has frustrated me many times, so I decided to write my own *tailer* adding more functionalities as I needed them. There might be other tools out there, but to use them you must install them system wide or write your own regexes and remember the terminal escape characters to colorize the output. It wouldn't be the first time I see people writing perl one liners with complex regexes in order to tail logs. **Alerta** is an standalone application that you can run from your /home directory if you desire with default common sense default colours. Of course, you will need read access to those logs you want to monitor.

1.1 Why should I use it?

Just to name a few advantages:

1. **Alerta** is opensource, free software.
2. It runs in standalone mode. At this very moment, no installation is necessary. If you are a sysadmin or an engineer monitoring logs in a network operation center, you can run **Alerta** from its folder. Just untar and run the tailer.
3. Why do it in black and white, when you can do it with colors?

1.2 Components

Alerta has two components, namely:

- log4tailer. log4tailer is the main monitoring application. It has multiple features from tailing logs in multicolor, to a real daemon that communicates with servers.
- log4server. It's a centralized web server that can communicate with the log4tailer daemon to report alertable logtraces.

2 Installation and system requirements

2.1 System Requirements

You'll need at least version 2.4 of Python installed in your system. Python is multiplatform and you can have a copy in http://www.python.org.

Alerta has been tested in Linux servers and it should run in any xterm compatible terminal such as Putty, GNOME terminal and others.

2.2 Installation

Alerta can run in standalone mode, so no installation is necessary if you don't want to. Untar, go to the **bin** folder and run ;-). If for convenience, you want to install it, just type (having root access):

```
python setup.py install
```

and it will be installed system wide. For convenience, you can download the rpm, which has been packaged for SUSE Enterprise Server 10.

2.3 Running the automated tests

The tests can be run using nosetests. You'll need the pymox, mocker and paramiko packages to do that. Alternatively, you can run:

```
make all
```

and it will install the required dependencies in the current directory plus a test script inside the bin folder. Then, just type:

```
make runtests
```

3 Command line parameters

The full command line list is as follows:

```
./log4tail [-s silencemode] [-n numlines] [-t targets] [--throttle secs] [-i inactivityTime]
                [-m mailnotification] [--no-mail-silence] [-c configfile] fullpathToLogs
```

Part II
Advanced features

4 Targets

Target option -t provides an easy way to identify, by means of a regular expression, a particular log trace in your logs. By default, it will tail as normal and if your regex is matched, then that line will be emphasized in a red background.
Example:

this is a log trace matched by target option

You can specify multiple comma separated targets. Each target is just an string being itself a simple string or just a regular expression. It must be enclosed within simple or double quotes.

```
./log4tail -t [--target] 'regex1,regex2,regex3,...,regexN' pathToLogs
```

4.1 Every log with its own target scheme

If you want every log with its own targets, then you must provide them in a config file. In order to do that, just edit a text file and specify the next key, value optional parameters:

```
targets fullpathToLog0 = regex0 : color0; regex1 : color1;..; regexN : colorN
targets fullpathToLog1 = regex0 : color0; regex1 : color1;..; regexN : colorN
```

Where color0, ... colorN are comma separated combinations of foreground, background colors. Please take a look at section 7.1.1.

An example of that could be:

```
targets /var/log/messages = iptables$ : blue, on_red; ^2009-08-07 user : yellow
targets /var/log/mail.log = incoming \d+ : on_cyan; \d{1,3} something
```

If an specific regex does not have a corresponding color, then it will be emphasized using the default color setup, which is the log trace on a red background.

5 Throttling

In release v05, the throttling option is provided. Many applications when started usually tend to upload configuration parameters and log them very fast, which makes nearly impossible to identify anything (unless you open the log and check). In that case, or in other ones where an application logs very fast, you can provide the *throttle* option specifying the number of seconds you want to output in the screen the logging information one line at a time. The number of seconds can be a floating point number.

```
./log4tail --throttle 0.5 pathToLogs
```

That would tail the logs showing the information every half a second one line at a time.

6 Hostname appending

In the case that log traces do not specify the hostname of the server they are being written, you can tell **Alerta** to include it at the very beginning of each log trace. In order to enable that you must provide the following in a configuration file:

```
print_hostname = true
```

Example:

stageserver: INFO trace in here

where *stageserver* would be the hostname where **Alerta** is tailing at.

7 Notifications

Alerta uses *Notifications* to identify what kind of action needs to trigger when it matches a certain line with an specific log4j level. The actions provided as of version v05 are:

- Print (see subsection 7.1)
- Filter (see subsection 7.3)
- Ignore (see subsection 7.3)
- CornerMark (see subsection 7.4)
- Mail (see subsection 7.5)
- Inactivity (see subsection 7.6)
- Executor (see subsection 7.7)
- Poster (see subsection 7.8)
- PrintShot (see subsection 7.9)
- SlowDown (see subsection 7.10)

7.1 Print Notification

PrintAction does what the famous *tail* command does, just printing in the screen (terminal) but with colors. When an application logs information very fast, colors provide and easy way to quickly identify a certain pattern or problem. Every color identify an specific level according to the log4j java framework.

7.1.1 Available colors

Alerta provides the following foreground colors:

black (default for debug and trace)	**This is a test logtrace**
red (default for fatal and critical)	**This is a test logtrace**
green (default for info)	**This is a test logtrace**
yellow (default for warning)	**This is a test logtrace**
blue	**This is a test logtrace**
magenta (default for error)	**This is a test logtrace**
cyan	**This is a test logtrace**
white	**This is a test logtrace**

and the additional background colors:

on_black	this is a test logtrace
on_red	this is a test logtrace
on_green	this is a test logtrace
on_yellow	this is a test logtrace
on_blue	this is a test logtrace
on_magenta	this is a test logtrace
on_cyan	this is a test logtrace
on_white	this is a test logtrace

7.1.2 Providing user defined colors

Default colors used in **Alerta** work well in clear background terminal, such as white. If your terminal has black as background you can provide your own colors in a config file combining foreground and background as you like, such as:

- warn = yellow, on_cyan **this is a WARN test logtrace**
- fatal = red **this is a FATAL test logtrace**
- critical = red, on_yellow this is a CRITICAL test logtrace
- error = magenta **this is an ERROR test logtrace**
- info = green, on_black this is an INFO test logtrace
- debug = cyan **this is a DEBUG test logtrace**
- trace = black **this is a TRACE test logtrace**

and pass -c pathtoconfig as a parameter to **Alerta**.

It is not necessary to provide all the colors in the config file. If yellow is fine for warn and red for fatal, you could say something like:

```
error = red
info = magenta
debug = green
```

error, info and debug colors will be overriden by your ones provided in the config file.

7.1.3 Every log with its own color

If you want to just tail different logs and you want each log with its own specific color, then you can specify that in the config file. This feature overrides the level colors for that specific log, printing all traces with the same color. For example, if we write in a config file something like:

```
/opt/log4/out0.log = green
/opt/log4/out1.log = yellow, on_blue
```

In that specific case, if we run:

```
log4tail -c configfile /opt/log4/out0.log /opt/log4/out1.log /opt/log4/out2.log
```

In the output you'll see *all* traces from /opt/log4/out0.log in green, *all* traces from /opt/log4/out1.log in yellow as foreground and blue as background and in the specific case of /opt/log4/out2.log it will use the default color for each level. The example in Figure 1 will clarify that.

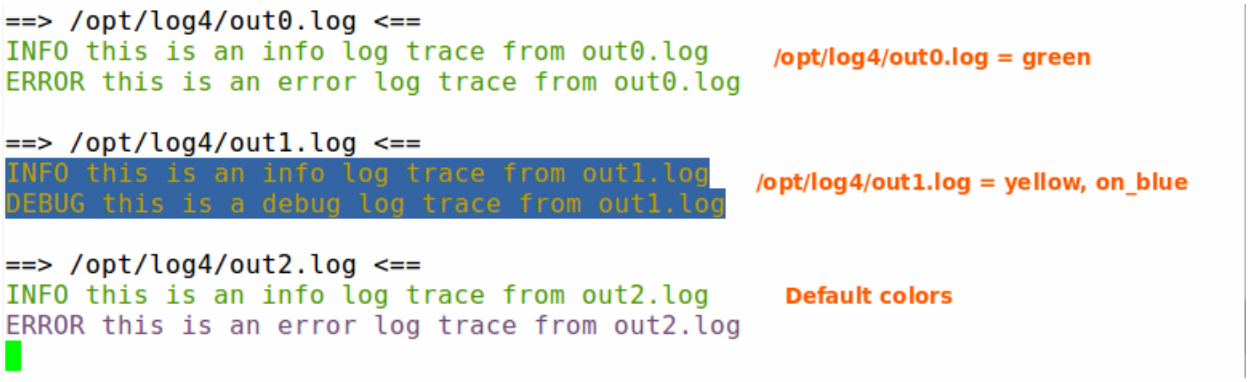


Figure 1: **Alerta** each log with its own color scheme

7.1.4 Line spacing

You can specify if you want line spacing in between log traces. By default, there is no line spacing, but you can specify in your configuration file the next parameter:

```
tracespacing = 2
```

and **Alerta** will leave two white lines in between log traces. That configuration is good for easy identification of specific patterns.

7.2 Filter Notification

It implements the tail and grep. You tail the log and grep for the information you are interested in. In order to enable that feature you need to execute **Alerta** with the -f option:

```
./log4tail -f[--filter] regularexpression pathToLogs
```

7.3 Ignore Notification

The contrary of Filter Notification. It allows you to not notify all those log traces that match an specific regular expression. Basically, it would be like tail and grep -v. In order to enable *ignore* just pass the –ignore option in the command line as follows:

```
./log4tail [--ignore] regularexpression pathToLogs
```

7.4 CornerMark Notification

CornerMark notification will display a colored box in the bottom right side of the terminal in case a Fatal, Error, Warning or Target trace has been found during the specified time. In order to activate this type of notification you need to pass the option cornermark in the command line as follows:

```
./log4tail --cornermark numberofseconds pathToLogs
```

where numberofseconds can be any number.

Example of what you would see if that notification is activated can be seen in Figure 2.

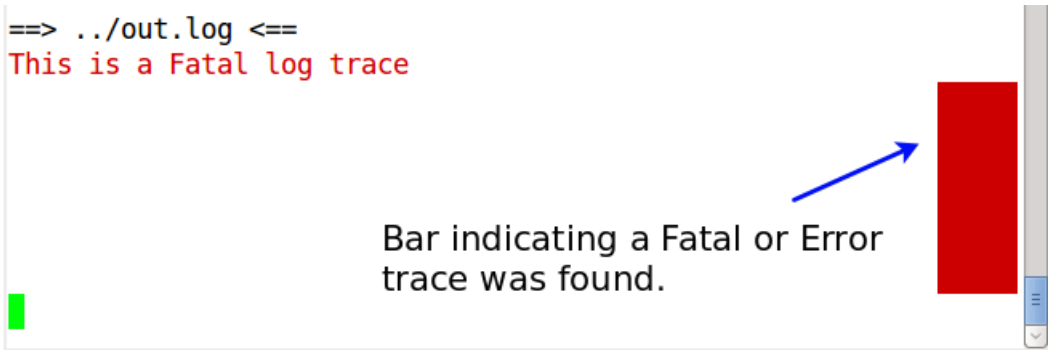


Figure 2: **Alerta** with cornermark activated

For Fatals and Errors the mark will be red. For Warnings will be yellow and for Targets will be cyan. These three colors are fine for easy identification in either clear terminal backgrounds or dark ones.

The motivation for having a corner mark is when you need to go for a break and want some kind of visual alert when something goes wrong. The visual alert will be displayed for the number of seconds you specify in the command line, so it is always advisable to be a number greater than the number of seconds you'll be out of your desktop.

7.5 Mail Notification

Alerta has an SMTP email client built-in if you want to be notified by email. Mail notification is used when you want to be notified by email when a target or level (error or fatal) has been found in the logs. It is very useful when you are tailing for long hours and you cannot take a look at the screen from time to time. It's not necessary to run in silent mode anymore to use this action. This action can be triggered by specifying the command line option -m and specifying the mail details in a configuration file.

```
./log4tail -m [--mail] -c [--config] [[-t [--targets] 'regex1,...,regexN'] pathToLogs
```

You'll need to provide a configuration file with the following key parameters:

```
mail_username = yourusername
mail_hostname = youremailhostname
mail_port = port
mail_ssl = True or False
mail_from = Email from, it can be the same as your to address
mail_to = Email to where you will receive the alerts
```

The password will be asked during runtime to avoid being left in plain ascii in the configuration file. For SSL connection you will need a Python 2.6 runtime, otherwise mail_ssl should be left to False.

If an alert is raised, you will receive an email from mail_from with the subject "Log4tailer alert" for easy filtering.

Alerta works with SMTP Google email accounts[1] if you have one. In Figure 3 you can see how it looks like.

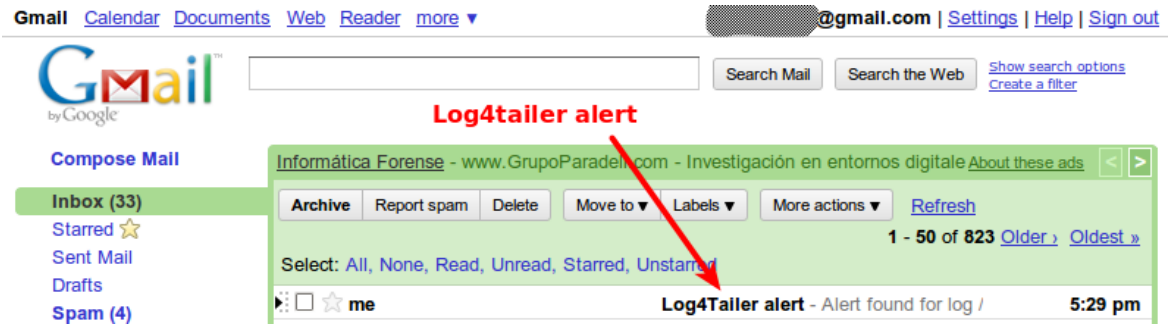


Figure 3: **Alerta** email alert in gmail

I have not tested **Alerta** with **sendmail**. For that you would have to configure sendmail to accept smtp localhost connections. Explaining that is out of the scope of this document. You can take a look in the sendmail documentation book in the Sendmail Consortium webpage at http://www.sendmail.org.

7.5.1 Flood control

In order to avoid being sent lots of notification emails when a flood of undesirable log traces turn up, log4tailer has a way to control that by means of a 60 second gap, which means that non desirable levels that happen within that gap are not notified. After that expiration time, the first undesirable log trace to be found will be notified, triggering again a 60 second gap period.

7.5.2 When should I use Mail notification

MailAction would be like having additional eyes taking a look at the logs. That means, that you can take a rest from time to time basically. If something is found, then you are notified. At this stage, it will notify errors and fatals, considered to be non desired levels in an application. Along with that, you can specify a series of patterns (regexes), log4tailer's targets, that if found could mean that the application is not behaving as expected. In that specific case, you will get notified as well.

[1] Using SMTP SSL connection. For SMTP SSL connectivity you'll need a Python 2.6 runtime, as Python 2.4 does not support it.

7.6 Inactivity Notification

Inactivity monitors for inactivity time in the logs. Inactivity as of this release will just send an emphasized line in the standard output notifying that there has been a lot of inactivity in that log. The inactivity time must be provided in the command line with the -i parameter followed by the number of seconds of inactivity to be monitored in the log. If there has not been any activity for the number of seconds given, **Alerta** will print an emphasized line in the standard output.
As an example:

Inactivity in the log for 5.99955296516 seconds

The command line interface to activate the inactivity monitoring is:

```
./log4tail -i [--inact] numberinseconds pathToLogs
```

7.6.1 Inactivity Mail Notification

If you want a notification by email when inactivityAction is raised, just specify in the config file:

```
inactivitynotification = mail
```

By default is notification to the standard output as shown before.

7.7 Executor

The executor is another type of notification. **Alerta** will execute a program provided by the user if the levels Error, Fatal or Critical have been found in the log trace. The user must provide the command line option *–executable* along with a config file specifying the key *executor* with value the program you want log4tailer to execute and its parameters separated by whitespaces. You can specify a couple of place holders as well, where the first will be the log trace found and the second the log path where the trace was found.

```
./log4tail --executable -c configfile pathToLogs
```

where in configfile you could write something like:

```
executor = anyscript %s %s
```

Where anyscript can be any program accepting two parameters, namely, lograce and log path. The script you provide, of course, will need to have execution permissions for the user owning the **Alerta** process.

As a simple example, if you cannot configure SMTP for log4tailer, then you could setup a script to use the *mail* linux command line to send you an email by means of sendmail.

7.8 Poster notification

The poster notification is basically a REST client built in **Alerta**. That will open the possibility to communicate with a centralized web server with a frontend. Poster notification will register to the server and notify all those fatal, critical, errors or targets found in the log.

```
log4tail --post -c configfile.txt pathToLogs
```

where in the configfile.txt you will need to specify the following parameters:

```
server_url = url to the server
server_port = port
server_service_uri = /where/go/notifications
server_service_register_uri = /register/log4tailer/toserver
server_service_unregister_uri = /unregister/log4tailer/fromserver
```

Upon unregistration, the server will delete the log from the database and all its related logtraces. So, if you don't want that happen, just point the unregister uri to a non existent one, and the server will not do anything, although it will keep the log as registered. Please, take a look at section 15.

7.9 PrintShot

The PrintShot notification is basically the Print notification adding the capability of taking an screenshot whenever we find an alertable log trace. An alertable log trace is any trace within levels Critical, Fatal and Error or those that are targetable. In order to activate this notification, you need to execute **Alerta** in the following manner:

```
log4tail --screenshot -c configfile.txt pathToLogs
```

Where in the configfile.txt you should specify the following key:

```
screenshot = fullpathtopicture.png
```

7.9.1 requirements

The PrintShot notification requires you have the **import** command line program to take screenshots. So, make sure you have that software installed. Most modern Linux distributions have it.

7.9.2 When should I use this feature

This feature is useful whenever you are testing in your local desktop any software and you would like to have a proof that some log trace is getting traced by that software. It would be good for software documentation purposes as well.

7.10 SlowDown

When activated with the command line parameter *slowdown* it will slow down the tailing any time it finds a target, warning, error or fatal log traces. Currently, it will tail one trace per second up to ten consecutives traces.

It can be activated as follows:

```
log4tail --slowdown pathToLogs
```

8 Pause Modes

As of release 1.2 **Alerta** includes pausemodes feature. You will be able to pause or freeze the output by a number of seconds if an specific level or target has been found. In order to enable pausemodes, you must configure them in a config file providing any of the following keys[2] :

```
pausedebug = secondsfordebug
pauseinfo = secondsforinfo
pausewarn = secondsforwarn
pauseerror = secondsforerror
pausefatal = secondsforfatal
pausetarget = secondsfortarget
```

You specify only those ones you want to use. For instance, if we want to freeze the output momentarily (one second) for warnings:

```
pausewarn = 1
```

Then, we should run log4tailer like:

```
./log4tail -c yourconfig /pathToLogs
```

Pausetarget keyword will pause the output for any regex found in the log when running log4tailer with -t option.

[2] the keys are case insensitive, so is the same pauseDEBUG or pausedebug...

9 Reporting

Every time we finish the tailing, log4tailer will output a report, specifying how long log4tailer has been running and the number of events for debug, info and warn. In case of error and fatal, it will provide the timestamps when they were found and their corresponding logtrace. Example:

```
Analytics:
Uptime:
0.0 years 0.0 days 0.0 hours 0.0 mins 45.9482619762 secs
Report for Log out.log
Levels Report:
FATAL:
ERROR:
15 May 2009 17:17:43=>> There was an error here
15 May 2009 17:17:44=>> There was another one in here
15 May 2009 17:17:45=>> Oops, another one
WARN:
4
INFO:
9
DEBUG:
14
TARGET:
3
OTHERS:
18 Jul 2010 11:45:44=>> Inactivity action detected
Ended log4tailer, because colors are fun
```

9.1 Reports by email

If you want a report by email after a given amount of time, then you can do that by means of the config file. There are two values that can be setup, namely:

```
analyticsnotification = mail
analyticsgaptime = 10.5
```

If these two values are uncommented, then you will be required to provide the mail details in the same configuration file, please check the subsection 7.5. The analyticsgaptime should be given in seconds, by default is 3600 seconds (1 hour). You'll receive a report after that period. After that period the statistical information is flushed and then sent again once the gap notification time is expired and so on. In Figure 4 can see an email report notification in a Google gmail account.

9.2 Reports to a file

Alerta can give you a report to a file if you want to. Just provide the analyticsnotification in the configuration file pointing to the reporting file (full path). An example of that would be:

```
analyticsnotification = /opt/reportlog4tailer.txt
analyticsgaptime = 10.5
```

10 Silent Mode

Silent mode, tails the logs in the background (daemonized tailer) and triggers the Mail notification, notifying if error, fatal or any target has been found in the logs.

The syntax to activate the silent mode is:

```
./log4tail -s -c [--config] configfile [-t [--targets] 'regex1,..,regexN'] fullPathToLogs
```

Alerta will require a configuration file with your email details. Please take a look at the subsection 7.5 where specifies the key parameters that need to be specified.

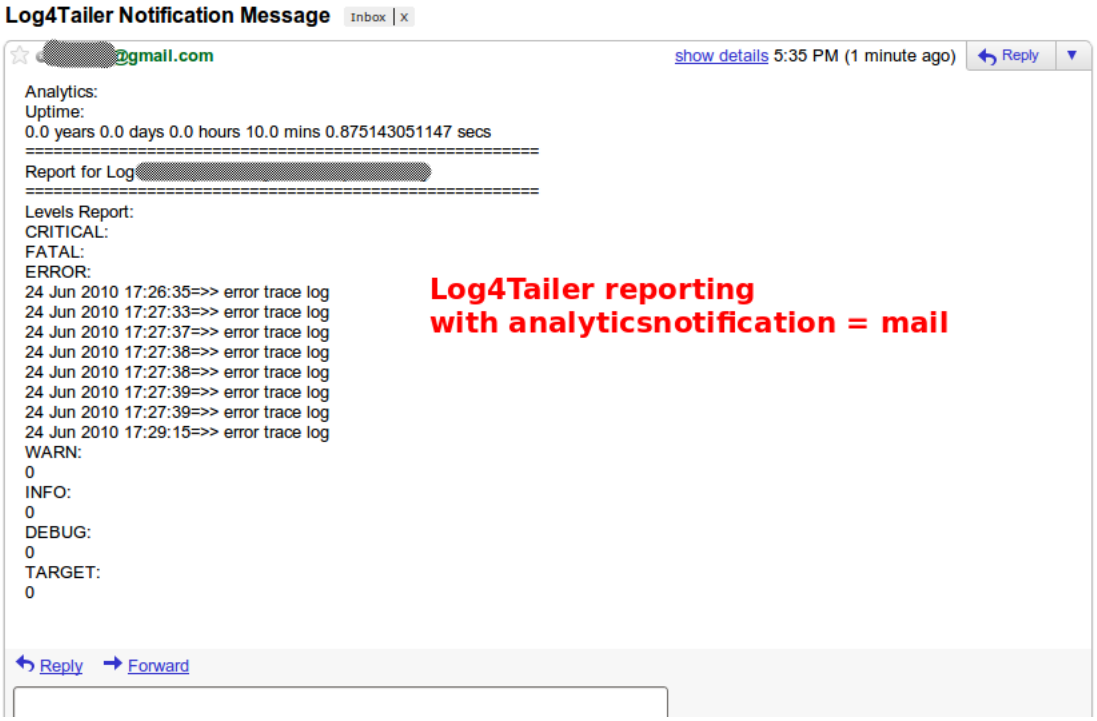


Figure 4: **Alerta** email report notification

Note

It is very important to note that the path to the logs must be the full path, no relative. That's because when **Alerta** enters in daemonized mode, it switches to the root directory closing all buffers and detaching itself from the terminal, in other words, it becomes a real daemon. As a consequence, it is not necessary to execute it with the nohup Linux command line tool.

10.1 Silent Mode when no access to email notification

Alerta provides the no-mail-silence optional command line parameter, where it enters in daemonized mode with no notification setup. It will be up to the user to setup some type of notification by means of a configuration file. Actually, this option is thought to be used along with the -config parameter where you can specify some notification. One of the scenarios would be when you want automatic monitoring when the server you are running **Alerta** has no ports available for email notification. You could provide a config file with *analyticsnotification* pointing to a file where **Alerta** would do a report of the logs status every *analyticsgaptime* seconds or one hour by default.

```
./log4tail --no-mail-silence -c [--config] configfile fullPathToLogs
```

11 Coloring Standard Input

Log4tailer can colorize its standard input to the standard output. Main use would be when your application does some output and finishes. In order to do that just type:

```
yourapplication | log4tail -
cat somelog.log | log4tail -
```

You can use the *more* Linux/Unix application in order to page the output. Example:

```
cat somelog.log | log4tail - | more
```

12 Tailing last N lines

You can tail last *N* lines from the log with the *-n* option. Just type:

```
./log4tail -n numberOfLines pathToLog
```

and it will output the last *numberOfLines* from the log colorizing the corresponding levels.

13 SSH Tailing

SSH Tailing or remote tailing will allow you to tail multiple remote logs from different hosts. As of now, only PrintAction is available, so you'll be able to tail multiple remote logs in a colorful way as specified in section 7.1. In order to tail remotely you'll need to pass as a parameter the -r option along with some config file parameters:

```
./log4tail -r -t targets -c yourconfig.txt
```

In your configfile you must provide the following parameters:

```
sshhostnames = hostname0, hostname1, hostnameN-1
hostname0 = username0, /var/log/log0, /var/log/log1, /var/log/logN-1
hostname1 = username1, /var/log/log0, /var/log/log1, /var/log/logN-1
hostnameN-1 = usernameN-1, /var/log/log0, /var/log/log1, /var/log/logN-1
```

Where:

- **sshhostnames** is a comma separated values of hostnames
- every **hostname** must be a parameter itself where first value should be its username and then the logs you want to tail.

By default log4tailer will try to authenticate by using your rsa key under your ~/.ssh/id_rsa key if it exists, otherwise it will use normal username, password authentication.

Note

Please be aware that you need to be the same username in both client and server. So, if you are logging in as root the ssh key needs to be for the root user. Otherwise, it would request your password.

If you want to use another rsa key other than the default id_rsa key then you can provide one in the config file by using the rsa_key parameter. Example:

```
rsa_key = /home/youruser/.ssh/myrsakey
```

rsa_key value must be the full path to the rsa_key.

Some considerations are to be taking into account. As of now, remote tailing **only** provides PrintAction along with targets, that means that you will be able to tail with colors and emphasize those log traces that match the comma separated regexes provided with -t. Besides, you'll be able to use pauseModes set up in the config file as explained in section 8.

To finish, just Ctrl-c and it will close all channels opened to communicate to the remote hosts.

13.1 Dependencies for SSH tailing

It is very important to note that for remote tailing, you'll need to install the **paramiko module**, available in major Linux distributions. In most of them is available under the name of python-paramiko. In Debian systems, you'll need to type:

```
sudo apt-get install python-paramiko
```

14 Configuration file

Config file is provided fully documented for convenience; just uncomment those lines you are interested to enable and modify them for your specific purposes. In order to enable those values in the config file, you must notify that to **Alerta** as a parameter in startup time.

```
./log4tail -c yourconfig.txt logs
```

If you always use the same configuration file, you can copy it in your HOME directory as *.log4tailer* and log4tail will read it next time you execute the program, even if you don't provide the -c option.

The config file provided for convenience is called log4tailerconfig.txt and is quoted below:

```
# Optional config for log4tailer
# to activate it
# log4tail -c config yourlogs

# ======================================================
# Custom colours for every level. Available
# colours are: red, green, yellow, blue, magenta, cyan
# and white. Uncomment to override the default ones.
# ======================================================

# warn = yellow, on_cyan
# error = magenta
# fatal = red, on_green
# info = green
# debug = black

# ======================================================
# White lines in between log traces
# ======================================================

# tracespacing = 1

# ================================
# targets: which lines do you want
# to emphasize by using regexes
# uncomment and provide your values.
# ================================

# targets fullpathToLog0 = regex0 : color0; regex1 : color1;..; regexN : colorN
# targets fullpathToLog1 = regex0 : color0; regex1 : color1;..; regexN : colorN

# ======================================================
# Every log with its own color scheme, overriding colors
# for every level.
# ======================================================

# /path/to/log0 = yellow
# /path/to/log1 = red

# ======================================================
# Pause the output by the number of seconds specified if
# a level or target has been found. Uncomment the ones
# you want. The value can be any number in seconds.
# ======================================================

# pausedebug = 4
```

```
# pauseinfo = 2
# pausewarn = 1
# pauseerror = 1
# pausefatal = 1
# pausetarget = 1

# ======================================================
# Mail details
# ======================================================

# mail_username = yourhostusername
# mail_hostname = mailhostname
# mail_port = 25
# mail_ssl = True or False
# mail_from = any from address
# mail_to = alerts will be sent in the address you specify in here

# ====================================================================
# executor notification
# ====================================================================

# executor = program command1 command2 %s %s
# executor = program command1 command2

# ====================================================================
# poster notification
# ====================================================================

# server_url = url to the server
# server_port = port
# server_service_uri = /where/go/notifications
# server_service_register_uri = /register/log4tailer/toserver
# server_service_unregister_uri = /unregister/log4tailer/fromserver

# ====================================================================
# Inactivity notification, by email or stdout.
# Possible values can be "mail" or "print". By default is "print".
# ====================================================================

# inactivitynotification = mail

# ====================================================================
# PrintShot notification
# ====================================================================

# screenshot = fullpathtopicture.png

# ====================================================================
# Analytics notification. You can make log4tailer send you
# a report every analyticsgaptime seconds. By default it will be
# printed out once finished. Uncomment analyticsnotification to
# report by email or to a file. Another possible value can be "print".
# ====================================================================

# analyticsnotification = mail
# analyticsnotification = fullPathToaFile
# analyticsgaptime = 10.5
```

```
# ==============================
# SSH Tailing parameters
# ==============================

# sshhostnames = hostname0, hostname1, hostnameN-1
# hostname0 = username0, /var/log/log0, /var/log/log1, /var/log/logN-1
# hostname1 = username1, /var/log/log0, /var/log/log1, /var/log/logN-1
# hostnameN-1 = usernameN-1, /var/log/log0, /var/log/log1, /var/log/logN-1

# rsa_key defaults to ~/.ssh/id_rsa, if that's not your case then
# provide yours

# rsa_key = fullpathToRsaKeyName
```

Part III
Log4Server

15 A centralized reporting web application

The **Alerta** project includes a web backend application that receives notifications from the log4tailer clients, notifying in a web front page about the status of the logs in several machines. The log4server is implemented using the Django web framework and can run in any wsgi compliant web server, such as Apache, Cherokee, Nginx or CherryPy, just to name a few. Basically, the clients will register first to the server and then notify if any fatal, error, critical or target logtrace has been found.

A network diagram is showed in Figure 5.

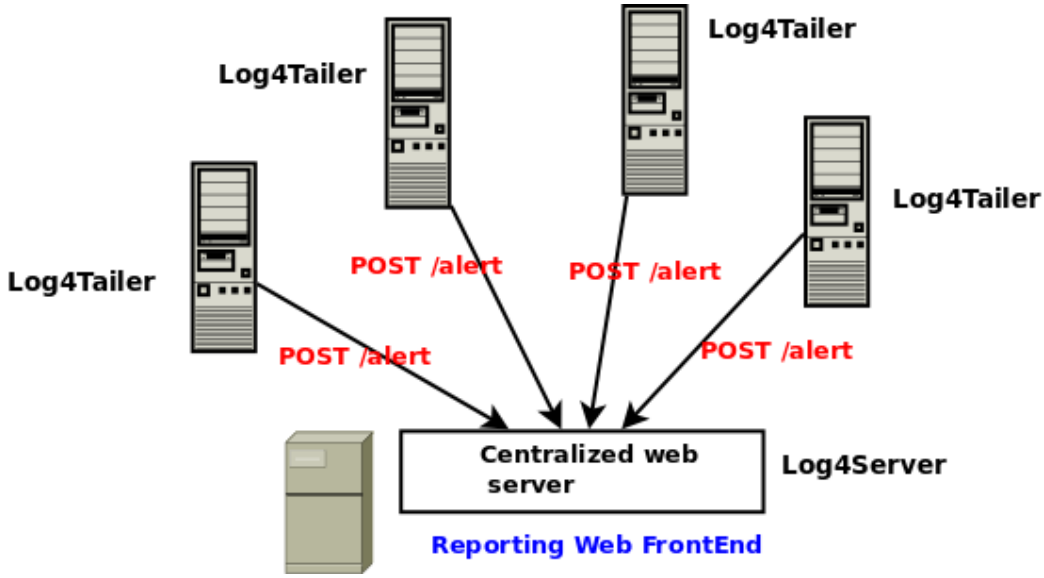


Figure 5: Log4Server network diagram

In the next section, we will describe the API that the log4server application implements and the one being used by the log4tailer poster notification.

16 Log4Tailer compliant server

Alerta client has the poster notification that allows the tailer to communicate with a centralized web application, notifying it of possible problems from remote logs. The poster notification will first register the tailer client to the server, and next calls will be for alert notifications only.

16.1 Log4Tailer server API REST interface

The compliant API interface is as follows:

16.1.1 Registration

Log4tailer client registers to the server on startup when poster notification is provided.

HTTP Method	*URL*	*Description*
POST	/register/	Log4tailer client registration to the server

The POST method will have in the body a JSON object with the following information:

- logpath Full path of the log
- hostname log's server hostname

Upon a successul POST the server will reply a 201 CREATED answer.
Example:

```
POST /register/
{“logpath” : “/var/log/messages”, “logserver” : “localhost”}
RESPONSE
{“id” : 3}
HTTP/1.1 201 CREATED.
```

The id returned is the log identifier id.

16.1.2 Unregistration

Log4tailer client unregisters from the server ones it is stopped.

HTTP Method	*URL*	*Description*
POST	/unregister/	Log4tailer client unregistration to the server

The POST method will have in the body a JSON object with the following information:

- id log registration id

Upon a successul POST the server will reply a 200 OK answer
Example:

```
POST /unregister/
{“id” : 4 }
RESPONSE
HTTP/1.1 200 OK.
```

16.1.3 Alerting

HTTP Method	*URL*	*Description*
POST	/alerts/	New alert has been found.

The POST method will have in the body a JSON object with the following information:

- logtrace logtrace that triggered the alert.
- level level of the aforementioned logtrace.
- log log where the logtrace belongs to.
  - logpath Full path of the log
  - hostname log’s server hostname

Upon a successul POST the server will reply a 201 CREATED answer.
Example:

```
POST /alerts/
{“logtrace” : “This is an error trace”, “loglevel” : “error”, “log” : {“id” : “logid”, “logpath” :
“/var/log/messages”, “logserver”: “192.168.1.1”}}
RESPONSE
HTTP/1.1 201 CREATED.
```

16.1.4 Status

HTTP Method	*URL*	*Description*
GET	/alerts/status	Status of the log files.

It returns all log traces triggered along with the log and server they belong to. The date when it happened is reported as well. The results are paginated in reverse order, the newest first.
Example:

```
GET /alerts/status
RESPONSE
HTTP/1.1 200 OK.
```

If you go to /alerts/status in your web browser, you would see something as showed in Figure 6.

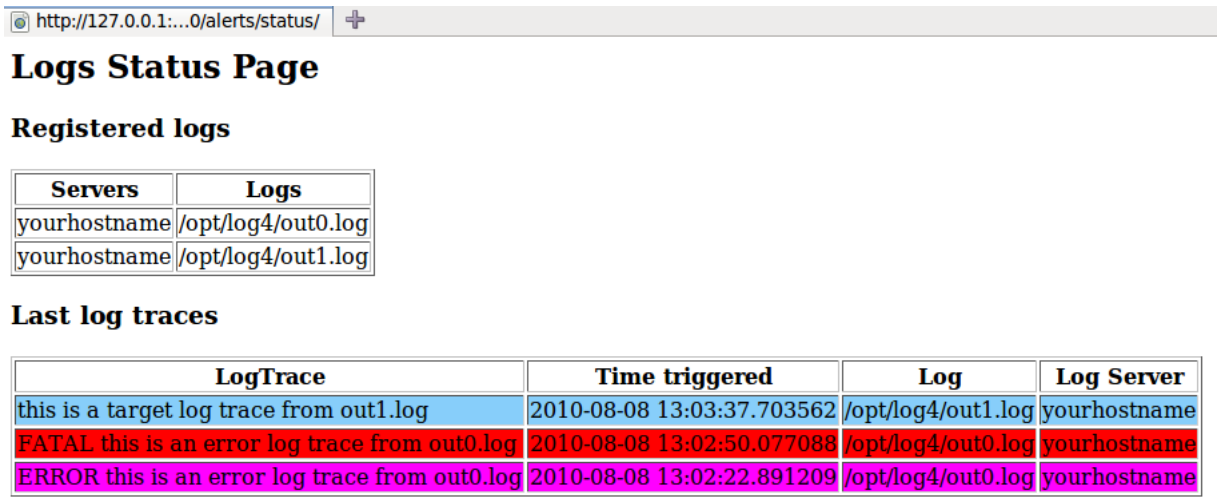


Figure 6: Log4Server status web reporting

17 Log4Server deployment

Log4Server is implemented using the Django web framework. It persists alertable log traces into a database for easy reporting and log4tailer client registrations.

The application uses buildout to manage all application dependencies and deployment. By issuing the bin/buildout command, it will build a log4server.wsgi file, that is the one that you will need to use when setting up the web server. In this document, we will show the instructions on how to set it up by using the Apache web server.

17.1 Building the wsgi file

First of all, download the log4server distribution sources from googlecode site. Untar the package and execute[3]:

```
virtualenv --no-site-packages ENV
. ENV/bin/activate
python bootstrap.py
bin/buildout
```

If everything is fine[4] you will see a directory called *bin*. If you go into that directory you will see a file called log4server.wsgi.

17.2 Apache configuration

In a Debian system such as Ubuntu, you'll need to install Apache and libapache2-mod-wsgi. Once installed, proceed as follows:

[3]The project provides a Makefile that executes the bin/buildout command for convenience. It can take a while, so relax and take a cup of tea.

[4]You need external internet connectivity, as it will download all dependencies such as Django web framework in the current directory.

```
cd /etc/apache2/sites-available
```

and place a file called log4tailer.conf with the following contents[5]:

```
Listen 127.0.0.1:8000

Alias /media/ /path\_to\_log4server-version/src/log4server/media/

<VirtualHost *:8000>
    WSGIDaemonProcess log4server processes=1 threads=5 display-name=%{GROUP}
    WSGIProcessGroup log4server
    WSGIScriptAlias / /path\_to\_thewsgi/log4server/bin/log4server.wsgi
</VirtualHost>

WSGISocketPrefix     run/wsgi
WSGIRestrictStdout   off
```

Then, go to:

```
cd /etc/apache2/sites-enabled
```

and place a soft link to the previous created log4tailer.conf file:

```
ln -s ../sites-available/log4tailer.conf
```

17.3 Deciding on database

Log4Server needs a database in order to persist the alertable logtraces. Sqlite3, Mysql or PostGres should be, by far, enough. Actually, with just an Sqlite3 would be just fine. Before starting the server, you should sync the database the first time or any time that the database does not exist already. In order to do that, execute the following command:

```
bin/log4server syncdb
```

log4server is just a wrapper for the *manage.py* Django python script.

17.4 Restarting Apache

Once you have placed the Apache configuration file pointing to the wsgi file, you'll need to restart Apache.

17.5 Main Web Frontend urls

Once Apache is running, open a web browser and go to the next url to see if it works:

```
http://hostname:port/alerts/status
```

If you see a web page with the title **Logs Status Page** then it means it's working. Congratulations!!.

[5] this Apache2 configuration has been tested on an Ubuntu Maverick 10.10 Server

Part IV
Case Studies

18 Full Automatic Log Monitorization

Full Automatic log monitorization can be performed when you execute **Alerta** in silent mode passing the parameters *-s*. Log4Tailer will run silently in the background notifying by email when something goes wrong. As of now, it will notify errors, fatals and those targets specified as a parameter or in the config file. It is important to notice that every log can have its own set of targets (regexes). Apart from that, you can make log4tailer to monitor inactivity in the log and notify you by email as well. You just need to specify that in the config file as explained in the section 7.6.

Summing up, full automatic monitorization will monitor inactivity, errors, fatals and targets specified in the config file or command line as parameters. This will give you extra confidence on the monitoring of your application if your application uses already nagios or other monitoring software. [6]

19 Semi Automatic Log Monitorization

You can have a mix of email notification and normal colorized print action. You just need to execute log4tailer passing as a parameter *-m* and the corresponding configfile if you want to enable additional features.

20 No SMTP email access for log4tailer

Sometimes a server can have the email ports closed (firewalled) due to security policies. Alternatives:

- For those cases you can use the executor notification using the mail Linux command line to send email provided that the server runs some kind of MTA like sendmail. Let's see an example:

  ```
  log4tail --executable -c configfile.txt /var/log/out.log
  ```

  where in the configfile.txt you could write something like[7]:

  ```
  executor = echo ' %s %s ' | mail -s 'log4tailer alert' -t youremail@hostname.com
  ```

  If sendmail sends email to your localhost, then you could read the email easily by using the famous command line client mutt for example. It's important to note, that you can daemonize **Alerta** in that case as well:

  ```
  log4tail --no-mail-silence --executable -c configfile.txt /var/log/out.log
  ```

  That means that log4tailer will be a daemon monitoring the out.log and sending email by using the mail Linux command line.

- You can always make **Alerta** to report you in a file every some minutes or activate the cornermark notifications (see section 7.4). Both features are really nice to activate them when you need to go for a break. You can setup the cornermark feature with the *cornermark* parameter specifying a time in seconds bit longer than the time you'll be out of your desktop to avoid the mark going away. The marks stay in the terminal for the time you specify.

[6] It is important to notice that pausemodes should not be enabled. That feature is to pause the output when having PrintAction enabled.

[7] If you use the *echo* command line tool providing both place holders (log trace, log path), make sure you leave a white space in between quotes.