```
#!/usr/bin/python
"""

                    LOLLERSKATES

Lots Of Logs Left Easily Rendered So Knowledgeable Admins Time Easily Saved

                    Log analyzer program

Compares logfiles against a series of regular expressions to filter
out uninteresting bits and email whatever is left to the
administrator. The administator can now increase security and
awareness of what the system is doing without having to read the
entirety of the boring log files which is often impossible.

Tracy R Reed
treed@tracyreed.org

Copyright 2014

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA


"""

import re, smtplib, socket, os.path, sys, time

# Load in the config file
import lollerskates_config as config

hostname = socket.gethostname()

# Ugh. How can I get rid of these globals? Globals are bad, right?
regexes      = []
events       = []
remove_lines = []

class regex:

    """ A trivial little class to hold our compiled regex and the
  original source line so I can reference them by attribute name
  instead of as indexes into a tuple."""

    compiled = ""
    line     = ""
    macro    = ""


class SingleInstance:

    """Stolen from
  http://stackoverflow.com/questions/380870/python-single-instance-of-program
  StackOverflow is sweet. We don't ever want more than one copy
  running."""

    def __init__(self):
        import sys, os, errno, tempfile
```



```
        self.lockfile = os.path.normpath(tempfile.gettempdir() + '/' + os.path.b
asename(__file__) + '.lock')
        if sys.platform == 'win32':
            try:
                # file already exists, we try to remove (in case previous execut
ion was interrupted)
                if(os.path.exists(self.lockfile)):
                    os.unlink(self.lockfile)
                self.fd =  os.open(self.lockfile, os.O_CREAT|os.O_EXCL|os.O_RDWR
)
            except OSError, e:
                if e.errno == 13:
                    print "Another instance is already running, quitting."
                    sys.exit(-1)
                print e.errno
                raise
        else: # non Windows
            import fcntl, sys
            self.fp = open(self.lockfile, 'w')
            try:
                fcntl.lockf(self.fp, fcntl.LOCK_EX | fcntl.LOCK_NB)
            except IOError:
                print "Another instance is already running, quitting."
                sys.exit(-1)

    def __del__(self):
        import sys
        if sys.platform == 'win32':
            if hasattr(self, 'fd'):
                os.close(self.fd)
                os.unlink(self.lockfile)

def write_last_offset(file,fd,statefiles):

    """ Write out where we leave off in this file. """

    offset        = fd.tell()
    location_file = open("%s/%s.offset" % (statefiles, os.path.basename(file)),"w")
    location_file.write(str(offset))
    location_file.close()
    if VERBOSE: print "Wrote last offset for file %s: %s" % (file, offset)


def get_last_offset(file,installdir,statefiles):

   """ Read in where we last left off in this file. """

   if (os.path.isfile("%s/%s.offset" % (statefiles, os.path.basename(file)))):
       location_file = open("%s/%s.offset" % (statefiles, os.path.basename(file)),"
r")
       offset        = location_file.readline()
       # Maybe the file shrank or was log rotated
       if os.path.getsize(file) < int(offset):
           return 0
       if VERBOSE: print "Read last offset for file %s: %s" % (file, offset)
       return offset
   else:
       return 0


def process_line(line,regexes,matchdates):

    """ Compare the line to our list of regexes to ignore and append
  it to our events array if it is a keeper."""

    line = line.strip()
    # loop over the regex's checking to see if there is a match
    for currentregex in regexes:
        # If a match, stop looping, note the time in matchdates, move on.
```

```
        if currentregex.compiled.match(line):
            matchdates[currentregex.line] = int(time.time())
            # If in VERBOSE mode tell the user which actual regex matches this l
ine
            if VERBOSE: print "%s MATCHED %s" % (currentregex.line, line)
            break
    else:
        # If there is no match add it to our array of events to report
        if VERBOSE:
            print "NO MATCH: %s %s %s" % (line, currentregex.line, currentregex.
macro)
        events.append(line)

        # Complicated generator based way of doing the above for-else comparison
        #    try:
        #        regex = (regex.match(line) for regex in regexes if regex.match(
line)).next()
        #    except StopIteration:
        #        events.append(line)


def send_mail(events,email,smtp_server):

    """ If we have some events to report send email to the admin """

    if (len(events) != 0):
        message = """From: "Administrator" <%s>
To: "Administrator" <%s>
Subject: %s LOLLERSKATES results

%s
""" % (email, email, hostname, ''.join('\n'.join(events)))
        smtp = smtplib.SMTP(smtp_server)
        smtp.sendmail("root@%s" % hostname, email, message)
        smtp.quit()
        if VERBOSE: print message


def replace_tokens(line,macros):

    """ Take a line as input and replace the tokens defined in the
  macros list with their corresponding regular expressions anywhere
  they appear in the line."""

    if VERBOSE: print "replacing tokens"
    for token, regex in macros:
        line = line.replace(token, regex).strip()
        if VERBOSE: print line, token, regex
    return line


def insert_tokens(events,macros):

    """ We do the opposite of replace_tokens. We look at each line in
  events and compare it with the regexes and replace anything that
  matches with the appropriate token. Then we append this to the end
  of the events. Now we have something we can easily paste into
  ignore.conf to ignore these lines."""

    # Create a dictionary so we can uniq our lines by setting the line
    # as the key equal to 1
    addtoignore = {}

    if (len(events) != 0):
        for line in events:
            for token, regex in macros:
                line = re.sub(regex, token, line)
            # And while we're in here we should escape any parens,
            # braces, stars, etc. Basically anything that might be a
```



```
            # special char to a regex since this is intended to be
            # suitable use in ignore.conf
            line = re.sub("([\\\.\*\(\)\[\]])", r"\\1", line)
            if VERBOSE: print line, token, regex
            addtoignore[line] = 1
        events.append("\nYou can add the following to ignore.conf if you do not want to see these log events
again:\n ")
        addtoignoreitems = addtoignore.items() # Turn the dictionary
                                               # into a list of items
                                               # that were in the dict
        addtoignoreitems.sort()
        for item in addtoignoreitems:
            events.append(item[0])

def load_ignore(installdir, macros, ignorefile):

    """ Read in our file of regexes to ignore while substituting the
  macro tokens with their corresponding regexes. This array is an
  array of regex objects where regex.macro is our macro replaced
  actual full regex, regex.compiled is the compiled regex,
  regex.line is the original line with macros in it that the user
  will be familiar with."""

    try:
        regexfile = open("%s/%s" % (installdir,ignorefile), 'r')
    except IOError, e:
        if e[0] == 2:
            create_ignorefile = open("%s/%s" % (installdir, ignorefile), 'w')
            regexfile         = open("%s/%s" % (installdir, ignorefile), 'r')
        else:
            print e                  # any other error
            sys.exit(1)

    regexes = []
    for line in regexfile.readlines():
        if not line:
            if VERBOSE: print "ignore file contains a blank line! Not matching it."
            break

        if line[-1:] != "\n":
            line += "\n"
        newregex          = regex()
        newregex.macro    = replace_tokens(line,macros)
        newregex.compiled = re.compile(newregex.macro)
        newregex.line     = line
        regexes.append(newregex)

    for currentregex in regexes:
        if currentregex.compiled.match("this is a string which nothing in the ignore file should ever
match"):
            print "You have a regex in the ignore file which will match everything. This is surely"
            print "not what you want since it ensures that you will never be emailed anything from"
            print "the logs. The offending regex is:"
            print currentregex.line
            sys.exit()
    return regexes


def load_regexes(installdir,macros):

    regexes = load_ignore(installdir, macros, "ignore.conf")
    regexes.extend(load_ignore(installdir, macros, "local_ignore.conf"))
    return regexes


def load_matchdates(statefiles,regexes):

    """ Load our tab separated file of dates when each regex last
  matched so that we can warn on any regex that hasn't been matched
```

```
    in a given length of time. This will weed out unnecessary or typo
    regexes that never match anything."""

    matchdates    = {}
    newmatchdates = {}
    if (os.path.isfile("%s/matchdates" % (statefiles))):
        matchdate_file = open("%s/matchdates" % (statefiles),"r")
        for line in matchdate_file:
            line.strip()
            (date,currentregex)      = line.split('\t')
            matchdates[currentregex] = date
    # Loop over our regex strings and initialize any that aren't
    # already in matchdate to now so we have something to compare with
    # in the future.
    for currentregex in regexes:
        if currentregex.line not in matchdates:
            matchdates[currentregex.line] = int(time.time())
    # Now let's loop over matchdates and remove any matchdates that do
    # not have a corresponding regex in the ignore file. If it is not
    # in the ignore file it will never match again but since a time it
    # last matched was recorded for the line in matchdates we will try
    # to remove it from ignore.conf and send the user a notice that it
    # was removed every time the program is run. Very annoying.
    for key in matchdates.keys():
        for currentregex in regexes:
            if (key == currentregex.line):
                newmatchdates[currentregex.line] = matchdates[currentregex.line]
    return newmatchdates


def save_matchdates(matchdates,statefiles):

    """ Save it all back out to a file for next time while ensuring
    that we only write out things that were actually in our
    local_ignore. """

    regexes = load_ignore(config.installdir, config.macros, "local_ignore.conf")
    matchdate_file = open("%s/matchdates" % (statefiles),"w")
    for key in matchdates.keys():
        for regex in regexes:
            if regex.line == key:
                matchdate_file.write("%s\t%s" % (matchdates[key], key))


def process_matchdates(matchdates,config):

    """ Check for any regexes that haven't matched in a certain amount
    of time and add them to the events to report. They are candidates
    for deletion from the ignore file and possibly typos."""

    for key in matchdates.keys():
        if time.time() - int(matchdates[key]) > config.matchdays*60*60*24:
            events.append("Unmatched in %d days, removing: " % config.matchdays + key.st
rip())
            # If remove is true add this to our list of lines to
            # remove later.
            if config.remove:
                remove_lines.append(key)
    if config.remove:
        remove_ignores(config,matchdates)
    # Save our matchdates back out so we can compare next time
    save_matchdates(matchdates,config.statefiles)

def remove_ignores(config,matchdates):

    """ If remove is true we now remove the lines we stored in
    remove_lines from the local_ignore.conf file and write them out to
    the removed_ignores file just to keep a record of what we removed."""
```



```
    local_ignores   = load_ignore(config.installdir, config.macros, "local_ignore.co
nf")
    new_ignorefile  = open("%s/local_ignore.conf" % config.installdir, 'w')
    removed_ignores = open("%s/removed_ignores" % config.statefiles, 'a')
    for local_ignore in local_ignores:
        if local_ignore.line not in remove_lines:
            new_ignorefile.write(local_ignore.line)
        else:
            removed_ignores.write(local_ignore.line)
            try:
                del matchdates[local_ignore.line]
            except KeyError:
                pass
    new_ignorefile.close()
    removed_ignores.close()


def process_file(file,config,regexes,matchdates):

    """ Go through each logfile processing each line to look for
    interesting things starting from where we left off last time and
    recording where we leave off this time. """

    if (os.path.isfile(file)):
        fd = open(file,"r")
        offset = get_last_offset(file,config.installdir,config.statefiles)
    else:
        return
    fd.seek(int(offset))
    if VERBOSE: print "Processing file %s" % file
    for line in fd:
        # We don't touch matchdates in process_file and just pass it on to proce
ss_line.
        # Any way to avoid passing it in here in the first place?
        process_line(line,regexes,matchdates)
    write_last_offset(file,fd,config.statefiles)


def main():
    me = SingleInstance()
    regexes    = load_regexes(config.installdir,config.macros)
    matchdates = load_matchdates(config.statefiles,regexes)
    for file in config.logfiles:
        process_file(file,config,regexes,matchdates)
    insert_tokens(events,config.macros)
    if config.matchdays:
        process_matchdates(matchdates,config)
    send_mail(events,config.email,config.smtp_server)


if __name__ == "__main__":
    if "-v" in sys.argv:
        VERBOSE = 1
    else:
        VERBOSE = 0
    main()
```