NAME

logscan - scan log files for problems & display suspicious areas in context

SYNOPSIS

logscan -k <KIND> *<OPTIONS>* *<FILES-TO-SCAN>*

SHORT OPTION LIST

```
 -?          short help (this text)
 -banner     Indicate PASS/FAIL as a banner
 -c <RANGE>  context specificaton (default 3..20)
 -d [<FILE>] dump parsed rules in a compiled format to file (default
logscan.rules)
 -e <EXTN>   file name extension used for rules (default '.rules')
 -f <FILE>   include file containing auxiliary rules
 -F <FILE>   include file containing auxiliary rules if present
 -h          display manpage and exit
 -html       produce an html formatted report
 -j          justify rule by displaying rule number in output
 -k <KIND>   base type of rules (Default 'default')
 -keep <N>   maximum number of logfiles to keep (default 1)
 -l <FILE>   log file for results (default $tool.log)
 -man        output manpage to file $tool.1 and exit
 -n          no context, just message pointers
 -o <FILE>   same as -l
 -p <PATH>   search path for rules files (default '/usr/local/etc:.')
 -passfail   indicate PASS/FAIL status as single line message
 -q          quiet (minimal runtime messages)
 -tee        display messages to screen and logfile simultaneously
 -v          verbose (maximal runtime messages)
 -V          display tool version
 -w[COL]     line wrap at COL (default off, 78 if specified w/o COL)
 -x <RULE>   explicit rule (should be quoted)
 -X          exact matching to preclude any allowances
 -INSTALL    simple installation option
 -XL         list built-in extractable files
 -XT <NAME>  Extract build-in file
```

DESCRIPTION

LOGSCAN filters log files with the intent of highlighting "features" that may be important to the user. "Features" refer to log information that may indicate errors or problems indicated by the log file; however, hidden by virtue of the volume of data frequently found in log files from logscans such as *Synopsys(TM)* Design Compiler.

To support a variety of log file types, **LOGSCAN** uses "rules" files that describe text patterns which detect the "features". Output from **LOGSCAN** shows line numbers and a critical number of "context" lines preceding the features.

Novel uses of this tool include creating rulesets that check coding conventions and light lint on source code.

OPTIONS

-?

Short help.

-banner

Indicate PASS/FAIL status as a banner.

-c

Specify the number of lines of context as a range (min..max).

-d [<FILE>]

Dump parsed rules in a compiled format (Perl) to <*FILE*>. Default to *logscan.rules*. This may be used to speed up rule compilation; although, it won't gain much performance. Log file scanning is the most time intensive portion of **logscan**.

-e <*EXTN*>

Extension (<*EXTN*>) used for rules files. Determines filename of rules file in conjunction with **-k** option. Default '.rules'.

-f <*FILE*>

Include file containing auxiliary rules. Use **-k** to specify the base rules.

-F <*FILE*>

Include file containing auxiliary rules if present. In other words, unlike **-f**, this option won't fail if the file is missing. Useful for scripts or generic makefiles.

-h

Display manpage and exit.

-html

Produces an HTML formatted report for use with web browsers. This option requires the presence of **vim 6.0** or above and utilizes the **2html.vim** script in conjunction with **logscan.vim**.

-j

Justify rule by displaying rule number in output report. Useful if you don't know why logscan is complaining because the message is too terse. This will usually require dumping the ruleset to interpret (see **-d** option).

-INSTALL

Simple installation.

-k <*KIND*>

Kind of base rules to be used. Determines filename of rules file in conjunction with **-e** option. Default 'default'. This option is normally specified.

-keep <*N*>

Maximum number of old logfiles to keep. Default 1.

-l <*FILE*>

Report filename to save results in. Default *logscan.rpt*.

-man

Output manpage to file logscan.1 for installation and exit.

-n

No context, just message pointers. In other words, don't output context and other useful information. Useful for some editors as "tag" files.

-o <*FILE*>

Same as **-l**.

-p <*PATH*>

Search path for rules files. Default '/usr/local/etc:.'.

-passfail

Indicate PASS/FAIL status textually.

-q

Quiet (minimal runtime messages)

-tee

Display messages to screen and logfile simultaneously.

-v

Verbose (maximal runtime messages)

-V

Display logscan version.

-x

Explicit rule (should be quoted).

-X

Exact matching -- preclude any allowances. See '**allow**' for more information.

-XL

List extractable files (using **-XT).**

-XT <*NAME*>

Extract <*NAME*>d file. For installation or examples.

INVOCATION EXAMPLES

```
% dc_shell -f synth.dcs >synth.log
% logscan -k synopsys synth.log

% verilog -f rtlsim.mft -l rtlsim.log
% logscan -k verilog -p "/usr/local/etc:../:./" \
  -F my.rules rtlsim.log

% setenv LOGSCAN "-k ignore -p /corp/lib:/proj/lib -f drc.rules"
% design_rule_check mydesign.data
% logscan mydesign.err
```

OUTPUT DESCRIPTION

Logscan outputs a minimal amount of information to the terminal (unless you specify **-tee**). The most import thing is the summary of errors or warnings (message events) found. Details of the scan are kept in the *logscan.rpt* file. This file starts out with a description of how the program was invoked and its version.

After the header information and parsing the rules, **logscan** outputs each message event with its context. The following is a sample *logscan.rpt* `*error*' message:

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
ERROR sample2.log, 19: Command failed
Context tag is NONEMPTY
--------------------------------------------------
|   17|{you are here}
|   18|find self
:   19|{}
```

Each message starts out with a message separator and message classification. The classification includes information about the file name and line number where the problem ocurred.

This is followed by the actual text of the message and surrounding lines. Lines include the line number in case you should need to look further into this message. The actual line where the message event was detected in noted with a colon (:) in the first column.

RULE DESCRIPTION

Good rules make the difference between the sucessful use of **logscan** and failure. The following is a loose description of the syntax for specifying rules. Hopefully, this will be sufficient to get users started writing their own rules. Feedback on this documentation is appreciated.

COMMENTS

Rule files should have comments embedded in them to aid the reader in understanding what or why the rules are. Although, rule syntax is fairly straightforward, it's not always obvious. **Logscan** accepts only full line comments in any of shell, Verilog or VHDL syntaxes. Thus the following are valid comments:

```
# This is a shell style comment.
// This is a Verilog/C++ style comment.
-- This is a VHDL/Ada style comment.
```

BASIC MESSAGES

For purpose of this dicusssion, consider the following four lines of logfile text. The trailing tilde (**~**) marks the end of each line.

```
+----------------------------------------------------------+
|Rythum Verilog-Excel VERSION 5.1.3~                       |
|Some regs block in this code~                             |
|ERROR: missing module name on line 200~                   |
|WARNING: non-blocking and blocking assignments to reg Y.~|
|Finish Rythum Verilog-Excel on Tue Mar 5, 2000 at 15:01~ |
+----------------------------------------------------------+
```

Most **logscan** rules have the basic syntax of:

```
RULE_TYPE CONDITION PATTERN [ACTION_OR_OPTION]
```

Basic RULE_TYPE's are any of the keywords: '**fatal**', '**severe**', '**error**', '**warning**', '**alert**', '**note**', and '**info**'. These affect the classification of an event and the return status code.

Conditions are one of '**if**' or '**unless**'. The '**if**' is the most basic condition. If the event occurs in the file being scanned, the action will occur immediately. **unless** handles expectations and exceptions (discussed later).

IMPORTANT: **unless** is a very special case used only for ensuring items are not omitted. This feature does not work well with **min**.

Patterns are specified in one of several manners. Which manner is specified with one of the keywords '**equals**', '**contains**', '**firstword**', '**words**', or '**matches**'. **Equals** requires that every character in a particular line must entirely equals the text provided. For example,

```
note if equals "Rythum Verilog-Excel VERSION 5.1.3"
```

would look for a line that entirely equals the text between the quotation marks and not one more or less anywhere on the line. This matches line one (1) of our sample text.

Contains allows the text to occur anywhere in the line. Thus,

```
info if contains "Verilog"
```

matches lines 1 and 5.

firstword and **words** match only whole words. Thus,

```
error if firstword "ERROR"
warn  if words "blocking"
```

matches lines 1 and 3 respectively.

matches is the most general form of matching and uses full Perl regular expression syntax. Thus,

```
severe if matches /DANGER:.*line \d+/
```

matches all lines with the string "DANGER:" following by "line " and a number. In this example, it matches none of the sample text.

EXPECTATIONS

Expectations are an important part of log file checking. Good examples are (1) ensuring the right version of the tool was used, and (2) ensuring the tool exited gracefully (ie. machine didn't crash in the middle). This type of checking is handled with the '**unless**' condition. Two examples corresponding to the above illustrate this best:

```
severe unless matches "Rythum Verilog-Excel VERSION 5.1.3"
severe unless firstword "Finish Rythum Verilog-Excel" max 1
```

Notice the addition of the **max**imum clause to ensure there is only a single run. This just in case a single log file gets appended to by multiple runs.

EXCEPTIONS

Suppose all errors are of the basic form "ERROR:"; however, the tool reports unconnected ports as an error and for some reason you have two unnconnected ports that are intentional (e.g. the QBAR output of some flip-flops are unused). In this situation, you want to catch all the errors except these two. This situation uses the the '**unless**' condition combined with the **only** clause. For example,

```
error unless matches {^ERROR:.*unconnected QBAR\b} only 2
```

If only two lines match this pattern, then logscan will silently proceed; however, if there are too many or too few, then an error will be noted. This differs from basic '**unless**' in that the "too many" situation is checked on the fly and the error message may appear if there are excess matches. This may also be accomplished using the **max** clause. In fact **only** is identical to **min N max N**.

CONTEXT

Frequently, messages in the text are only errors if the context is appropriate. For this **logscan** allows you to identify passages of text that establish the different contexts using the '**context**' rule and '**context**' clause. For example, Synopsys Design Compiler's **compile** command returns a line with a numeric exit status which is 1 if successful. Thus,

```
NONZERO: context if firstword "compile"
error if equals "0" context NONZERO
```

Another situation might be a multiple phase log file containing several tools' output. In this situation you could identify the beginning text in each tool (hopefuly unique) to distiguish different classifications of errors.

```
START: context if equals "Beginning run"
ANALYZE: context if firstword "analyze" context START
```

```
SIMULATE: context if firstword "Simulating" context ANALYZE
POST: context if firstword "Beginning post-processing" context SIMULATE
FINISH: context if equals "Finished run" context POST
```

Notice the that context transisitions specified are orderly. Of course this doesn't have to be the case. You must ensure that every statement has the appropriate **context** qualifier.

Context also causes the first name of the context to be displayed and as many lines as possible up to the upper context line limit. This significantly aids diagnosis of a problem. See **-c** command-line option or **limit** rule.

IMPORTANT: There is only *one* (1) active context at any point in time. Think of it as the *state* varaible of a finite state machine. You can change it dependently or not (e.g. reset might be independent).

Finally, there is a '**goto**' qualifier that can be used to change context in conjunction with informational messages. This avoids having both an **info** and a **context** rule for the same pattern, and improves execution performance.

```
info if firstword "Entering second stage" context STAGE1 goto STAGE2
```

CLAUSES

Use of the '**and**' qualifier permits additional requirements in the form of expressions computed on subfields of the matching expression. For instance, you might allow several versions of a tool to be used as long as they are greater than a particular one:

```
error unless matches {Version (\d+\.\d+)} and {=$1 > 2.3=}
```

Expression must be enclosed in **{= =}** and conform to Perl requirements. Additionally, the variables **$&**, **$+**, **$1**, **$2**, **$3**, **$4**, **$5**, and **$6** are available.

ALLOWANCES

Use of the '**allow**' qualifier permits so called "soft matching" based on expressions computed on subfields of the matching expression. For example, you may wish to ignore the time stamp in a simulation most of the time, but want to know if it changed when issuing an exception:

```
error unless matches {WARNING at time (\d+ ns): counter cleared} allow
{=$1 eq "52 ns"=}
```

Expression must be enclosed in **{= =}** and conform to Perl requirements. Additionally, the variables **$&**, **$+**, **$1**, **$2**, **$3**, **$4**, **$5**, and **$6** are available.

CONTROLLING CONTEXTS

By using the **enable** and **disable** clauses, you can also turn a set of context controlled rules on and off. This allows you suspend error messages for a certain portion of the file.

```
enable  USER_MESSAGES if contains "Start user messages"
disable USER_MESSAGES if contains "End user messages"
```

CUSTOM MESSAGES

By default, **logscan** will display error messages indicating the failing pattern; however, you may specify your own message to accompany any failures or info using the '**msg**' clause.

```
error if words "CRC error" msg "Found Cyclic Redundancy Check error"
```

Additionally, messages may contain references to some special "variables" and custom variables. Special variables include:

$&

the matching text

$1

matching text inside the 1st parenthesis pair

$2

matching text inside the 2nd parenthesis pair

$3

matching text inside the 3rd parenthesis pair

$4

matching text inside the 4th parenthesis pair

$5

matching text inside the 5th parenthesis pair

$6

matching text inside the 6th parenthesis pair

$tag

name of the rule (hopefully a unique tag label)

$typ

type of rule (e.g. 'error' or 'warn')

$cmp

comparison type

$pat

pattern being searched for

$cnt

number of times matched

$CNT

pluralized number of times matched

$max

maximum requirement

$min

minimum requirement

$RNG

expected range (min .. max)

An example of usage might be:

```
 CHECK1: error if matches {^Total of (\d+) failing packets} \
   and {= $1 != $expected =} \
   msg "Failed $1 packets out of $expected in $tag"
```

OTHER CONTROLS

A few other controls are available to help control logscan and interpret the results.

The '**echo**' rule simply outputs text during rule parsing. Use this to output a title for the rules and the version.

```
echo "Rythum simulation rules version 1.2"
```

Use the '**limit**' rule to control **logscan**'s context buffer. Perhaps you want to have at least three (3) lines of context, but no more than ten.

You may include other rules files either during parsing or on-the-fly with one of the rules '**use**', '**require**' or '**use**'. The '**use**' also resets all the rules. This is an alternate way to accomplish major tool context switches. At the end of a tool's output you could switch to a default rules set that attempts to figure out where to go next. Thus,

```
use "default.rules" if equals "Finished."

use "dcshell.rules" if contains "Design-Compiler"
use "verilog.rules" if contains "Verilog-XL"
use "vcs.rules" if contains "Synopsys VCS"
```

Use the '**version**' rule to require a specific version of **logscan**.

Use **verbose** or **quiet** respectively to increase or decrease the amount of information sent to STDOUT. All information is recorded in the log file.

You may also specify **logsan**'s logfile name with the '**log**' rule.

NAMES

If you don't like the keywords supplied by logscan, you can supply your own in the form of aliases. This might be useful for foreign languages too. There are some predefined aliases too. For example,

```
alias firstwords=firstword
```

CONTROLLING RULES

It is possible to disallow or reset entire classifications of rules (i.e. RULE_TYPE's. Once disallowed, a keyword can never be reallowed; however, if you setup an alias it is possible to use the alias. An administrator might use this capability.

USING CALCULATIONS

Sometimes it is necessary to gather statistics and make error judgements at the end. This is accomplished using the **eval** clause in conjunction with the **unless expr** operation.

```
count if matches {ERROR: CRC discarded (\d+) blocks of (\d+) bytes} \
     eval {= $my_count += $1 * $2 =}

# Following takes place after entire file scanned
error unless expr expr {= $my_count != 15 =} \
     msg "Discarded $my_count rather than 15 expected"
```

Note that the **eval** clause behaves slightly differently for **if** vs. **unless expr** operations. In the former, **eval** is executed only when the condition is true, but for **unless expr** the **eval** is executed unconditionally.

FINAL THOUGHTS

Use the **post** condition to issue messages unconditionally after processing. This is useful to display the results of calculations.

```
info post msg "Saw $crc_count CRC blocks"
```

RULE SYNTAX

The following is the concise syntax of rule for logscan. Rules are contained in one or more files that are specified either on the command line (via **-k**, **-f** or **-F**) or via other rule files invoking them (via **use**, **require** or **include** statements).

Rules are read in REVERSE order of presentation to allow local overrides.

Rules are restricted to a single line unless the last character of the line is a backslash '\' or in the middle of a multi-line SCAN_PATTERN. See RULE DESCRIPTION and RULE EXAMPLES for clarification.

```
#  COMMENT
// COMMENT
-- COMMENT
[<TAG>:] <RULE_TYPE> if     <RULE_EXPRESSION> <RULE_CLAUSE>..
[<TAG>:] <RULE_TYPE> unless <RULE_EXPRESSION> <RULE_CLAUSE>..
         <RULE_TYPE> post   <RULE_CLAUSE>..
[<TAG>:] enable|disable <TAG_PATTERN> [if <CLAUSE>]
define NAME TEXT
disallow|clear <RULE_TYPE>[s]
use|require|include <FILE>
echo <TEXT>
version <NUMBER>
quiet
verbose
log <FILE>
limit <NUMBER>..<NUMBER>
alias <NEW>=<OLD>
```

NOTE: **if** is processed only during scanning. **unless expr** and **post** are processed only after scanning. Other **unless** are examined both during and after.

RULE TYPES

```
NAME    RETURN  ACTION
----    ------  ------
fatal   128     message & exit program
severe  64..127 message & next line
error   1..63   message & next line
warning 0       message & next line
alert   0       message (considered a warning)
note    0       message & next line
info    0       message (considered a note)
count   0       count
ignore  0       next line
context -       load context register with line <TAG>
require -       append rules
include -       append rules if found
use     -       read new set of rules
```

RULE EXPRESSIONS

```
equals    {ENTIRE_LINE}
contains  {TEXT}
firstword {WORD}
words     {WORD...WORD}
matches   {PERL_REGULAR_EXPRESSION}
expr      {PERL_EXPRESSION}
```

NOTE 1: **{}** may be replaced with any pair of (), [], <> or simple "", '', //.

NOTE 2: TEXT or EXPRESSION may extend over multiple lines (be careful).

NOTE 3: ('=', '?', '~') may be used instead of ('equals', 'contains', 'matches') respectively.

NOTE 4: **expr** is only valid in conjunction with **unless**.

RULE CLAUSES

```
 allow {=EXPR=}         allows Perl EXPR to be false, but notes it
 eval  {=EXPR=}         evaluates Perl EXPR
 and {=EXPR=}           additional constraint based on evaluation of Perl
EXPR
 context <TAG_PATTERN>  context must match <TAG_PATTERN>
 goto <TAG>             changes context to specified <TAG>
 enable <TAG_PATTERN>   enable tagged rules matching <TAG_PATTERN>
 disable <TAG_PATTERN>  disable tagged rules matching <TAG_PATTERN>
 msg <TEXT>             display <TEXT>
 only <NUMBER>          minimum and maximum occurence of <NUMBER> times
 min <NUMBER>           must appear at least <NUMBER> times to be
considered
 max <NUMBER>           ignored if appears more than <NUMBER> times
 show <NUMBER>[ more[ lines]]
 always                 force additional interpretation
```

RULE NOTES

Rules are read in REVERSE order.

Because rules are interpreted strictly in the order encountered, and because rule of class: {severe, error, warning, note, info, & ignore} cause immediate skipping to the next line in the log file, special care should be taken when applying these rules in case they might disable a 'context' rule.

As a potential mechanism to ensure contexts or other important rules, the 'always' clause may be added to force a rule's interpretation even if was indicated to be skipped. This option is powerful, and equally as dangerous in the reverse sense.

'**use**' resets the rules (ie. clears out all patterns) & must exist

'**require**' must find the specified file

'**include**' merely adds to the rule set (ie. no complaints about missing files)

TEXT and EXPR may reference $&, $+, $1, $2, $3, $4, $5, and $6 which work as in Perl. For example, refers to the first matched text in parentheses. These only work for 'matches' (ie. regular expression) patterns.

RULE EXAMPLES

```
 my.rules
 +-------------------------------------------------------------------+
 |# The following are some typical rules used with Synopsys          |
 |  verbose                                                          |
 |  # First setup contexts                                           |
 |  NONZERO: context if firstword "compile"                          |
 |  NONZERO: context if firstword "link"                             |
 |  KEYWORD: context if firstword "if"                               |
 |  KEYWORD: context if firstword "while"                            |
 |  KEYWORD: context if contains  "} else {"                         |
 |  # Now handle the errors associated with contexts                 |
 |  error if matches /^0\$/  msg "Command failed" context NONZERO  |
 |  error if matches /^{}\$/ msg "Command failed" context NONEMPTY |
 |  # Handle errors of a more general nature                       |
 |  severe if matches /^ABORT\b/                                   |
 |  warn  if firstword "WARNING" msg "User warning detected";      |
 |  error if firstword "ERROR" msg "User error detected";          |
 |  error if contains "latch inferred" show 2 more lines           |
 |  info if matches /inferred (\d+)/ and {=\$1>9=}                 |
 |  # Following illustrates various multi-line features            |
 |  REVCHECK: \                                                    |
 |    error unless matches {.*rev\. 5.1                            |
 |\s+15 Dec} msg "Must use rev \$1." \                             |
 |           disable REVCHECK                                      |
 |  # Finally add the exceptions                                   |
 |  ignore if words "No latch inferred"                            |
 |  # - require 2 latches                                          |
 |  error unless matches {latch inferred} only 2                   |
 +~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~+
```

ENVIRONMENT

LOGSCAN contains command line options that are interpreted prior to parsing the command line.

FILES

Rules files are selected two different ways. First, an path is searched for files of the name <*KIND*>.<*EXTN*>. The default path, basename, and filename extension may be changed with the **-p**, **-k**, and **-e** command line options.

BUGS

None known

NOTES

The sanity counter outputs a character every 15 seconds unless **-q** (quiet) is in effect.

LOGSCAN may be under the GPL via the Internet at URL: <http://www.hldwizard.com/logscan.tar.gz>

AUTHOR

David C. Black <dcblack@hldwizard.com>

COPYRIGHT

Copyright (C) 1997-2001,2008 by David C. Black <dcblack@hldwizard.com> All rights reserved.

LICENSE

This software, **logscan**, is CharityWare in the manner of Bram Moolenaar's vim text editor (Vi IMproved). You may use and copy it as much as you like, but you are encouraged to make a donation to a non-profit charity organization addressing poverty-hunger, poverty-housing, or racial justice. Payment should be made directly to the charity of your choice. Please extract the LICENSE details using the command: logscan -XT LICENSE.

Redistribution and in source and binary forms are permitted provided that the above copyright notice and license are duplicated in all such forms and that any documentation, advertising materials, and other materials related to such distribution and use acknowledge that the software was developed by David C. Black, High Level Design Wizard. <http://www.hldwizard.com>

There are no restrictions on distributing an unmodified copy of Logscan. Parts of Logscan may also be distributed, but this text must always be included. You are allowed to include executables that you made from the unmodified Logscan sources, your own usage examples and Logscan scripts.

If you distribute a modified version of Logscan, you are encouraged to send the maintainer a copy, Never-the-less, you are encouraged to send bug reports to me, David C Black <dcblack@hldwizard.com>. Registered users may expect some support for severe bugs as determined by me. Enhancements and/or feature requests will be noted; however, no commitments will be made. There are no guarantees that response will be timely. Please include the word **LOGSCAN** in the Subject header of your email.

VERIFICATION

All valid releases of this logscan are accompanied with a PGP signature file logscan.pgp to verify you have an unmodified copy. Also, the license file may be extracted as a PGP signed document.