GNU Direvent

version 5.3, 29 December 2021

Sergey Poznyakoff.

Short Contents

Table of Contents

1 Introduction

GNU `direvent` monitors events in file system directories. For each event that occurs in a set of pre-configured directories, the program calls an external program associated with it, supplying it the information about the event and the location within the file system where it took place.

GNU `direvent` provides an easy way to configure your system to react immediately if certain files undergo changes. This may be helpful, for example, to track changes in important configuration files.

Interfaces for tracking changes to file systems are highly system-specific. GNU `direvent` aims to provide a uniform and system-independent command-level interface. As of version 5.3 `direvent` works with modern Linux kernels (since v. 2.6.13) and BSD systems (FreeBSD, NetBSD, OpenBSD, Darwin).

2 Overview

GNU `direvent` monitors a set of directories on the file system and reacts when a file system event occurs in any of them. Directories and events to monitor are specified in the configuration file. When an event occurs, the program reacts by invoking an external command configured for that event.

File system events can be divided into two major groups. The *system-dependent events* are specific for each particular kernel interface. In the contrast, *generic events* don't depend on the underlying system. They provide a higher level of abstraction and make it possible to port GNU `direvent` configurations between various systems and architectures.

The generic events are:

`create` [generic event]
A file was created. This includes files moved from another directory.

`delete` [generic event]
A file was deleted or moved to another directory.

`write` [generic event]
A file was written to. This does not imply that the file was closed.

`change` [generic event]
A file was modified and closed. This is a *compound* event, i.e. it is delivered when a system event that means that the file opened for writing was closed (`CLOSE_WRITE`), is delivered for a file on which one or more `write` events have been previously delivered. As such it depends on the operating system ability to deliver the `CLOSE_WRITE` event. Linux and FreeBSD have this ability. Many other systems, such as NetBSD and Darwin, don't.

`attrib` [generic event]
File attributes have changed. This includes changes in the file ownership, mode, link count, etc.

A *watcher* is a configuration entity that associates a set of directories with a set of events and instructs `direvent` to run a specified external command when any of these events occur in any of these directories. This external command (called a *handler*) can obtain information about the event that triggered it from the environment variables, or from its command line.

Watchers are defined in the configuration file, which `direvent` reads at startup. The following outlines its syntax:

Three types of comments are allowed: inline comments, that begin with a '`#`' or '`//`' and extend to the end of line, and multi-line comments, which comprise everything enclosed between '`/*`' and '`*/`'. Comments and empty lines are ignored. Whitespace characters are ignored as well, except as they serve to separate tokens.

A token is a string of consecutive characters from the following classes: alphanumeric characters, underscores, dots, asterisks, slashes, semicolons, commercial at's, and dashes.

Any other sequence of characters must be enclosed in double quotation marks in order to represent a single token.

Adjacent quoted strings are concatenated.

A configuration statement consists of a keyword and value separated by any amount of whitespace and is terminated with a semicolon. A block statement is a collection of statements enclosed in curly braces.

A watcher is declared using the following block statement:

```
watcher {
    path pathname [recursive [level]];
    file pattern-list;
    event  event-list;
    command command-line;
    user name;
    timeout number;
    environ { ... };
    option string-list;
}
```

Each `watcher` statement instructs `direvent` to monitor events from *event-list* occurring in directories specified by *pathname*s in `path` statements (any number of `path` statements can be given). When any such event is detected, the supplied *command-line* will be executed.

Each directory defined with the `recursive` keyword will be watched recursively. This means that for each subdirectory created in it, `direvent` will install a watcher similar to that of its parent directory. Optional *level* statement can be used to set up a cut-off nesting level, beyond which the recursive operation is disabled.

It is a common practice for the `path` statement to refer to a directory. However, it is not a requirement. The *pathname* argument can as well point to any other type of file[1]. Moreover, it is not required to exist, either. If it does not, GNU `direvent` will remember the watcher definition and will set it up when the *pathname* is eventually created.[2]

The rest of statements are optional. The `file` statement instructs GNU `direvent` to react only if the event concerned the file whose name matches one of the patterns given in its argument. The `user` statement can be used to execute the *command-line* as the user *name* (provided, of course, that `direvent` is started with root privileges). The `timeout` specifies the maximum amount of time (in seconds) the command is allowed to run. It defaults to 5. The `environ` statement modifies the command environment.

[1] Obviously, the '`recursive`' keyword is valid only if *pathname* is a directory.

[2] See [path], page 18, for a detailed description.

Finally, the `option` statement supplies additional options. It can be used, for example, to divert the command's output to syslog.

3 Quick Start

Let's suppose you have a directory where users can upload their files and you want these files to be processed right after upload, in real time. Let this directory be `/home/ftp/incoming` and the program to process the upload be `/usr/bin/upload`. Let's also suppose that this program expects name of the uploaded file as its argument.

To make `direvent` handle this task, you would need to create a watcher for the upload directory which would handle the '`create`' event:

```
watcher {
    path /home/ftp/incoming;
    event create;
    # more statements follow...
```

On this event, the watcher is to invoke `/usr/bin/upload` with the name of the created file as an argument. To make it possible, the `direvent` configuration file provides *macro variables*, which can be used in the `command` argument at configuration time and which are expanded to the actual values before the command is executed. Macro variables are referred to using the same syntax as shell variables: a dollar sign followed by the variable name, optionally enclosed in curly braces. The '`file`' variable is expanded to the name of the file for which the event is reported. This name is relative to the current working directory which, by the time the handler is executed, is set to the directory where the event occurred. Thus, the handler can be configured as:

```
    command "/usr/bin/upload $file";
```

To summarize, the watcher declaration is:

```
watcher {
    path /home/ftp/incoming;
    event create;
    command "/usr/bin/upload $file";
}
```

Before invoking the handler, the following operations are performed:

1. The current working directory is set to the directory where the event occurred.
2. If the global `environ` statement is present, the current environment is modified according to its rules.
3. If the `environ` statement is present in the `watcher` block, the environment is further modified according to its rules. (see Section 5.5 [environ], page 15)
4. The standard input is closed.
5. If the '`stdout`' option is supplied, the standard output is captured and redirected to the syslog. Otherwise it is closed.
6. If the '`stderr`' option is supplied, the standard error is captured and redirected to the syslog. Otherwise it is closed.

7. File descriptors above 2 are closed.
8. Macro variables are expanded in the command line. Unless the `shell` option is set, environment variables are expanded as well. See Section 5.2 [variable expansion], page 12.
9. If the `shell` option is set, the handler is invoked via the shell, as `$SHELL -c "`*`command`*`"`.

   Otherwise, word splitting is performed on the resulting command line. The first word is treated as the pathname of the program, which is then invoked via the `execve` system call.

4 Invocation

The invocation syntax is:

```
direvent [options] [config]
```

where *options* are command line options discussed below and optional *config* supplies the configuration file to use instead of the default `/etc/direvent.conf`.

The options are:

`-d`
`--debug` Increase debug level.

`-F name`
`--facility=name`
Set syslog facility.

`-f`
`--foreground`
Remain in foreground.

`-I dir`
`--include=dir`
Add *dir* to the beginning of the include search path (see [include search path], page 9).

`-l prio` While connected to a terminal, `direvent` outputs its diagnostics messages to stderr and, if configured, to syslog. This option limits the amount of information output to the standard error. The *prio* argument is one of the following priorities (in order of increasing severity): '`debug`', '`info`', '`notice`', '`warning`', '`err`', '`crit`', '`alert`', '`emerg`'. When this option is given, only messages with the priority level equal to or greater than *prio* will be duplicated on the standard error.

`-P file`
`--pidfile=file`
Upon successful startup store the PID of the daemon process in *file*.

`-T command`
`--self-test=command`
Run in *self-test mode*. In this mode, `direvent` starts external command supplied as the argument to this option and continues running until the command exits. If *command* terminates normally, `direvent` exits with the code returned by it. If *command* terminates on signal, `direvent` exits with code '`0`' if this signal was `SIGHUP`, and with code '`2`' otherwise.

The *command* can include any command line options or arguments, provided that it is properly quoted. It is invoked as

`/bin/sh -c` *`command`* in the environment of the parent `direvent` process.

This mode is used in `direvent` test suite. The idea is to configure the handler (see [handler], page 2) so that it sends `SIGHUP` to *command* before exiting. To this effect, the special macro variable `$self_test_pid` is defined (see Section 5.2 [variable expansion], page 12) to the PID of the running *command* process. For example, consider configuration file `test.conf`, which contains the following:

```
watcher {
    path /tmp;
    command "/bin/kill -HUP $self_test_pid";
}
```

Then, the following command can be used to check whether `direvent` correctly reacts on file creation in the watched directory:

```
$ direvent --foreground \
    --self-test 'touch /tmp/file && /usr/bin/sleep 20 && exit
    test.conf
```

The command will return '`0`' if the handler was invoked, and '`1`' if it was not.

`-t`
`--lint` Check configuration file for errors and exit.

`-u` *`name`*
`--user=`*`name`*
Run as this user. This option overrides the `user` configuration statement (see Section 5.3 [general settings], page 14).

The following options are *informative*. They cause the program to display the requested piece of information and terminate:

`-H`
`--config-help`
Show configuration file summary.

`-h`
`--help` Give a short usage summary.

`--usage` Display available command line options.

`-V`
`--version`
Print program version.

5 Configuration

5.1 Configuration Syntax

The configuration file consists of statements and comments.

There are three classes of lexical tokens: keywords, values, and separators. Blanks, tabs, newlines and comments, collectively called *white space* are ignored except as they serve to separate tokens. Some white space is required to separate otherwise adjacent keywords and values.

5.1.1 Comments

Comments may appear anywhere where white space may appear in the configuration file. There are two kinds of comments: single-line and multi-line comments. *Single-line* comments start with '`#`' or '`//`' and continue to the end of the line:

```
# This is a comment
// This too is a comment
```

Multi-line or *C-style* comments start with the two characters '`/*`' (slash, star) and continue until the first occurrence of '`*/`' (star, slash).

Multi-line comments cannot be nested. However, single-line comments may well appear within multi-line ones.

5.1.2 Pragmatic Comments

Pragmatic comments are similar to usual single-line comments, except that they cause some changes in the way the configuration is parsed. Pragmatic comments begin with a '`#`' sign and end with the next physical newline character.

`#include <`*file*`>`
`#include "`*file*`"`

Include the contents of the file *file*. If *file* is an absolute file name, the named file is included. An error message will be issued if it does not exist.

If *file* contains wildcard characters ('`*`', '`[`', '`]`' or '`?`'), it is interpreted as a shell globbing pattern and all files matching that pattern are included, in lexicographical order. If no matching files are found, the directive is replaced with an empty line.

Otherwise, the form with angle brackets searches for file in the *include search path*, while the second one looks for it in the current working directory first, and, if not found there, in the include search path. If the file is not found, an error message will be issued.

Include search path is formed by two directory sets: the user-defined search path, as defined by eventual `-I` (see [include option], page 7) command line options, and the standard include

search path, defined at compile time. The latter can be inspected using the `--help` option.

The order of directories is as follows. First, `direvent` scans any directories given with `-I` options, in the same order as given on the command line. If *file* is not found in any of them, the standard include search path is scanned. It is defined at the compile time and by default consists of two directories:

- *prefix*`/share/direvent/include`
- *prefix*`/share/direvent/5.3/include`

where *prefix* is the installation prefix. The default can be changed when configuring the package. To inspect the actual standard include search path at the runtime, run `direvent --help`, and look for the string '`Include search path:`' in its output.

`#include_once <`*file*`>`
`#include_once` *file*
: Same as `#include`, except that, if the *file* has already been included, it will not be included again.

`#line` *num*
`#line` *num* `"`*file*`"`
: This line causes the parser to believe, for purposes of error diagnostics, that the line number of the next source line is given by *num* and the current input file is named by *file*. If the latter is absent, the remembered file name does not change.

`#` *num* `"`*file*`"`
: This is a special form of `#line` statement, understood for compatibility with the C preprocessor.

5.1.3 Statements

A *simple statement* consists of a keyword and value separated by any amount of whitespace. Simple statement is terminated with a semicolon ('`;`').

The following is a simple statement:

```
standalone yes;
pidfile /var/run/direvent.pid;
```

A *keyword* begins with a letter and may contain letters, decimal digits, underscores ('`_`') and dashes ('`-`'). Examples of keywords are: '`expression`', '`output-file`'.

A *value* can be one of the following:

number
: A number is a sequence of decimal digits.

boolean
: A boolean value is one of the following: '`yes`', '`true`', '`t`' or '`1`', meaning *true*, and '`no`', '`false`', '`nil`', '`0`' meaning *false*.

unquoted string
: An unquoted string may contain letters, digits, and any of the following characters: '`_`', '`-`', '`.`', '`/`', '`@`', '`*`', '`:`'.

quoted string
: A quoted string is any sequence of characters enclosed in double-quotes ('`"`'). A backslash appearing within a quoted string introduces an *escape sequence*, which is replaced with a single character according to the following rules:

Sequence	Replaced with
\a	Audible bell character (ASCII 7)
\b	Backspace character (ASCII 8)
\f	Form-feed character (ASCII 12)
\n	Newline character (ASCII 10)
\r	Carriage return character (ASCII 13)
\t	Horizontal tabulation character (ASCII 9)
\v	Vertical tabulation character (ASCII 11)
\\	A single backslash ('\')
\"	A double-quote.

Table 5.1: Backslash escapes

In addition, the sequence '`\`*`newline`*' is removed from the string. This allows to split long strings over several physical lines, e.g.:

```
"a long string may be\
 split over several lines"
```

If the character following a backslash is not one of those specified above, the backslash is ignored and a warning is issued.

Here-document
: A *here-document* is a special construct that allows to introduce strings of text containing embedded newlines.

The `<<`*`word`* construct instructs the parser to read all the following lines up to the line containing only *word*, with possible trailing blanks. Any lines thus read are concatenated together into a single string. For example:

```
<<EOT
A multiline
string
EOT
```

The body of a here-document is interpreted the same way as a double-quoted string, unless *word* is preceded by a backslash (e.g. '`<<\EOT`') or enclosed in double-quotes, in which case the text is read as is, without interpretation of escape sequences.

If *word* is prefixed with - (a dash), then all leading tab characters are stripped from input lines and the line containing *word*. Furthermore, if - is followed by a single space, all leading whitespace is stripped from them. This allows to indent here-documents in a natural fashion. For example:

```
<<- TEXT
    The leading whitespace will be
    ignored when reading these lines.
TEXT
```

It is important that the terminating delimiter be the only token on its line. The only exception to this rule is allowed if a here-document appears as the last element of a statement. In this case a semicolon can be placed on the same line with its terminating delimiter, as in:

```
help-text <<-EOT
        A sample help text.
EOT;
```

list

A *list* is a comma-separated list of values. Lists are enclosed in parentheses. The following example shows a statement whose value is a list of strings:

```
option (stdout,stderr);
```

In any case where a list is appropriate, a single value is allowed without being a member of a list: it is equivalent to a list with a single member. This means that, e.g.

```
option wait;
```

is equivalent to

```
option (wait);
```

A *block statement* introduces a logical group of statements. It consists of a keyword, followed by an optional value, and a sequence of statements enclosed in curly braces, as shown in the example below:

```
syslog {
  facility local0;
  tag "direvent";
}
```

The closing curly brace may be followed by a semicolon, although this is not required.

5.2 Variable Expansion

Arguments of some configuration statements undergo variable expansion before use. During variable expansion, *variable references* found in string are replaced with the actual values of the corresponding variables.

A variable reference has the form '`$variable`' or '`${variable}`', where *variable* is the variable name. The two forms are entirely equivalent. The form with curly braces is normally used if the variable name is immediately

followed by an alphanumeric symbol, which will otherwise be considered part of it. This form also allows for specifying the action to take if the variable is undefined or expands to an empty value:

${*variable*:-*word*}
: *Use Default Values*. If *variable* is unset or null, the expansion of *word* is substituted. Otherwise, the value of *variable* is substituted.

${*variable*:=*word*}
: *Assign Default Values*. If *variable* is unset or null, the expansion of *word* is assigned to variable. The value of *variable* is then substituted.

${*variable*:?*word*}
: *Display Error if Null or Unset*. If *variable* is null or unset, the expansion of *word* (or a message to that effect if *word* is not present) is output to the current logging channel. Otherwise, the value of *variable* is substituted.

${*variable*:+*word*}
: *Use Alternate Value*. If *variable* is null or unset, nothing is substituted, otherwise the expansion of *word* is substituted.

Two kinds of variables take part in variable expansion: environment variables and *macro variables*. The latter are special variable-like entities defined by `direvent` to carry information about the event and its target file.

`file` [macro variable]
: Name of the file that triggered the event.

`genev_code` [macro variable]
: Generic (system-independent) event code. It is a bitwise OR of the event codes represented as a decimal number.

`genev_name` [macro variable]
: Generic event name. If several generic events are reported simultaneously, the value of this variable is a list of event names separated by space characters. Each name corresponds to a bit in '`$genev_code`'.

`sysev_code` [macro variable]
: A system-dependent event code. It is a bitwise OR of the event codes represented as a decimal number.

`sysev_name` [macro variable]
: A system-dependent event name. If several events are reported, the value of this variable is a list of event names separated by space characters. Each name corresponds to a bit in '`sysev_code`'. See Chapter 6 [System dependencies], page 21, for a list of system-dependent event names.

`self_test_pid` [macro variable]
The PID of the external command started with the `--self-test` option (see [self-test mode], page 7). If `direvent` is started without this option, this variable is not defined.

Statements whose arguments undergo variable expansion are: `command` and `environ` (all substatements and legacy syntax use). Although syntactically both environment and macro variables are treated the same way, there are some subtle differences between them that you should be aware of.

First of all, macro variables are not reflected in the environment of handlers. There are special environment variables for that purpose (see Section 5.5 [environ], page 15).

Secondly, macro variables take precedence before the environment. It is possible, for example, that the *environment* variable '`sysev_name`' is defined in the environment inherited by `direvent` or set using the `environ` statement. To ensure that such improper usage won't affect functionality of the watchers, `direvent` unconditionally deletes from the environment any variables whose names coincide with macro variables.

When used in `command` argument and the `shell` option is set, macro variables are expanded whereas environment variables are not (they will be expanded later by the shell). Consider, for example, the following definition:

```
watcher {
   option shell;
   path "/tmp";
   command "$BINDIR/handler $file $DIREVENT_GENEV_CODE";
}
```

When an event wakes up this watcher, only `$file` will be expanded. Suppose that an event was delivered for file `/tmp/myfile`. Then, `direvent` will run the following command:

```
$SHELL -c '$BINDIR/handler myfile $DIREVENT_GENEV_CODE'
```

The remaining environment variable references will be expanded by the shell.

5.3 General Settings

`user` *`name`* [Config]
Sets the user to run as. The *name* argument must be a name of an existing user.

`foreground` *`bool`* [Config]
Run in foreground.

`pidfile` *`file`* [Config]
Upon successful startup store the PID of the daemon process in *file*.

`debug` *number* [Config]
Set debug level. Valid *number* values are '`0`' (no debug) through '`4`' (maximum verbosity).

5.4 Syslog

While connected to the terminal, `direvent` outputs its diagnostics and debugging messages to the standard error. After disconnecting from the controlling terminal it closes the first three file descriptors and directs all its output to the syslog. When running in foreground mode, its messages are sent both to the standard error and to the syslog.

The following configuration statement controls the syslog output:

```
syslog {
    facility string;
    tag string;
    print-priority bool;
}
```

The statements are:

`facility` *string* [Config]
Set syslog facility. The argument is one of the following: '`user`', '`daemon`', '`auth`' or '`authpriv`', '`mail`', '`cron`', '`local0`' through '`local7`' (case-insensitive), or a facility number.

`tag` *string* [Config]
Tag syslog messages with '`string`'. Normally the messages are tagged with the program name.

`print-priority` *'bool'* [Config]
Prefix each message with its priority.

An example `syslog` statement:

```
syslog {
    facility local0;
    print-priority yes;
}
```

5.5 Environment modification

By default, each handler inherits the environment of the master `direvent` process augmented with the following variables:

`DIREVENT_SYSEV_CODE` [environment variable]
The system-dependent event code (see [sysev_code], page 13).

`DIREVENT_SYSEV_NAME` [environment variable]
The system-dependent event name (see [sysev_name], page 13). If several system-dependent events are delivered, this variable contains their names separated with single horizontal space character.

`DIREVENT_GENEV_CODE` [environment variable]
The generic event code (see [genev_code], page 13).

`DIREVENT_GENEV_NAME` [environment variable]
The generic event name (see [genev_name], page 13). If several generic events are delivered, this variable contains their names separated with single horizontal space character.

`DIREVENT_FILE` [environment variable]
The name of the affected file relative to the current working directory (see [file], page 13).

This environment can be further modified, using the `environ` configuration statement:

```
environ {
   clear;
   keep pattern;
   set "name=value";
   eval "value";
   unset pattern;
}
```

Statements inside the `environ` block define operations that modify the environment. Their arguments undergo variable expansion (see Section 5.2 [variable expansion], page 12). The `clear` and `keep` statements are executed first. Then, the `set` and `unset` statements are applied in the order of their appearance in the configuration.

`clear` [environ]
Clears the environment by removing (unsetting) all variables, except those listed in `keep` statements, if such are given (see below). The `clear` statement is always executed first.

`keep` *pattern* [environ]
`keep "`*name*`=`*value*`"` [environ]
Declares variables matching *pattern* as exempt from clearing. This statement implies `clear`.

The *pattern* is either a variable name or a globbing pattern matching one or more names.

In the second form, the variable will be retained only if it has the given *value*. Note, that the argument must be quoted.

For example, the following configuration fragment removes from the environment all variables except '`HOME`', '`USER`', '`PATH`', and variables beginning with '`DIREVENT_`':

```
environ {
  clear;
  keep HOME;
  keep USER;
  keep PATH;
  keep "DIREVENT_*";
}
```

`set "`*name*`=`*value*`"` [environ]
Assigns *value* to the environment variable *name*. The value is subject to *variable expansion* using the same syntax as in shell. The `set`, `eval`, and `unset` (see below) statements are executed in order of their appearance. For example

```
environ {
  set "MYLIB=$HOME/lib";
  set "LD_LIBRARY_PATH=$LD_LIBRARY_PATH${LD_LIBRARY_PATH:+:}$MYLIB";
}
```

`eval "`*expr*`"` [environ]
Perform variable expansion on *expr* and discard the result. This is useful for side-effects. For example, to provide default value for the `LD_LIBRARY_PATH` variable, one may write:

```
environ {
  eval "${LD_LIBRARY_PATH:=/usr/local/lib}";
}
```

`unset `*pattern* [environ]
`unset "`*name*`=`*value*`"` [environ]
Unset environment variables matching *pattern*.

The *pattern* is either a variable name or a globbing pattern matching one or more names.

In the second form, the variable will be unset only if it has the given *value*. Note, that the argument must be quoted.

E.g., the following will unset the `LOGIN` variable:

```
unset LOGIN;
```

The following statement will unset all variables starting with '`LD_`':

```
unset "LD_*";
```

Notice, that patterns containing wildcard characters must be quoted.

5.6 Watcher

The '`watcher`' statement configures a single event watcher. A watcher can control several events in multiple pathnames. Any number of `watcher` statements is allowed in the configuration file, each of them declaring a separate watcher.

```
watcher {
    path pathname [recursive [level]];
    file regexp-list;
    event event-list;
    command command-line;
    user name;
    timeout number;
    environ { ... };
    option string-list;
}
```

Statements within a `watcher` block are:

`path` *pathname* [*recursive* [*number*]] [Config]

Defines a pathname to watch. The *pathname* argument must be the name of a directory or file in the file system. If *pathname* refers to a directory, the watcher will watch events occurring for all files within that directory. If the optional `recursive` clause is specified, this directory will be watched recursively, i.e. when any subdirectory is created in it, `direvent` will set up a watcher for files in this subdirectory. This new watcher will be an exact copy of the parent watcher, excepting for the pathnames. The optional *number* parameter defines a cut-off nesting level for recursive watching. If supplied, the recursive behaviour will apply only to the directories that are nested below that level.

If *pathname* refers to a regular file, the changes to that file will be monitored. Obviously, in that case the '`recursive`' keyword makes no sense. If present, it will be silently ignored.

If the *pathname* does not exist, GNU `direvent` will defer setting up the watcher until it is created. In order to do so, it will find the longest directory prefix that exists in the file system and will construct a *sentinel watcher* to monitor creation of the next directory component. When this component is created, the sentinel wakes up to set up a similar watcher for the next directory component. Once it is done, the sentinel removes itself. This process continues until the *pathname* is eventually created. When it happens, the last sentinel will activate the configured watcher.

These actions are performed in reverse order upon removal of *pathname* or any of its trailing directory components.

Any number of `path` statements can appear in a `watcher` block. At least one `path` must be defined.

`file` *regexp-list* [Config]

Selects which files are eligible for monitoring. The argument is a list of globbing patterns (in the sense of see Section "fnmatch" in *fnmatch(3)*) or extended regular expressions (see Section "Extended regular expressions" in *GNU sed*) one of which must match the file name in order for the watcher to act on it. A '`!`' in front of a pattern or regular expression indicates negation. Such construct matches if the file name doesn't match

the pattern. Regular expressions must be surrounded by a pair of slashes, optionally followed by the following flags:

b Use basic regular expressions.

i Enable case-insensitive matching.

For example:

```
file ("*.cfg", "/.*\\.jpg/i");
```

In this statement, the first string (‘`*.cfg`’) is treated as a shell globbing pattern. The second one is a case-sensitive extended regular expression.

`event` *`string-list`* [Config]

Configures the filesystem events to watch for in the directories declared by the `path` statements. The argument is a list of event names. Both generic and system-dependent event names are allowed. Multiple `event` statements accumulate.

A missing `event` statement means “watch all events”.

For example:

```
event (open,delete);
```

`command` *`string`* [Config]

Defines a command to execute on event. The *string* is a command line just as you would type it in `sh`. It may contain macro and environment variables (see Section 5.2 [variable expansion], page 12), which will be expanded prior to execution.

For example:

```
command "/bin/prog -event $genev_name -file $file";
```

By default, the command is executed directly via `execve` system call. If ‘`shell`’ option is set, the command is executed via the shell set in the `SHELL` environment variable.

See [handler environment], page 5, for a detailed discussion of how the command is executed.

`user` *`string`* [Config]

Run command as this user.

`timeout` *`number`* [Config]

Terminate the command if it runs longer than *number* seconds. The default is 5 seconds.

`option` *`string-list`* [Config]

A list of additional options. The following options are defined:

shell Invoke the handler command via shell, as `$SHELL -c "`*`command`*`"`. If this option is set, only macro variables are expanded in *command*. Environment variable references are left to be expanded by the shell. See Section 5.2 [variable expansion], page 12.

wait
: Wait for the program to terminate before handling next event from the event queue. Normally the program runs asynchronously.

stdout
: Capture the standard output of the command and redirect it to the syslog with the '`LOG_INFO`' priority.

stderr
: Capture the standard error of the command and redirect it to the syslog with the '`LOG_ERR`' priority.

`environ` *{ ... }* [Config]

Modify the handler command environment. See Section 5.5 [environ], page 15, for a detailed discussion of configuration statements within the curly braces. This statement applies to the environment, modified by the global `environ` statement, if any.

For compatibility with earlier versions of the program, GNU `direvent`also supports a *legacy syntax* of the `environ` statement. It is described in Appendix A [environ legacy syntax], page 24.

6 System Dependencies

`Direvent` relies on the event monitoring API provided by the kernel.

6.1 GNU/Linux systems.

On GNU/Linux the program uses `inotify`. See Section "monitoring file system events" in *inotify(7) man page*.

The maximum number of watches a user process can have is controlled by the '`fs.inotify.max_user_watches`' system variable. Normally it is set to 8192, which is quite enough for most purposes. However, if you monitor a big number or directories and/or are using recursive watchers, you may need to increase this number. In that case, use `sysctl` (see Section "configure kernel parameters at runtime" in *sysctl(8) man page*) to raise the limit, e.g.:

```
sysctl -w fs.inotify.max_user_watches=16384
```

Most GNU/Linux distributions provide the file `/etc/sysctl.conf` which can be used to set this variable on startup.

The following system-dependent events are defined on systems that use `inotify`:

ACCESS A file was accessed.

ATTRIB A file's metadata changed.

CLOSE_WRITE
: A writable file was closed.

CLOSE_NOWRITE
: An unwritable file closed.

CREATE A file was created.

DELETE A file was deleted.

MODIFY A file was modified.

MOVED_FROM
: A file was moved into a monitored directory.

MOVED_TO
: A file was moved out from a monitored directory.

OPEN A file was opened.

6.2 BSD systems

When compiled on BSD systems (including Darwin), `direvent` uses `kqueue` (see Section "kernel event notification mechanism" in *kqueue(2) man page*).

This interface needs an open file handle for each file in a monitored directory, which means that the number of watchers is limited by the maximum

number of open files. Use '`ulimit -n` *NUM*' in order to raise it to a higher number.

Since it operates on files, `kqueue` does not provide direct support for the '`create`' generic event. `Direvent` works over this disadvantage by keeping track of the contents of each monitored directory and rescanning it each time a '`WRITE`' system event is reported for it. It then generates the '`open`' event for each file that appeared after the last scan. Such a rescan can consume considerable time if a directory has a very large number of files in it.

The following system-dependent events are available:

DELETE The `unlink()` system call was called on the monitored file.

WRITE A write occurred on the file.

EXTEND The file was extended.

ATTRIB The file attributes have changed.

LINK The link count on the file changed.

RENAME The file was renamed.

REVOKE Access to the file was revoked via `revoke()` (see Section "revoke file access" in *revoke(2) man page*) or the underlying file system was unmounted.

On FreeBSD and NetBSD, the following events are additionally available:

CLOSE File not opened for writing was closed.

CLOSE_WRITE
File opened for writing was closed.

OPEN File was opened.

READ File was read.

The `change` generic event (see [generic events], page 2) is supported only on FreeBSD and NetBSD.

6.3 Darwin (Mac OS X)

Essentially the same as BSD. The main difference compared to Linux and BSD is that on Darwin the watchers are set after disconnecting from the controlling terminal, because Darwin lacks the `rfork` call and the event queue cannot be inherited by the child process.

The `change` generic event (see [generic events], page 2) is not supported on Darwin.

7 How to Report a Bug

Please, report bugs and suggestions to bug-direvent@gnu.org.ua.

You hit a bug if at least one of the conditions below is met:

- `direvent` terminates on signal 11 (SIGSEGV) or 6 (SIGABRT).
- The program fails to do its job as described in this manual.

If you think you've found a bug, please be sure to include maximum information available to reliably reproduce it, or at least to analyze it. The information needed is:

- Version of the package you are using.
- Command line options and configuration file.
- Conditions under which the bug appears.

Any errors, typos or omissions found in this manual also qualify as bugs. Please report them, if you happen to find any.

Appendix A Legacy Syntax of the `environ` Statement

This appendix describes the syntax of the `environ` statement used in GNU `direvent`versions 5.2 and earlier. The use of this legacy syntax is discouraged. It is supported for backward compatibility only and will be removed in future versions.

`environ` *`args`* [legacy syntax]
Modify command environment. Arguments are a single environment modification directive, a whitespace-delimited list of directives, or a proper list (see [list], page 12) of directives.

The following directives are available. To facilitate switching to the modern `environ` syntax, the discussion below lists, for each legacy directive, its modern syntax equivalent (see Section 5.5 [environ], page 15).

'`-`' (a single dash)
Clear the inherited environment, but retain the variables added by `direvent` itself. The removed environment variables can be selectively restored using the directives discussed below.

If used, this must be the first directive in the list.

The modern syntax equivalent is:

```
environ {
  clear;
  keep "DIREVENT_*";
}
```

'`--`' (double-dash)
Clear the entire environment, including the variables added by `direvent`.

If used, this must be the first directive in the list.

The modern syntax equivalent is:

```
environ {
  clear;
}
```

-name Unset the variable *name*.

The modern syntax equivalent is

```
environ {
  unset name;
}
```

-name=val
Unset the environment variable *name* only if its value is *val*. The modern syntax equivalent is:

```
environ {
  unset "name=val";
}
```

name Restore the environment variable *name*. This directive is useful after '`-`' or '`--`' to retain some variables from the environment. The modern syntax equivalent is:

```
keep name;
```

name=*value*
Define environment variable *name* to have given *value*. It is equivalent to:

```
environ {
  keep "name=value";
}
```

name+=*value*
Retain variable *name* and append *value* to its existing value. If no such variable is present in the environment, it is created and *value* is assigned to it. However, if *value* begins with a punctuation character, this character is removed from it before the assignment. This is convenient for using this construct with environment variables like `PATH`, e.g.:

```
PATH+=:/sbin
```

In this example, if `PATH` exists, '`:/sbin`' will be appended to it. Otherwise, it will be created and '`/sbin`' will be assigned to it.

In modern syntax, use shell variable references, e.g.:

```
environ {
  set "PATH=${PATH}${PATH:+:}/sbin";
}
```

name=+*value*
Retain variable *name* and prepend *value* to its existing value. If no such variable is present in the environment, it is created and *value* is assigned to it. However, if *value* ends with a punctuation character, this character is removed from it before assignment.

In modern syntax, use shell variable references, e.g. instead of doing

```
environ PATH=+/sbin:
```

use

```
environ {
  set "PATH=/sbin${PATH:+:}$PATH";
}
```

Appendix B GNU Free Documentation License

Version 1.2, November 2002

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or

to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque

copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If

there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

   You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

   The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

   In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

   If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire

aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

B.1 ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright (C) *year your name*.
> Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled ‘‘GNU Free Documentation License’’.

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

> with the Invariant Sections being *list their titles*, with the Front-Cover Texts being *list*, and with the Back-Cover Texts being *list*.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Concept Index

This is a general index of all issues discussed in this manual

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W