Eli System Administration Guide

$Revision: 1.36 $

Compiler Tools Group
Department of Electrical and Computer Engineering
University of Colorado
Boulder, CO, USA
80309-0425

Table of Contents

This manual gathers together information that may be needed by an administrator of an Eli installation.

In some cases more up to date information can be obtained by consulting the information at Eli pages on the World Wide Web. Also, new system releases or patches will be made available via these pages. At present the following Web pages of interest exist:

```
(United States)
       http://www.cs.colorado.edu/~eliuser
(Australia)
       http://www.comp.mq.edu.au/~asloane/eli.html
(Germany)
       http://www.uni-paderborn.de/fachbereich/AG/agkastens/eli_home.html
```

1 Installing Eli

Eli is a collection of *packages* defining processes needed to build a translator from specifications. These processes are managed by an expert system called *Odin*. Odin is designed to manage complex software construction activities of any kind, and is therefore a stand-alone tool that is completely independent of Eli.

The Eli distribution contains a version of Odin that is capable of managing the Eli software construction process. That version of Odin will be built automatically when you run `configure` followed by `make` in the distribution directory.

As described in the distribution's '`README`' file, you can use Eli from the distribution directory without further installation. Because the distribution directory contains source code, however, it is much larger than necessary to simply run Eli (see the end of the distribution's '`README`' file for typical sizes). Therefore we recommend that you run the `make install` command and then delete the distribution directory.

Running `make install` will install both Odin and the Eli packages in the directory you specified by the `--prefix` parameter when you configured the distribution. (Your home directory is used if no `--prefix` parameter was supplied to the `configure` command.)

The installation process will result in the following files and directories being added to the directory in which Eli is installed (the '`man`', '`bin`', and '`lib`' sub-directories will be created by the installation process if they don't already exist):

'`man/man1/odin.1`'
: Man page for Odin.

'`bin/odin`'
: Command file to execute Odin with its basic packages.

'`bin/eli`'
: Command file to execute Odin with the Eli packages added to the basic Odin packages.

'`lib/Eli`'
: The Eli package directory.

'`lib/Odin`'
: The Odin package directory.

These files and directories can be moved to other locations, provided that certain references are updated to reflect the change:

- The Odin man page file and the '`eli`' command file can be moved freely without changing any other file.
- If the '`odin`' command file is moved, the '`eli`' command file must be updated to invoke it from its new location.
- If the '`Eli`' package directory is moved, the '`eli`' command file must be updated to refer to its new location.
- If the '`Odin`' package directory is moved, either the '`odin`' command file must be updated to refer to its new location, or the environment variable `ODINPKGS` must be set to the full path name of the new location.

If you want to install Eli on multiple architectures, you will need a separate build for each. That means running through the configure-make cycle described in the configuration directory's '`README`' file for each architecture. Usually a multi-architecture installation has a distinct file system for multi-architecture software. That file system is managed in such a way that its name is the same on every architecture, but the actual storage referenced is architecture-specific. Your strategy for such a system is simple:

1. Have the system administrator create a directory in the multi-architecture file system that is owned by you. Suppose that directory has the name '`/tools/Eli`'.
2. For each architecture, go to a machine having that architecture and run the following sequence of commands:

   ```
   ./configure --prefix=/tools/Eli
   make
   make install
   make distclean
   ```

3. Put '`/tools/Eli/bin`' in your path.

You will now be able to execute the command `eli` on each architecture.

If you need a multi-architecture implementation, and space is at a premium, you can replace multiple copies of architecture-independent files by symbolic links to a single copy. Typically, about half of an installed Eli system can be shared. This implies, for example, that you can have three shared versions but only consume disk space equivalent to two separate versions.

The '`elilink`' script in the '`Eli`' sub-directory of the distribution can be used to set up this sharing:

`elilink` *working-directory master-directory*

Master-directory is a directory containing an installed version of Eli which is to be used as the complete master copy. It will be unchanged by `elilink`. *working-directory* is *another* installed version of Eli which is to share files with the master directory. That is, files in *working-directory* which are identical to their counterparts in *master-directory* will be linked to the master copy. '`Elilink`' produces messages to let you know what it is doing as it performs the link operations.

'`Elilink`' calls itself recursively. This means that you will either need to put the script in a directory that is on your `PATH` or change the script so that it knows where to look. If you want to do the latter, change the `ELILINK=` line near the top of the script to point to where the script resides.

2 Procedure for reporting a problem with Eli

Should you have a problem with Eli please E-mail a note describing the problem to:

```
elibugs@cs.colorado.edu
```

When reporting problems, please include the Eli version number (printed when Eli starts up, or available in the distribution in '`pkg/version/version.dg`') and the output from running the shell command `uname -a` on your system. Also, please give *complete* output illustrating the problem. If necessary, please include relevant specification files.

In some cases it will also be helpful to identify the version of the software in question. Most distributed files have RCS tags embedded in them. Most executables are generated from sources that have an initialized variable containing version information:

```
char rcsid[] = "$Id: misc.c,v 1.2 88/11/30 12:57:39 bob Exp $";
```

It is easy to identify the source versions from which binaries have been generated by running the RCS `ident` utility on them. For example, running `ident` on '`pkg/gla/gla_fe`' would yield something like the following:

```
gla_fe:
     $Id: driver.c,v 1.15 1996/03/22 19:28:48 kadhim Exp $
     $Id: dfltclp.c,v 1.8 1994/12/08 03:09:52 tony Exp $
     $Id: source.c,v 2.5 1995/01/16 02:19:21 waite Exp $
  ...
```

Sites that do not have `ident` can extract this information by one of the following commands:

Source
: `grep '\$Id:'` '`file-name`'

Executable
: `strings` '`file-name`' `| grep '\$Id:'`

Index