fastdep manual

Bart Vanhauwaert

bvh-cplusplus@irule.be
http://www.irule.be/bvh/

fastdep manual

by Bart Vanhauwaert

Revision History

Revision 1 2002-10-28 Revised by: bvh
Initial version

Table of Contents

List of Examples

Chapter 1. About fastdep and this manual

Fastdep generates dependency information for C of C++ files suitable for inclusion in makefiles. There are two main advantages in using it instead of the normal programs.

- It’s fast due to a unique parse-once technology.
- It has special provision for robust dependency regeneration.

This manual assumes basic knowledge of make and the build process for C or C++ programs.

I am not a native English speaker, nor an experienced documentation writer. Thus it should not come as a surprise to you if this text contains· typographical, grammatical or even simple spelling errors. To overcome this problem I am actively looking for help, suggestions or improvements to this document from my readers. If you are willing to do so, send email to bvh-cplusplus@irule.be (mailto:bvh-cplusplus@irule.be)· to make this guide a great resource for everyone trying to learn fastdep.

Chapter 2. Basic usage

The commandline synopsis for basic usage is

fastdep [--version] *`sourcefile`*...

fastdep --version will show version information and a short copyright statement and exit immediatly.

To produce dependency information for two files named file1.cc and file2.cc, execute **fastdep file1.cc file2.cc** from the shell.

The result will be written to standard output. For example it could be

```
file1.o: file1.cc \
 header1.h \
 header2.h \
 header3.h
file2.o: file2.cc \
 header1.h \
 header3.h \
 header4.h
```

To save the output, use your shell to redirect standard output to a file which you can include in a makefile.

Chapter 3. Writing a make-rule to generate dependency information

In general it is much more convenient to let your makefiles generate their own dependency information instead of doing it by hand as described in the previous chapter.

Let's say we have a variable in our makefile that lists all source files, `SOURCES`. We also want to save dependency information to a file named `.depend`. Here is how to write a make rule to accomplish just that.

Example 3-1. Makefile fragment to generate dependencies from a number of sources

```
.depend:
 fastdep $(SOURCES) > .depend

-include .depend
```

It adds a new target `.depend`, which is file to hold the dependency information generated by fastdep for all files listed in `SOURCES`. The fragment also includes this file in the Makefile itself when it exist. If not make will generate it first (using the rule we specified), restart itself and include it next.

Now each time we want to regenerate dependencies, all we have to do is delete the `.depend` file and launch make.

Chapter 4. Automatically regenerating dependency information

Every time you change a dependency relationship (for example when you include an extra header in some source file), you have to regenerate the dependency information manually by executing **make .depend** (as mentioned in the previous chapter deleting `.depend` also works)

Off course this is very error-prone. Why don't we let make regenerate the dependencies everytime one of the source files changes? Easy enough :

Example 4-1. Makefile fragment to generate dependencies each time a source file changes (wrong).

```
.depend: $(SOURCES)
 fastdep $(SOURCES) > .depend

-include .depend
```

This seems to work fine, but in fact doesn't. Suppose one (or more) of our source files in `SOURCES` includes a certain header `foo.h`. Now imagine that during an edit-compile cycle we don't change anything in the source files, but we added a new include of `bar.h` in `foo.h`. This means that the dependencies for each source file that includes `foo.h` must change. However since none of the source files themself change, make will find that `.depend` is still up to date and not regenerate it. The end result is incorrect dependency information.

To summarize, the dependency information of a source file not only depends on the source file itself, but also on all files it includes. Hence the earlier makefile fragment is incorrect.

To solve this problem, the info manual of GNU make proposes a solution where the output of the dependency generator is modified by piping it through sed. While this works for normal dependency generators like the GNU C/C++ compiler (and still works with fastdep), there is a much more elegant solution in fastdep. By adding the **--remakedeptarget=file** command line option fastdep will also emit a suitable dependency line for its own output.

Example 4-2. Makefile fragment to generate dependencies each time a source or included file changes (right).

```
.depend:
 fastdep --remakedeptarget=.depend $(ALLSOURCES) > .depend

-include dependinfo
```

The `.depend` file will now look like this :

```
file1.o: file1.cc \
 header1.h \
 header2.h \
 header3.h
file2.o: file2.cc \
 header1.h \
 header3.h \
```

```
 header4.h \
.depend: \
 file1.cc \
 header1.h \
 header2.h \
 header3.h \
 file2.cc \
 header4.h \
```

which is exactly how we want it to be.

Chapter 5. Adding an external dependency

Suppose you change your Makefile. For example you add -O2 to the CFLAGS variable, because finally it’s release time. Of course all object files have to be regenerated. The classical way to do that is

```
make clean
make
```

But isn’t it easier to let all .o files depend on the Makefile itself? So that once you touch the makefile, all objects are immediatly out of date and thus regenerated. That’s what the --extraremakedep= option is for.

Example 5-1. Adding an extra dependency to all targets

```
.depend:
 fastdep --extraremakedep=Makefile \
     --remakedeptarget=.depend $(SOURCES) > .depend

-include .depend
```

Here is a possible result

```
file1.o: file1.cc \
 header1.h \
 header2.h \
 header3.h
file2.o: file2.cc \
 header1.h \
 header3.h \
 header4.h \
.depend: \
 file1.cc \
 header1.h \
 header2.h \
 header3.h \
 file2.cc \
 header4.h \
 Makefile
```

Chapter 6. Links and references

fastdep

Homepage (news and download) : http://www.irule.be/bvh/c++/fastdep/

GNU compiler collection (gcc)

Homepage : http://www.gnu.org/software/gcc/gcc.html

Manual : http://www.gnu.org/software/gcc/onlinedocs/

GNU make

Homepage : http://www.gnu.org/software/make/make.html

Manual : http://www.gnu.org/manual/make/html_chapter/make_toc.html