make (software)

From Wikipedia, the free encyclopedia

In software development, **make** is a utility for automatically building executable programs and libraries from source code. Files called **makefiles** specify how to derive the target program from each of its dependencies. **Make** can decide where to start through topological sorting. Though Integrated Development Environments and language-specific compiler features can also be used to manage the build process in modern systems, *make* remains widely used, especially in Unix-based platforms.

make

Original author(s)	Stuart Feldman
Initial release	1977
Type	build automation tool

Contents

- 1 Origin
- 2 Modern versions
- 3 Advantages and disadvantages
- 4 Makefile structure
- 5 Example makefile
- 6 See also
- 7 References
- 8 External links

Origin

There are now a number of dependency-tracking build utilities, but *make* is one of the most widespread, primarily due to its inclusion in Unix, starting with the PWB/UNIX 1.0, which featured a variety of tools targeting software development tasks. It was originally created by Stuart Feldman in 1977 at Bell Labs. In 2003 Dr. Feldman received the ACM Software System Award for the invention of this important tool.[1]

Before make's introduction, the Unix build system would most likely consist of "make" and "install" shell scripts accompanying a program's source. Being able to combine the commands for the different targets into a single file, and being able to abstract out dependency tracking and archive handling, was an important step in the direction of modern build environments.

Modern versions

Make has gone through a number of rewrites, and a number of from-scratch variants which used the same file format and basic algorithmic principles, and also provided a number of their own non-standard enhancements, in the time that followed. Some of them are:

- BSD make, which is derived from Adam de Boor's work on a version of make capable of building targets in parallel, and survives with varying degrees of modification in FreeBSD, NetBSD and OpenBSD. Most notably, it has conditionals and iterative loops which are applied at the parsing stage and may be used to conditionally and programmatically construct the makefile, including generation of targets at runtime.
- GNU make, which is part of Linux distributions and is frequently used in conjunction with the GNU build system. Its departures from traditional make are most noticeable in pattern-matching in dependency graphs and build targets, as well as a number of functions which may be invoked to have the make utility do things like collect a list of all files in the current directory.
- Microsoft *nmake*, commonly available on Windows. It is fairly basic in that it offers only a subset of the features of the two versions of make mentioned above. Microsoft's *nmake* is not to be confused with *nmake* from AT&T and Bell Labs for Unix.

POSIX includes standardization of the basic features and operation of the make utility, and is implemented with varying degrees of completeness in Unix-based versions of make. In general, simple makefiles may be used between various versions of make with reasonable success. Some versions of GNU make and BSD make will look first for files named "GNUmakefile" and "BSDmakefile" respectively, which allows one to put makefiles which use implementation-defined behaviour in separate locations.

Advantages and disadvantages

Make requires the user to describe all dependencies between files, for example which C header files contribute to a specific object file. This keeps Make general and useful with any kind of file, but also opens up a possibility for mistakes. A forgotten or an extra dependency might not be immediately obvious, but instead surface as subtle bugs in the generated software. Such faulty makefiles are common. It is possible to write makefiles that generate these dependencies by calling external tools, but a more common solution is to use one of the available generators to make, e.g. the Automake toolchain provided by the GNU Project.

Another problem not well handled by make is the tailoring of a build process to a given platform. E.g., the compiler used on one platform might not accept the same options as the one used on another. This problem is typically handled by generating platform specific build instructions, which in turn are processed by make. Common tools for this process are Autoconf and CMake.

Make decides whether a target needs to be regenerated by comparing file modification times. While this is a simple solution to the problem of avoiding to build files which are already up to date, it fails when a file changes but its modification time stays in the past. Such changes are frequently caused by the use of revision control software, or when a network filesystem is a source of files -and its clock or timezone is not synchronized with the machine running Make. The user must handle this situation by forcing a complete build. An alternate problem is the source files being in the future. This triggers too much rebuilding, which also inconveniences developers.

The syntax used by Make gives tab, a whitespace character, a different meaning from the space character. This is problematic, since there is usually no visual difference between a tab and a number of space characters. Thus, the syntax of make is often subject to criticism. For programmers using makefile generators or text editors with explicit makefile support, this issue is likely unimportant.

With Integrated Development Environments, especially on non-Unix platforms, many programmers do not manually manage dependency tracking, or even the listing of which files are part of a project. Instead, the task is automated by the integrated environment. Likewise, many modern programming languages have language-specific ways of listing dependencies which are more efficiently tracked through the use of language-specific build utilities. These approaches typically have the drawback that support for arbitrary build instructions is limited.

Make is considered to be a mainly declarative programming language,[2][3][4][5] and these languages are sometimes considered more difficult for programmers used to imperative programming languages.[6]

Makefile structure

A makefile consists of lines of text which define a file (or set of files) or a rule name as depending on a set of files. Output files are marked as depending on their source files, for example, and on files which they include internally, since they all affect the output. After each dependency is listed, a series of lines of tab-indented text may follow which define how to transform the input into the output, if the former has been modified more recently than the latter. In the case where such definitions are present, they are referred to as "build scripts" and are passed to the shell to generate the target file. The basic structure is:[7]

```
# Comments use the hash symbol
target: dependencies
        command 1
        command 2
          .
          .
          .
        command n
```

A makefile also can contain definitions of variables and inclusion of other makefiles. Variables in makefiles may be overridden in the command-line arguments passed to the make utility. This allows users to specify different behaviour for the build scripts and how to invoke programs, among other things. For example, the variable "CC" is frequently used in makefiles to refer to a C compiler, and the user may wish to provide an alternate compiler to use.

Example makefile

Below is a very simple makefile that would compile a source called "helloworld.c" using cc, a C compiler. The PHONY tag is a technicality that tells make that a particular target name does not produce an actual file. The `$@` and `$<` are two of the so-called internal macros and stand for the target name and "implicit" source, respectively. There are a number of other internal macros.[7][8]

```
helloworld: helloworld.o
	cc -o $@ $<

helloworld.o: helloworld.c
	cc -c -o $@ $<

.PHONY: clean

clean:
	rm -f helloworld helloworld.o
```

See also

- Apache Ant
- CMake
- configure script
- Flowtracer
- GNU Automake
- imake
- Jam
- mk
- qmake
- Rake
- SCons
- Debian Package Maker

References

1. ^ Matthew Doar (2005). *Practical Development Environments*. O'Reilly Media. pp. 94. ISBN 978-0596007966.
2. ^ [1] (http://phoenix.labri.fr/wiki/doku.php?id=an_overview_on_dsls)
3. ^ [2] (http://www.cs.ualberta.ca/~paullu/C201/Slides/c201.21-31.pdf)
4. ^ [3] (http://lists.w3.org/Archives/Public/www-ws-arch/2002Aug/0105.html)
5. ^ [4] (http://www.robots.ox.ac.uk/~tgtjr/makefiles.html)
6. ^ [5] (http://hsivonen.iki.fi/honkala-xforms/)
7. ^ ***a b*** Note that make is sensitive to whitespace; the indented lines in the example below would be preceded by tab characters in functional code.
8. ^ http://www.opengroup.org/onlinepubs/009695399/utilities/make.html#tag_04_84_13_07

External links

- The Open Group Base Specifications Issue 6 IEEE Std 1003.1, 2004 Edition manual (http://www.opengroup.org/onlinepubs/009695399/utilities/make.html)
- The GNU make manual (http://www.gnu.org/software/make/manual/html_node/index.html)
- FreeBSD make manual page (http://www.freebsd.org/cgi/man.cgi?query=make&sektion=1)
- OPUS Makefile Tutorial (http://www.opussoftware.com/tutorial/TutMakefile.htm)
- *Recursive Make Considered Harmful* (http://www.pcug.org.au/~millerp/rmch/recu-make-cons-harm.html)
- *What is wrong with make?* (http://freshmeat.net/articles/view/1702/)
- *What's Wrong With GNU make?* (http://www.conifersystems.com/whitepapers/gnu-make/)
- Advanced Auto-Dependency Generation (http://make.paulandlesley.org/autodep.html) .
- GNU make Standard Library (http://gmsl.sourceforge.net/) .

Retrieved from "http://en.wikipedia.org/wiki/Make_(software)"

Categories: GNU project | Compiling tools | Unix software | Build automation

- This page was last modified on 17 June 2009 at 02:54.
- Text is available under the Creative Commons Attribution/Share-Alike License; additional terms may apply. See Terms of Use for details.
  Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a U.S. registered 501(c)(3) tax-deductible nonprofit charity.