GNU Automake

Для версии 1.4, 10 January 1999

David MacKenzie и Tom Tromey

Это первая редакция документации по GNU Automake,
и она согласована с GNU Automake 1.4.

Published by the Free Software Foundation
59 Temple Place - Suite 330,
Boston, MA 02111-1307 USA

1 Введение

Automake — это утилита для автоматического создания файлов ‘`Makefile.in`’ из файлов ‘`Makefile.am`’. Каждый файл ‘`Makefile.am`’ фактически является набором макросов для программы `make` (иногда с несколькими правилами). Полученные таким образом файлы ‘`Makefile.in`’ соответствуют стандартам GNU Makefile.

Стандарт GNU Makefile (см. раздел “Makefile Conventions” в *The GNU Coding Standards*) — это длинный, запутанный документ, и его содержание может в будущем измениться. Automake разработан для того, чтобы убрать бремя сопровождения Makefile с плеч человека, ведущего проект GNU (и взвалить его на человека, сопровождающего Automake).

Типичный входной файл Automake является просто набором макроопределений. Каждый такой файл обрабатывается, и из него создается файл ‘`Makefile.in`’. В каталоге проекта должен быть только один файл ‘`Makefile.am`’.

Automake накладывает на проект некоторые ограничения; например, он предполагает, что проект использует программу Autoconf (см. раздел “Введение” в *Руководство Autoconf*), а также налагает некоторые ограничения на содержимое файла ‘`configure.in`’.

Automake требует наличия программы `perl` для генерации файлов ‘`Makefile.in`’. Однако дистрибутив, созданный Automake, является полностью соответствующим стандартам GNU и не требует наличия `perl` для компиляции.

Вы можете посылать пожелания по доработке и сообщения об ошибках Automake по адресу `bug-automake@gnu.org`.

2 Общая информация

Следующие разделы описывают основные принципы работы Automake.

2.1 Общие операции

Automake читает файл ‘`Makefile.am`’ и создает на его основе файл ‘`Makefile.in`’. Специальные макросы и цели, определенные в ‘`Makefile.am`’, заставляют Automake генерировать более специализированный код; например, макроопределение ‘`bin_PROGRAMS`’ заставит создать цели для компиляции и компоновки программ.

Макроопределения и цели из файла ‘`Makefile.am`’ копируются в файл ‘`Makefile.in`’ без изменений. Это позволяет вам добавлять в генерируемый файл ‘`Makefile.in`’ произвольный код. Например, дистрибутив Automake включает в себя нестандартную цель `cvs-dist`, которую использует человек, сопровождающий Automake, для создания дистрибутивов из системы контроля исходного кода.

Заметьте, что расширения GNU make не распознаются программой Automake. Использование таких расширений в файле ‘`Makefile.am`’ приведет к ошибкам или странному поведению.

Automake пытается сгруппировать комментарии к расположенным по соседству хцелям и макроопределениям.

Цель, определенная в ‘Makefile.am’, обычно переопределяет любую цель с таким же именем, которая была бы автоматически создана automake. Хотя этот прием и работает, старайтесь избегать его использования, поскольку иногда автоматически созданные цели являются очень важными.

Аналогичным образом, макрос, определенный в ‘Makefile.am’, будет переопределять любой макрос, который создает automake. Это часто более полезно, чем возможность переопределения цели. Но будьте осторожны, поскольку многие из макросов, создаваемых программой automake, считаются макросами только для внутреннего использования, и их имена могут измениться в будущих версиях.

При обработке макроопределения Automake рекурсивно обрабатывает макросы, на которые есть ссылка в данном макроопределении. Например, если Automake исследует содержимое foo_SOURCES в следующем определении:

```
xs = a.c b.c
foo_SOURCES = c.c $(xs)
```

то он будет использовать файлы ‘a.c’, ‘b.c’ и ‘c.c’ как содержимое foo_SOURCES.

Automake также вводит форму комментария, который *не* копируется в выходной файл; все строки, начинающиеся с ‘##’, полностью игнорируются Automake.

Очень часто первая строка файла ‘Makefile.am’ выглядит следующим образом:

```
## Process this file with automake to produce Makefile.in
## Для получения Makefile.in обработайте этот файл программой automake
```

2.2 Типы иерархии каталогов пакета

automake поддерживает три типа иерархии каталогов: плоскую, неглубокую и глубокую.

Если все файлы пакета располагаются в одном каталоге, то это *плоский* пакет. В файле ‘Makefile.am’ для этого типа пакета по определению отсутствует макрос SUBDIRS. Примером такого пакета может служить termutils.

Глубокий пакет — это такой, в котором все исходные тексты лежат в подкаталогах; каталог верхнего уровня содержит в основном конфигурационную информацию. Хорошим примером такого пакета является GNU cpio, а так же GNU tar. Файл ‘Makefile.am’ в каталоге верхнего уровня глубокого пакета содержит макрос SUBDIRS, но в нем нет никаких других макросов для определения объектов компиляции.

Неглубокий пакет подразумевает, что основные файлы исходных текстов располагаются в каталоге верхнего уровня, а различные части этого пакета (обычно библиотеки) находятся в подкаталогах. К пакетам такого типа относится Automake (а также GNU make, который в настоящее время не использует automake).

2.3 Ограничения

Хотя Automake предназначен для использования людьми, сопровождающими пакеты GNU, он также старается приспособиться и к тем, кто хочет использовать Automake, но не хочет соблюдать все соглашения GNU.

Сейчас Automake поддерживает три уровня *строгости (strictness)*, задающих, сколь строго Automake должен проверять соответствие стандартам.

Строгость может принимать следующие значения:

‘`foreign`’ Automake проверит только те вещи, которые совершенно необходимы для правильного функционирования. Например, в то время как стандарты GNU требуют наличия файла ‘`NEWS`’, он не требуется при использовании этого режима. Название режима (`"иностранный"`, `"внешний"`) произошло от того факта, что Automake предназначен для использования программ GNU; эти ослабленные требования не являются стандартным режимом функционирования.

‘`gnu`’ Automake проверит, насколько это возможно, соответствие стандартам GNU для пакетов. Этот режим действует по умолчанию.

‘`gnits`’ Automake проверит совместимость с еще не написанными *стандартами Gnits*. Они основан на стандартах GNU, но еще более детальны. Если вы не являетесь помощником в разработке стандартов Gnits, вам лучше избегать этой опции до тех пор, пока стандарт Gnits не будет опубликован.

Для более детальной информации о точном смысле уровня строгости смотрите Глава 19 [Gnits], с. 42.

2.4 Единообразная схема наименования

Макросы Automake (с этого места мы будем ссылаться на них как на *переменные*) в общем следуют *Единообразной Схеме Именования*, которая позволяет легко понять, как собираются программы (и другие результирующие объекты), и как они устанавливаются. Эта схема также поддерживает определение того, что должно быть собрано во время выполнения `configure`.

Во время выполнения `make`, некоторые переменные используются для определения того, что должно быть собрано. Эти переменные называются *основными переменными*. Например, основная переменная `PROGRAMS` содержит список программ, которые должны быть скомпилированы и собраны.

Различные наборы переменных используются для принятия решения о том, куда должны быть установлены собранные объекты. Эти переменные называются подобно основным переменным, но имеют префикс, указывающий, какой из стандартных каталогов должен быть использован в качестве каталогах для установки. Имена стандартных каталогов определены в стандартах GNU (см. раздел “Directory Variables” в *The GNU Coding Standards*). Automake расширяет это список переменными `pkglibdir`, `pkgincludedir` и `pkgdatadir`, которые имеют те же значения, что и не-‘`pkg`’ версии, но с прибавленным к ним суффиксом ‘`@PACKAGE@`’. Например, `pkglibdir` определена как `$(datadir)/@PACKAGE@`.

Для каждой из основных переменных также существует дополнительная переменная, имя которой образовано добавлением префикса ‘`EXTRA_`’ к имени основной переменной. Эта переменная используется для перечисления объектов, которые могут быть собраны, а могут и не собраны в зависимости от принятого `configure` решения. Для того, чтобы создать файл ‘`Makefile.in`’, работающий в любой ситуации, Automake должен сразу узнать полный список объектов, которые вообще могут быть собраны. Исходя из этого, переменные с префиксом ‘`EXTRA_`’ обязательны.

Например, пакет `cpio` во время конфигурации принимает решение о том, какие программы необходимо скомпилировать. Некоторые программы устанавливаются в `bindir`, а некоторые — в `sbindir`:

```
EXTRA_PROGRAMS = mt rmt
bin_PROGRAMS = cpio pax
sbin_PROGRAMS = @PROGRAMS@
```

Определение основной переменной без префикса (например, `PROGRAMS`) является ошибкой.

Заметьте, что общий суффикс '`dir`' опускается при создании имен переменных; таким образом, имя переменной записывается как '`bin_PROGRAMS`', а не '`bindir_PROGRAMS`'.

Нельзя устанавливать любые типы объектов в любые каталоги. Automake будет расценивать такие попытки как ошибку. Automake также будет диагностировать очевидные ошибки в именах каталогов.

Иногда стандартных каталогов— даже с расширениями Automake— недостаточно. В частности, иногда полезно для ясности устанавливать объекты в подкаталог какого-то предопределенного каталога. Здесь Automake также позволяет вам расширить список возможных каталогов для установки. Заданный префикс (например, '`zar`') является разрешенным, если определена переменная, имеющая такое же имя, но с суффиксом '`dir`' (например, `zardir`).

Например, пока поддержка HTML не станет частью Automake, вы можете использовать такой фрагмент кода для установки документации в формате HTML:

```
htmldir = $(prefix)/html
html_DATA = automake.html
```

Специальный префикс '`noinst`' показывает, что указанные объекты вообще не должны быть установлены.

Специальный префикс '`check`' показывает, что указанные объекты не должны быть скомпилированы до тех пор, пока не будет запущена команда `make check`.

Вот список возможных основных имен: '`PROGRAMS`', '`LIBRARIES`', '`LISP`', '`SCRIPTS`', '`DATA`', '`HEADERS`', '`MANS`' и '`TEXINFOS`'.

2.5 Как именуются порожденные переменные

Иногда имена переменных в Makefile образуются на базе некоторого текста, заданного пользователем. Например, имена программ преобразуются в имена макросов Makefile. Automake канонизирует этот текст, так что он не должен следовать правилам именования макросов Makefile. Все имена в имени, за исключением букв, чисел, и подчеркиваний, при создании ссылки на макрос преобразуются в подчеркивания . Например, если ваша программа называется `sniff-glue`, то унаследованная переменная будет называться `sniff_glue_SOURCES`, а не `sniff-glue_SOURCES`.

3 Некоторые примеры пакетов

3.1 Простой пример, от начала до конца

Давайте предположим, что мы только что закончили писать `zardoz` — программу, от которой у всех кружится голова. Вы использовали Autoconf для обеспечения переносимости, но ваш файл '`Makefile.in`' был написан бессистемно. Вы же хотите сделать его пуленепробиваемым, и поэтому решаете использовать Automake.

Сначала вам необходимо обновить ваш файл '`configure.in`', чтобы вставить в него команды, которые необходимы для работы `automake`. Проще всего для этого добавить строку `AM_INIT_AUTOMAKE` сразу после `AC_INIT`:

```
AM_INIT_AUTOMAKE(zardoz, 1.0)
```

Поскольку ваша программа не имеет никаких осложняющих факторов (например, она не использует `gettext` и не будет создавать разделяемые библиотеки), то первая стадия на этом и заканчивается. Это легко!

Теперь вы должны заново создать файл '`configure`'. Но для этого нужно сказать `autoconf`, где найти новые макросы, которые вы использовали. Для создания файла '`aclocal.m4`' удобнее всего будет использовать программу `aclocal`. Но будьте осторожны... у вас уже есть '`aclocal.m4`', поскольку вы уже написали несколько собственных макросов для вашей программы. Программа `aclocal` позволяет вам поместить ваши собственные макросы в файл '`acinclude.m4`', так что для сохранения вашей работы просто переименуйте свой файл с макросами, а уж затем запускайте программу `aclocal`:

```
mv aclocal.m4 acinclude.m4
aclocal
autoconf
```

Теперь пришло время написать свой собственный файл '`Makefile.am`' для программы `zardoz`. Поскольку `zardoz` является пользовательской программой, то вам хочется установить ее туда, где располагаются другие пользовательские программы. Вдобавок, `zardoz` содержит в комплекте документацию в формате Texinfo. Ваш скрипт '`configure.in`' использует `AC_REPLACE_FUNCS`, так что вам необходимо скомпоновать программу с '`@LIBOBJS@`'. Вот что вам необходимо написать в '`Makefile.am`'.

```
bin_PROGRAMS = zardoz
zardoz_SOURCES = main.c head.c float.c vortex9.c gun.c
zardoz_LDADD = @LIBOBJS@

info_TEXINFOS = zardoz.texi
```

Теперь можно запустить `automake --add-missing`, чтобы создать файл '`Makefile.in`', используя дополнительные файлы, и вот, все готово!

3.2 Классическая программа

GNU hello известен своей классической простотой и многогранностью. В этом разделе показывается, как Automake может быть использован с пакетом GNU Hello.

Примеры, приведенные ниже, взяты из последней бета-версии GNU Hello, но убран код, предназначенный только для разработчика пакет, а также сообщения об авторских правах.

Конечно же, GNU Hello использует больше возможностей, чем традиционная двухстроковая программа: GNU Hello работает с разными языками, выполняет обработку ключей командной строки, имеет документацию и набор тестов. GNU Hello является глубоким пакетом.

Вот файл ‘configure.in’ из пакета GNU Hello:

```
dnl Обработайте этот файл программой autoconf для создания скрипта configure.
AC_INIT(src/hello.c)
AM_INIT_AUTOMAKE(hello, 1.3.11)
AM_CONFIG_HEADER(config.h)

dnl Набор доступных языков.
ALL_LINGUAS="de fr es ko nl no pl pt sl sv"

dnl Проверка наличия программ.
AC_PROG_CC
AC_ISC_POSIX

dnl Проверка имеющихся библиотек.

dnl Проверка наличия заголовочных файлов.
AC_STDC_HEADERS
AC_HAVE_HEADERS(string.h fcntl.h sys/file.h sys/param.h)

dnl Проверка библиотечных функций.
AC_FUNC_ALLOCA

dnl Проверка наличия поля st_blksize в структуре stat
AC_ST_BLKSIZE

dnl Макросы поддержки различных языков
AM_GNU_GETTEXT
AC_OUTPUT([Makefile doc/Makefile intl/Makefile po/Makefile.in \
           src/Makefile tests/Makefile tests/hello],
   [chmod +x tests/hello])
```

Макросы ‘AM_’ предоставляются Automake (или библиотекой Gettext); остальные макросы является макросами Autoconf.

Файл ‘Makefile.am’ в корневом каталоге выглядит следующим образом:

```
EXTRA_DIST = BUGS ChangeLog.0
SUBDIRS = doc intl po src tests
```

Как видите, вся работа выполняется в подкаталогах.

Каталоги ‘po’ и ‘intl’ автоматически создаются программой gettextize; они не будут обсуждаться в этом документе.

В файле ‘doc/Makefile.am’ мы видим строки:

```
info_TEXINFOS = hello.texi
hello_TEXINFOS = gpl.texi
```

Этого достаточно для сборки, установки и распространения руководства GNU Hello.

Вот содержимое файла ‘tests/Makefile.am’:

```
TESTS = hello
EXTRA_DIST = hello.in testdata
```

Скрипт ‘hello’ создается configure, и является единственным тестовым случаем. При выполнении make check будет запущен именно этот тест.

В заключение мы приведем содержимое ‘src/Makefile.am’, где и выполняется вся настоящая работа:

```
bin_PROGRAMS = hello
hello_SOURCES = hello.c version.c getopt.c getopt1.c getopt.h system.h
hello_LDADD = @INTLLIBS@ @ALLOCA@
localedir = $(datadir)/locale
INCLUDES = -I../intl -DLOCALEDIR=\"$(localedir)\"
```

3.3 Компиляция программ etags и ctags

Вот другой, более изощренный пример. Он показывает, как собрать две программы (ctags и etags) из одного и того же исходного файла (‘etags.c’). Самая трудное в том, что каждая компиляция файла ‘etags.c’ требует задания разных флагов для cpp.

```
bin_PROGRAMS = etags ctags
ctags_SOURCES =
ctags_LDADD = ctags.o

etags.o: etags.c
        $(COMPILE) -DETAGS_REGEXPS -c etags.c

ctags.o: etags.c
        $(COMPILE) -DCTAGS -o ctags.o -c etags.c
```

Заметьте, что переменная ctags_SOURCES определена как пустая — при этому не подставляется неявного значения по умолчанию. Для создания etags из файла ‘etags.o’, однако, используются неявные значения.

Переменная ctags_LDADD используется для вставки ‘ctags.o’ в строку компоновщика. ctags_DEPENDENCIES создается Automake.

Вышеприведенные правила не работают в том случае, если ваш компилятор не умеет одновременно работать с ключами ‘-c’ и ‘-o’. Самым простым способом исправить это недоразумение является введение поддельной зависимости (для того, чтобы избежать проблем с параллельной версией make):

```
etags.o: etags.c ctags.o
        $(COMPILE) -DETAGS_REGEXPS -c etags.c

ctags.o: etags.c
        $(COMPILE) -DCTAGS -c etags.c && mv etags.o ctags.o
```

Эти явные правила также не работают, если используется де-ANSI-фикация (см. Раздел 7.10 [ANSI], с. 29). Поддержка де-ANSI-фикации требует немного больше работы:

```
etags._o: etags._c ctags.o
        $(COMPILE) -DETAGS_REGEXPS -c etags.c

ctags._o: etags._c
        $(COMPILE) -DCTAGS -c etags.c && mv etags._o ctags.o
```

4 Создание файла ‘Makefile.in’

Для создания всех файлов ‘Makefile.in’ пакета запустите программу automake в каталоге верхнего уровня без аргументов. automake автоматически найдет каждый файл ‘Makefile.am’ (сканируя ‘configure.in’; см. Глава 5 [configure], с. 10) и сгенерирует соответствующий файл ‘Makefile.in’. Заметьте, что automake имеет более простое видение структуры пакета; он предполагает, что пакет имеет только один файл ‘configure.in’, расположенный в каталоге верхнего уровня. Если в вашем пакете имеется несколько файлов ‘configure.in’, то вам необходимо запустить automake в каждом из каталогов, где есть файл ‘configure.in’.

Также вы можете задать аргумент для automake; суффикс ‘.am’ добавляется к аргументу и результат используется как имя входного файла. В основном эта возможность используется для автоматической перегенерации устаревших файлов ‘Makefile.in’. Заметьте, что automake всегда должен запускаться из каталога верхнего уровня проекта, даже если необходимо перегенерировать ‘Makefile.in’ в каком-то из подкаталогов. Это необходимо, так как automake должен просканировать файл ‘configure.in’, а также потому, что automake в некоторых случаях изменяет свое поведение при обработке ‘Makefile.in’ в подкаталогах.

automake принимает следующие ключи командной строки:

‘-a’
‘-add-missing’
: В некоторых ситуациях Automake требует наличия некоторых общих файлов; например, если в ‘configure.in’ выполняется макрос AC_CANONICAL_HOST, то требуется наличие файла ‘config.guess’. Automake распространяется с несколькими такими файлами; этот ключ заставит программу автоматически добавить к пакету отсутствующие файлы, если это возможно. В общем, если Automake сообщает вам, что какой-то файл отсутствует, то используйте этот ключ. По умолчанию Automake пытается создать символьную ссылку на собственную копию отсутствующего файла; это поведение может быть изменено с помощью ключа --copy.

‘-amdir=*dir*’
: Этот ключ заставляет Automake искать файлы данных в каталоге *dir*, а не в каталоге установки. Этот ключ обычно используется при отладке.

‘-build-dir=*dir*’
: Сообщает Automake, где располагается каталог для сборки. Этот ключ используется при включении зависимостей в файл ‘Makefile.in’, созданный командой `make dist`; он не должен использоваться в других случаях.

‘-c’

‘-copy’
: При использовании с ключом `--add-missing`, заставляет копировать недостающие файлы. По умолчанию создаются символьные ссылки.

‘-cygnus’
: Заставляет сгенерированные файлы ‘Makefile.in’ следовать правилам Cygnus, вместо правил GNU или Gnits. Для дополнительной информации, смотрите Глава 20 [Cygnus], с. 42.

‘-foreign’
: Устанавливает глобальную строгость в значение ‘foreign’. За дополнительной информацией смотрите раздел Раздел 2.3 [Строгость], с. 2.

‘-gnits’
: Устанавливает глобальную строгость в значение ‘gnits’. За дополнительной информацией смотрите раздел Глава 19 [Gnits], с. 42.

‘-gnu’
: Устанавливает глобальную строгость в значение ‘gnu’. За дополнительной информацией смотрите раздел Глава 19 [Gnits], с. 42. По умолчанию используется именно такая строгость.

‘-help’
: Печатает список ключей командной строки и завершается.

‘-i’

‘-include-deps’
: Включить всю автоматически генерируемую информацию о зависимостях (см. Раздел 7.11 [Зависимости], с. 30) в генерируемый файл ‘Makefile.in’. Это делается в основном при создании дистрибутива; смотрите раздел Глава 13 [Дистрибутив], с. 36.

‘-generate-deps’
: Создать файл, объединяющий всю автоматически генерируемую информацию о зависимостях (см. Раздел 7.11 [Зависимости], с. 30), этот файл будет называться ‘.dep_segment’. В основном этот ключ используется при создании дистрибутива; смотрите Глава 13 [Дистрибутив], с. 36. Он полезен при сопровождении ‘SMakefile’ или файлов ‘Makefile’ для других платформ (‘Makefile.DOS’, и т. п.). Этот ключ может использоваться только с ключами ‘-include-deps’, ‘-srcdir-name’ и ‘-build-dir’. Заметьте, что если задан этот ключ, то никакой другой обработки не выполняется.

‘-no-force’
: Обычно `automake` создает все файлы ‘Makefile.in’, указанные в ‘configure.in’. Этот ключ заставляет обновлять только те файлы ‘Makefile.in’, которые устарели, с учетом зависимостей друг от друга.

‘-o *dir*’

‘-output-dir=*dir*’
: Поместить сгенерированный файл ‘Makefile.in’ в каталог *dir*. Обычно каждый файл ‘Makefile.in’ создается в том же каталоге, что и соответ-

ствующий файл ‘Makefile.am’. Этот ключ используется при создании дистрибутивов.

‘-srcdir-name=*dir*’
: Сообщает Automake имя каталога с исходными текстами текущего дистрибутива. Этот ключ используется при включении зависимостей в файл ‘Makefile.in’, сгенерированный командой make dist; он не должен использоваться в других случаях.

‘-v’
‘-verbose’
: Заставляет Automake выдавать информацию о том, какие файлы читаются или создаются.

‘-version’
: Выдает номер версии Automake и завершается.

5 Сканирование файла ‘configure.in’

Для определения разной информации о данном пакете Automake сканирует файл ‘configure.in’. Также ему требуется определение некоторых макросов и переменных autoconf в файле ‘configure.in’. Automake также использует информацию из файла ‘configure.in’ для определения параметров вывода.

Для того, чтобы облегчить сопровождение, Automake предоставляет некоторые макросы Autoconf. Эти макросы могут быть автоматически помещены в ваш файл ‘aclocal.m4’ при использовании программы aclocal.

5.1 Требования к конфигурации

Чтобы удовлетворить основным требованиям Automake, можно использовать макрос AM_INIT_AUTOMAKE (см. Раздел 5.4 [Макросы], с. 15). Но если хотите, то можете совершить требуемые шаги вручную:

- Определить переменные PACKAGE и VERSION с помощью AC_SUBST. Переменная PACKAGE должна содержать имя пакета, в том виде, в котором оно используется при создании дистрибутива. Например, Automake определяет переменную PACKAGE со значением ‘automake’. Переменная VERSION должна содержать номер разрабатываемой версии. Мы рекомендуем хранить номер версии в единственном месте, а именно, в файле ‘configure.in’: это упрощает выпуск новых версий.

  Automake не производит никакой интерпретации переменных PACKAGE или VERSION, за исключением работы в режиме ‘Gnits’ (см. Глава 19 [Gnits], с. 42).
- Если программа или скрипт устанавливаются, то используйте макрос AC_ARG_PROGRAM. См. раздел “Преобразование имен при установке” в *Руководство Autoconf*.
- Используйте макрос AC_PROG_MAKE_SET если пакет не является плоским. См. раздел “Создание файлов вывода” в *Руководство Autoconf*.
- Используйте макрос AM_SANITY_CHECK для того, чтобы убедиться, что среда, в которой будет производится сборка пакета, является нормальной.

- Вызовите макрос AC_PROG_INSTALL (см. раздел “Проверка отдельных программ” в *Руководство Autoconf*).
- Используйте макрос AM_MISSING_PROG для того, чтобы убедиться, что программы aclocal, autoconf, automake, autoheader и makeinfo находятся в среде в которой производится сборка пакета. Вот как это сделано:

```
missing_dir=‘cd $ac_aux_dir && pwd‘
AM_MISSING_PROG(ACLOCAL, aclocal, $missing_dir)
AM_MISSING_PROG(AUTOCONF, autoconf-ru, $missing_dir)
AM_MISSING_PROG(AUTOMAKE, automake, $missing_dir)
AM_MISSING_PROG(AUTOHEADER, autoheader, $missing_dir)
AM_MISSING_PROG(MAKEINFO, makeinfo, $missing_dir)
```

Вот список других макросов, которые требуются Automake, но которые не запускаются макросом AM_INIT_AUTOMAKE:

AC_OUTPUT
: Automake использует этот макрос для определения того, какие файлы необходимо создавать (см. раздел “Создание выходных файлов” в *Руководство Autoconf*). Файлы с именем ‘Makefile’ из этого списка считаются файлами для программы make. Остальные файлы интерпретируются по разному. В настоящее время отличие состоит лишь в том, что файлы ‘Makefile’ удаляются make distclean, тогда как другие файлы удаляются командой make clean.

5.2 Другие вещи, которые распознает Automake

Также Automake распознает использование некоторых макросов и в соответствии с ними генерирует ‘Makefile.in’. Вот список распознаваемых макросов и результатов их работы:

AC_CONFIG_HEADER
: Automake требует использования макроса AM_CONFIG_HEADER, который похож на AC_CONFIG_HEADER (см. раздел “Заголовочные файлы настройки” в *Руководство Autoconf*), но кроме этого выполняет полезную работу, специфичную для Automake.

AC_CONFIG_AUX_DIR
: Automake будет искать различные вспомогательные скрипты, такие как ‘mkinstalldirs’, в каталоге, указанном в качестве параметра макроса. Если скрипты там не обнаружены, то они ищутся в их *стандартном* месте (в каталоге верхнего уровня пакета, либо в каталоге исходных текстов, соответствующем текущему файлу ‘Makefile.am’). См. раздел “Нахождение ввода ‘configure’” в *Руководство Autoconf*.

AC_PATH_XTRA
: Automake при выполнении этого макроса для каждого файла ‘Makefile.in’, который компилирует программу или библиотеку на C, поместит туда определения переменных, указанных в AC_PATH_XTRA. См. раздел “Системные сервисы” в *Руководство Autoconf*.

`AC_CANONICAL_HOST`
`AC_CHECK_TOOL`
: Automake обеспечит существование файлов ‘config.guess’ и ‘config.sub’. Также в файле ‘Makefile’ появятся переменные ‘host_alias’ и ‘host_triplet’. Смотрите раздел “Получение канонического типа системы” в *Руководство Autoconf*, и раздел “Проверка базовых программ” в *Руководство Autoconf*.

`AC_CANONICAL_SYSTEM`
: Этот макрос подобен макросу `AC_CANONICAL_HOST`, но кроме этого он определяет в файле ‘Makefile’ переменные ‘build_alias’ и ‘target_alias’. См. раздел “Получение канонического типа системы” в *Руководство Autoconf*.

`AC_FUNC_ALLOCA`
`AC_FUNC_GETLOADAVG`
`AC_FUNC_MEMCMP`
`AC_STRUCT_ST_BLOCKS`
`AC_FUNC_FNMATCH`
`AM_FUNC_STRTOD`
`AC_REPLACE_FUNCS`
`AC_REPLACE_GNU_GETOPT`
`AM_WITH_REGEX`
: Automake обеспечит генерацию соответствующих зависимостей для объектов, относящихся к этим макросам. Также Automake проверит, что соответствующие файлы исходных текстов являются частью дистрибутива. Заметьте, что Automake поставляется без исходных текстов на C, которые требуются для использования этих макросов, так что `automake -a` не сможет установить их. За дополнительной информацией см. См. Раздел 7.2 [Библиотека], с. 20. Также смотри раздел “Проверка отдельных функций” в *Руководство Autoconf*.

`LIBOBJS` Automake также обнаружит операторы, которые помещают файлы с расширением ‘.o’ в `LIBOBJS`, и будет обрабатывать эти дополнительные файлы так, как если бы они описывались макросом `AC_REPLACE_FUNCS`. См. раздел “Проверка базовых функций” в *Руководство Autoconf*.

`AC_PROG_RANLIB`
: Этот макрос требуется, если в пакете собирается какая-нибудь библиотека. См. раздел “Проверка отдельных программ” в *Руководство Autoconf*.

`AC_PROG_CXX`
: Требуется если в пакет входят исходные тексты на языке C++. См. раздел “Проверка отдельных программ” в *Руководство Autoconf*.

`AC_PROG_F77`
: Требуется, если в пакет будут включаться исходные тексты на Fortran 77. Этот макрос распространяется с Autoconf версии 2.13 и более поздних. См. раздел “Проверка отдельных программ” в *Руководство Autoconf*.

`AC_F77_LIBRARY_LDFLAGS`
: Этот макрос требуется для программ и разделяемых библиотек, которые написаны на разных языках и включают Fortran 77 (см. Раздел 7.8.3 [Использование Fortran 77 с C и C++], с. 26). См. Раздел 5.4 [Макросы Autoconf поставляемые с Automake], с. 15.

`AM_PROG_LIBTOOL`
: Automake включит поддержку `libtool` (см. раздел “Введение” в *Руководство Libtool*).

`AC_PROG_YACC`
: Если в пакете есть исходный текст на Yacc, то вы должны либо использовать этот макрос, либо определить переменную ‘`YACC`’ в файле ‘`configure.in`’. Рекомендуется использовать первый вариант (См. раздел “Проверка отдельных программ” в *Руководство Autoconf*.)

`AC_DECL_YYTEXT`
: Этот макрос требуется, если в пакете есть исходный текст на Lex. См. раздел “Проверка отдельных программ” в *Руководство Autoconf*.

`AC_PROG_LEX`
: Если есть исходный текст на Lex, то должен использоваться этот макрос. См. раздел “Проверка отдельных программ” в *Руководство Autoconf*.

`ALL_LINGUAS`
: Если Automake обнаружит, что эта переменная установлена в файле ‘`configure.in`’, то он проверит каталог ‘`po`’, для того, чтобы обеспечить, что все указанные файлы с расширением ‘`.po`’ существуют, и что указаны все существующие файлы ‘`.po`’.

`AM_C_PROTOTYPES`
: Это макрос требуется при использовании автоматической де-ANSI-фикации; смотри Раздел 7.10 [ANSI], с. 29.

`AM_GNU_GETTEXT`
: Этот макрос требуется для пакетов, которые используют пакет GNU gettext (см. Раздел 9.2 [gettext], с. 32). Он распространяется вместе с gettext. Если Automake находит этот макрос, то он проверяет, отвечает ли данный пакет некоторым требованиям gettext.

`AM_MAINTAINER_MODE`
: Этот макрос добавляет ключ ‘`-enable-maintainer-mode`’ к скрипту `configure`. Если используется данный макрос, то `automake` отключит правило ‘`maintainer-only`’ в сгенерированных файлах ‘`Makefile.in`’. Этот макрос не разрешен в режиме ‘`Gnits`’ (см. Глава 19 [Gnits], с. 42). Этот макрос определяет условную переменную ‘`MAINTAINER_MODE`’, которую можно использовать в ваших собственных файлах ‘`Makefile.am`’.

AC_SUBST
AC_CHECK_TOOL
AC_CHECK_PROG
AC_CHECK_PROGS
AC_PATH_PROG
AC_PATH_PROGS

Для каждого из этих макросов, их первый аргумент автоматически определяется в качестве переменной в каждом сгенерированном файле ‘Makefile.in’. См. раздел “Установка переменных вывода” в *Руководство Autoconf*, и раздел “Проверка основных переменных” в *Руководство Autoconf*.

5.3 Автоматическая генерация ‘aclocal.m4’

Automake содержит некоторое количество макросов Autoconf, которые могут быть использованы в вашем пакете; в некоторых ситуациях они требуются для работы Automake. Эти макросы должны быть определены в вашем файле ‘aclocal.m4’; иначе они не будут обнаружены программой autoconf.

Программа aclocal автоматически создает файл ‘aclocal.m4’ на основе содержимого ‘configure.in’. Это обеспечивает удобный способ для получения макросов Automake, без выполнения дополнительного поиска. Механизм aclocal является также расширяемым для использования другими пакетами.

При запуске программа aclocal производит поиск макроопределений во всех файлах ‘.m4’, которые она может найти. Затем она сканирует ‘configure.in’. Любое упоминание одного из найденных на первом этапе макросов приводит к тому, что этот макрос и все макросы, требуемые для его работы, будут помещены в файл ‘aclocal.m4’.

Если файл ‘acinclude.m4’ существует, то его содержимое также будет автоматически включено в ‘aclocal.m4’. Это полезно для включения локальных макросов в ‘configure’.

Программа aclocal работает со следующими ключами командной строки:

-acdir=*dir*

Заставляет программу искать файлы с макросами в каталоге *dir*, вместо каталога, куда производилась установка программы. Этот ключ в основном используется для отладки.

-help Напечатать справку по ключам командной строки и закончить работу.

-I *dir* Добавляет каталог *dir* в список каталогов, в которых производится поиск файлов ‘.m4’.

-output=*file*

Вывод производится в файл *file*, а не в файл ‘aclocal.m4’.

-print-ac-dir

Печатает имя каталога, в котором aclocal будет производить поиск файлов ‘.m4’. При задании этого ключа подавляется обычная обработка. Этот ключ используется пакетом для определения места, куда будет производиться установка файлов с макросами.

`-verbose` Печатает имена обрабатываемых файлов.

`-version` Выдает номер версии и заканчивает работу.

5.4 Макросы Autoconf, поставляемые с Automake

`AM_CONFIG_HEADER`

При использовании этого макроса Automake сгенерирует правила для автоматической регенерации заголовочного файла конфигурации. Если вы используете этот макрос, то вы должны создать в каталоге исходных текстов файл ‘`stamp-h.in`’. Он может быть пустым.

`AM_ENABLE_MULTILIB`

Этот макрос используется, когда будет строиться “мульти-библиотека”. “Мульти-библиотека” компилируется несколько раз, по разу на каждую комбинацию флагов компиляции. Это полезно только в тех случаях, когда библиотека предназначена для кросс-компиляции. Первым необязательным аргументом макроса является имя создаваемого файла ‘`Makefile`’; значением по умолчанию является ‘`Makefile`’. Второй аргумент используется для нахождения каталога верхнего уровня исходных текстов; по умолчанию используется пустая строка (обычно этот аргумент не следует использовать, если вы не знакомы с внутренним устройством).

`AM_FUNC_STRTOD`

Если функция `strtod` недоступна, или работает неправильно (как в SunOS 5.4), то строка ‘`strtod.o`’ добавляется к выходной переменной `LIBOBJS`.

`AM_FUNC_ERROR_AT_LINE`

Если функция `error_at_line` не найдена, то строка ‘`error.o`’ добавляется к `LIBOBJS`.

`AM_FUNC_MKTIME`

Проверяет наличие работоспособной функции `mktime`. Если таковая не найдена, то к переменной ‘`LIBOBJS`’ добавляется ‘`mktime.o`’.

`AM_FUNC_OBSTACK`

Проверка наличия кода GNU obstacks; если код не найден, то добавить строку ‘`obstack.o`’ к переменной ‘`LIBOBJS`’.

`AM_C_PROTOTYPES`

Проверяет, распознает ли компилятор прототипы функций. Если это происходит, то определяет переменную ‘`PROTOTYPES`’ и устанавливает выходные переменные ‘`U`’ и ‘`ANSI2KNR`’ в пустую строку. В противном случае, устанавливает ‘`U`’ равным ‘`_`’, а ‘`ANSI2KNR`’ в ‘`./ansi2knr`’. Automake использует эти значения для реализации автоматической де-ANSI-фикации.

`AM_HEADER_TIOCGWINSZ_NEEDS_SYS_IOCTL`

Если использование `TIOCGWINSZ` требует наличия файла ‘`<sys/ioctl.h>`’, то этот макрос определяет переменную `GWINSZ_IN_SYS_IOCTL`. В противном случае поиск `TIOCGWINSZ` будет осуществляться в ‘`<termios.h>`’.

`AM_INIT_AUTOMAKE`
: Запускает множество макросов, в которых нуждается ‘configure.in’. Этот макрос требует два аргумента — имя пакета и номер версии. По умолчанию этот макрос определяет через `AC_DEFINE` макросы ‘PACKAGE’ и ‘VERSION’. Такого поведения можно избежать, передавая непустой третий аргумент.

`AM_PATH_LISPDIR`
: Ищет программу `emacs`, и если она найдена, то устанавливает выходную переменную `lispdir` равной полному пути к каталогу ‘site-lisp’ программы Emacs.

`AM_PROG_CC_STDC`
: Если по умолчанию компилятор C не работает в режиме ANSI C, то пробует добавить опцию к переменно `CC`, которая заставит его делать это. Этот макрос пробует различные ключи командной строки компилятора, которые включают режим ANSI C на некоторых системах. Считается, что компилятор находится в режиме ANSI C, если он корректно обрабатывает прототипы функций.

  Если вы используете этот макрос, то вы должны проверить, что после его вызова компилятор C будет работать в режиме ANSI C; если это не так, то переменная среды `am_cv_prog_cc_stdc` устанавливается в значение ‘no’. Если вы написали свою программу в стандарте ANSI C, то вы можете создать ее не-ANSI-фицированную копию, используя опцию `ansi2knr` (см. Раздел 7.10 [ANSI], с. 29).

`AM_PROG_LEX`
: Этот макрос похож на макросы `AC_PROG_LEX` и `AC_DECL_YYTEXT` (см. раздел “Проверка отдельных программ” в *Руководство Autoconf*), но использует скрипт `missing` на системах, в которых нет `lex`. Одной из таких систем является ‘HP-UX 10’.

`AM_SANITY_CHECK`
: Этот макрос выполняет проверку того, что файл, созданный в каталоге для компиляции, новее, чем файл в каталоге с исходными текстами. На системах с неправильно установленными часами произойдет сбой. Этот макрос автоматически запускается из `AM_INIT_AUTOMAKE`.

`AM_SYS_POSIX_TERMIOS`
: Проверяет, доступны ли заголовочные файлы POSIX ‘termios’ в данной системе. Если это так, то переменная среды `am_cv_sys_posix_termios` устанавливается в значение ‘yes’. Если нет, то значением переменной будет являться ‘no’.

`AM_TYPE_PTRDIFF_T`
: Определяет переменную ‘HAVE_PTRDIFF_T’ в том случае, если тип ‘ptrdiff_t’ определен в ‘<stddef.h>’.

AM_WITH_DMALLOC
: Добавляет поддержку пакета dmalloc Если пользователь выполняет конфигурацию с ключом ‘-with-dmalloc’, то будет определена переменная WITH_DMALLOC и добавлен ключ ‘-ldmalloc’ в переменную LIBS.

AM_WITH_REGEX
: Добавляет ‘-with-regex’ к ключам командной строки configure. Если этот ключ указан (по умолчанию), то используется библиотека регулярных выражений ‘regex’, файл ‘regex.o’ помещается в ‘LIBOBJS’ и определяется переменная ‘WITH_REGEX’. Если задан ключ ‘-without-regex’, то используется библиотека регулярных выражений ‘rx’, а ‘rx.o’ добавляется в переменную ‘LIBOBJS’.

5.5 Написание ваших собственных макросов aclocal

Программа aclocal сама по себе ничего не знает о каких-либо макросах, поэтому ее очень легко расширять, создавая свои собственные макросы.

Эта возможность в основном используется библиотеками, которые хотят предоставить собственные макросы Autoconf для использования другими программами. Например, библиотека gettext предоставляет макрос AM_GNU_GETTEXT, который должен быть использован любым пакетом, использующим gettext. При установке библиотеки устанавливается также этот макрос, чтобы программа aclocal смогла его найти.

Файл макросов должен быть серией вызовов AC_DEFUN. Программа aclocal также понимает директиву AC_REQUIRE, так что вполне безопасно помещать каждый макрос в отдельный файл. См. раздел “Необходимые макросы” в *Руководство Autoconf*, и раздел “Макроопределения” в *Руководство Autoconf*.

Имя файла макросов должно оканчиваться на ‘.m4’. Такие файлы должны устанавливаться в каталог ‘$(datadir)/aclocal’.

6 ‘Makefile.am’ верхнего уровня

В неплоских пакетах в файле ‘Makefile.am’ верхнего уровня надо указать Automake, в каких подкаталогах будет производится сборка. Это выполняется с помощью переменной SUBDIRS.

Макрос SUBDIRS содержит список подкаталогов, в которых могут производиться различные виды сборки. Многие цели (например, all) в сгенерированном файле ‘Makefile’ будут выполняться как в текущем каталоге, так и во всех указанных подкаталогах. Заметьте, что подкаталоги, перечисленные в SUBDIRS, не обязаны содержать файл ‘Makefile.am’, а только лишь ‘Makefile’ (после выполнения конфигурации). Это позволяет использовать библиотеки из пакетов, которые не используют Automake (например, gettext). Каталоги, упомянутые в SUBDIRS, должны быть прямыми потомками текущего каталога. Например, вы не можете поместить каталог ‘src/subdir’ в переменную SUBDIRS.

В глубоких пакетах ‘Makefile.am’ верхнего уровня часто очень короток. Например, вот ‘Makefile.am’ из дистрибутива GNU Hello:

```
EXTRA_DIST = BUGS ChangeLog.0 README-alpha
SUBDIRS = doc intl po src tests
```

Можно переопределить переменную SUBDIRS если, как в случае GNU Inetutils, вы хотите собрать только некоторое подмножество пакета. Для этого включите в ваш файл 'Makefile.am' следующие строки:

```
SUBDIRS = @SUBDIRS@
```

Затем в вашем файле 'configure.in' вы можете указать:

```
SUBDIRS = "src doc lib po"
AC_SUBST(SUBDIRS)
```

В результате этого Automake сможет при построении пакета заставить его принимать список каталогов, но точное содержимое этого списка станет известно только после запуска configure.

Хотя макрос SUBDIRS может содержать подстановки (например '@DIRS@'); сам Automake в действительности не проверяет содержимое этой переменной.

Если определена переменная SUBDIRS, то ваш файл 'configure.in' должен включать макрос AC_PROG_MAKE_SET.

Использование SUBDIRS не ограничено только 'Makefile.am' верхнего уровня. Automake может использоваться для создания пакетов любой глубины.

По умолчанию Automake создает файлы 'Makefile', которые работают, выполняя сначала make в подкаталогах (*постфиксный метод*). Однако, можно изменить это поведение, поместив '.' в переменную SUBDIRS. Например, поместив '.' в начало списка, вы заставите выполнять make сначала в текущем каталоге, а затем уже в подкаталогах (*префиксный метод*).

7 Построение программ и библиотек

Большая часть функциональности Automake направлена на то, чтобы облегчить компиляцию программ и библиотек.

7.1 Построение программ

В каталоге, содержащем исходные тексты, из которых будет построена программа (в отличие от библиотеки), в основном используется макрос 'PROGRAMS'. Программы могут быть установлены в каталоги bindir, sbindir, libexecdir, pkglibdir, или же вообще не устанавливаться ('noinst').

Например:

```
bin_PROGRAMS = hello
```

В этом простом примере результирующий 'Makefile.in' будет содержать код для генерации программы с именем hello. Переменная hello_SOURCES используется для указания того, какие файлы исходных текстов будут использованы для компиляции исполняемого файла:

```
hello_SOURCES = hello.c version.c getopt.c getopt1.c getopt.h system.h
```

В результате этого каждый упомянутый в этой переменной файл '.c' будет скомпилирован в соответствующий файл '.o'. Затем все они компонуются для создания 'hello'.

Если переменная '*prog*_SOURCES' необходима, но не указана, то она получает значение по умолчанию, равное единственному файлу 'prog.c'.

В одном каталоге могут компилироваться несколько программ. Эти программы могут совместно использовать один и тот же исходный файл, который должен быть указан в каждом определении '_SOURCES'.

Заголовочные файлы, перечисленные в определении '_SOURCES', включаются в дистрибутив, а в других случаях игнорируются. В том случае, если это не очень удобно, вы не должны включать файл, созданный 'configure' в переменную '_SOURCES'; этот файл не должен распространяться. Файлы Lex ('.l') и Yacc ('.y') также должны быть перечислены; смотрите раздел Раздел 7.6 [Поддержка Yacc и Lex], с. 22.

Automake должен знать все файлы исходных текстов, которые могут участвовать в компиляции программы, даже если не все файлы будут использоваться в каждом конкретном случае. Файлы, которые компилируются только при выполнении определенных условий, должны быть перечислены в соответствующей переменной 'EXTRA_'. Например, если 'hello-linux.c' будет, в зависимости от условий, включен в программу hello, то файл 'Makefile.am' должен содержать:

```
EXTRA_hello_SOURCES = hello-linux.c
```

Иногда также полезно аналогичным образом определить во время конфигурации, какие программы будут скомпилированы. Например, GNU cpio создает программы mt и rmt только при выполнении определенных условий.

В этом случае вы должны уведомить Automake обо всех программах, которые могут быть построены, но в то же время заставить сгенерированный файл 'Makefile.in' использовать программы, заданные при выполнении configure. Это делается подстановкой значений при выполнении configure в каждом определении '_PROGRAMS'. А все программы, которые можно создать, перечисляются в переменной EXTRA_PROGRAMS.

Если вы хотите скомпоновать программу с библиотеками, которые не найдены configure, то для этого вы должны использовать переменную LDADD. Эта переменная может использоваться для добавления ключей в командную строку компоновщика.

Иногда несколько программ компилируются в одном каталоге, но при этом у них различные требования к компоновке. В этом случае для переопределения глобальной переменной LDADD вы можете использовать переменную '*prog*_LDADD' (где *prog* является именем программы, как оно появляется в некоторых переменных '_PROGRAMS', и обычно записывается буквами в нижнем регистре). Если эта переменная существует для заданной программы, то программа компонуется без использования LDADD.

Например, в GNU cpio, pax, cpio и mt компонуются с библиотекой 'libcpio.a'. Однако, программа rmt, создаваемая в том же каталоге, не имеет такого требования к компоновке. Более того, программы mt и rmt создаются только на определенных типах машин. Вот как выглядит 'src/Makefile.am' из поставки cpio (в сокращенном виде):

```
bin_PROGRAMS = cpio pax @MT@
libexec_PROGRAMS = @RMT@
EXTRA_PROGRAMS = mt rmt
```

```
LDADD = ../lib/libcpio.a @INTLLIBS@
rmt_LDADD =

cpio_SOURCES = ...
pax_SOURCES = ...
mt_SOURCES = ...
rmt_SOURCES = ...
```

‘*prog*_LDADD’ не подходит для передачи специфических для программы флагов компоновщика (за исключением ‘-l’ и ‘-L’). Для передачи таких флагов используйте переменную ‘*prog*_LDFLAGS’.

Также иногда полезно собирать программу, в зависимости от цели, которая не является частью этой программы. Это может быть сделано с использованием переменной ‘*prog*_DEPENDENCIES’. Каждая программа зависит от содержимого такой переменной, но никакой дополнительной интерпретации не производится.

Если переменная ‘*prog*_DEPENDENCIES’ не определена, то она будет вычислена Automake. Автоматически присвоенная ей величина является содержимым переменной ‘*prog*_LDADD’ с большинством подстановок configure. Ключи ‘-l’ и ‘-L’ удаляются. Остающимися подстановками configure являются только ‘@LIBOBJS@’ и ‘@ALLOCA@’; они остаются потому, что они заведомо не приведут к генерации неправильных значений для ‘*prog*_DEPENDENCIES’.

7.2 Построение библиотеки

Построение библиотеки по большей части аналогично построению программы. В этом случае именем основной переменной является ‘LIBRARIES’. Библиотеки могут быть установлены в каталоги libdir или в pkglibdir.

Смотрите См. Раздел 7.4 [Разделяемые библиотеки], с. 21, для получения информации о том, как компилировать разделяемые библиотеки, используя программу Libtool и основную переменную ‘LTLIBRARIES’.

Каждая переменная ‘_LIBRARIES’ является списком библиотек, которые должны быть построены. Например, для того, чтобы создать библиотеку с именем ‘libcpio.a’, но не устанавливать ее, вы должны написать:

```
noinst_LIBRARIES = libcpio.a
```

Файлы исходных текстов для библиотек определяются точно так же, как и для программ, через переменные ‘_SOURCES’. Заметьте, что имя библиотеки является канонизированным (см. Раздел 2.5 [Канонизация], с. 4), так что переменная ‘_SOURCES’ для ‘liblob.a’ является равной ‘liblob_a_SOURCES’, а не ‘liblob.a_SOURCES’.

Дополнительные объекты могут быть добавлены в библиотеку, используя переменную ‘*library*_LIBADD’. Это можно использовать для объектов, определенных configure. Опять пример из cpio:

```
libcpio_a_LIBADD = @LIBOBJS@ @ALLOCA@
```

7.3 Специальная обработка переменных ‘LIBOBJS’ и ‘ALLOCA’

Automake явно распознает использование переменных `@LIBOBJS@` и `@ALLOCA@`, и использует эту информацию вместе со списком файлов `LIBOBJS`, полученным из ‘`configure.in`’, для автоматического включения соответствующих файлов исходных текстов в дистрибутив (см. Глава 13 [Дистрибутив], с. 36). Эти файлы исходных текстов также обрабатываются для автоматического определения зависимостей; смотрите раздел См. Раздел 7.11 [Зависимости], с. 30.

Использование `@LIBOBJS@` и `@ALLOCA@` распознается в переменных ‘`_LDADD`’ и ‘`_LIBADD`’.

7.4 Построение разделяемых библиотек

Построение разделяемой библиотеки является относительно сложной задачей. Для помощи в платформонезависимом построении разделяемых библиотек была создана программа GNU Libtool (см. раздел “Introduction” в *Libtool Manual*).

Automake использует Libtool для построения библиотек, указанных в переменной ‘`LTLIBRARIES`’. Каждая переменная ‘`_LTLIBRARIES`’ является списком разделяемых библиотек, которые нужно построить. Например, для создания библиотеки с именем ‘`libgettext.a`’ и соответствующей ей разделяемой библиотеки, а также их установки в ‘`libdir`’, вы должны написать:

```
lib_LTLIBRARIES = libgettext.la
```

Заметьте, что разделяемые библиотеки *должны* быть установлены, так что использование `check_LTLIBRARIES` не разрешено. Однако же, разрешено использование переменной `noinst_LTLIBRARIES`. Эта возможность должна быть использована для “готовых библиотек” libtool.

Для каждой библиотеки переменная ‘*library*`_LIBADD`’ содержит имена дополнительных объектов libtool (файлы ‘`.lo`’), которые будет добавляться в разделяемую библиотеку. Переменная ‘*library*`_LDFLAGS`’ содержит любые дополнительные флаги libtool, такие как ‘`-version-info`’ или ‘`-static`’.

В то время как обычные библиотеки могут включать `@LIBOBJS@`, библиотеки, использующие libtool, должны использовать `@LTLIBOBJS@`. Это требуется, поскольку имена объектных файлов, над которыми работает libtool, не обязательно оканчиваются на ‘`.o`’. Руководство по libtool содержит более детальное описание этой темы.

Для библиотек, устанавливаемых в некоторый каталог, Automake будет автоматически снабжать их соответствующим ключом ‘`-rpath`’. Однако для библиотек, определенных во время конфигурации (и таким образом перечисленных в переменной `EXTRA_LTLIBRARIES`), Automake не знает возможных каталогов установки; для таких библиотек вы должны сами добавить ключ ‘`-rpath`’ в соответствующую переменную ‘`_LDFLAGS`’.

Для подробного описания смотрите См. ⟨undefined⟩ [libtool], с. ⟨undefined⟩.

7.5 Переменные, используемые при построении программ

Иногда полезно знать, какие переменные ‘Makefile’ Automake использует для компиляции; например, вам в некоторых случаях может быть необходимо использовать ваш собственный способ компиляции.

Некоторые переменные наследуются от Autoconf: это CC, CFLAGS, CPPFLAGS, DEFS, LDFLAGS и LIBS.

Также есть некоторые дополнительные переменные, определенные самим Automake:

INCLUDES Задает список ключей ‘-I’. Может быть установлен в вашем файле ‘Makefile.am’, если у вас есть специальные каталоги, в которых вы хотите осуществлять поиск. Automake автоматически подставляет некоторые ключи ‘-I’. В частности, он генерирует строку с ключами ‘-I$(srcdir)’ и ‘-I’, указывающий на каталог с файлом ‘config.h’ (если вы используете AC_CONFIG_HEADER или AM_CONFIG_HEADER).

INCLUDES может быть использован для других ключей cpp, а не только для ‘-I’. Например, эта переменная используется для передачи специальных ключей ‘-D’ вашему компилятору.

COMPILE Эта команда используется для компиляции исходных файлов на языке C. Имя файла исходных текстов добавляется для формирования полной командной строки.

LINK Эта команда используется для компоновки программы на языке C.

7.6 Поддержка Yacc и Lex

В Automake есть некоторая поддержка Yacc и Lex.

Automake предполагает, что файлы с расширением ‘.c’, которые создаются yacc (или lex) должны называться точно так же, как и входной файл. Это значит, что при использовании исходного yacc-файла ‘foo.y’ Automake будет считать, что промежуточный файл будет называться ‘foo.c’ (а не более традиционно, ‘y.tab.c’).

Расширение имени yacc-файла используется для определения расширения имени готового файла на языках ‘C’ или ‘C++’. Файлы с расширением ‘.y’ будут превращены в файлы с расширением ‘.c’; аналогично ‘.yy’ станут ‘.cc’; ‘.y++’ станут ‘c++’; и ‘.yxx’ станут ‘.cxx’.

Подобным образом исходные тексты на lex могут быть использованы для создания файлов на ‘C’ или ‘C++’; распознаются файлы с расширениями ‘.l’, ‘.ll’, ‘.l++’ и ‘.lxx’.

Вы не должны явно упоминать промежуточные файлы (на ‘C’ или ‘C++’) в переменных ‘SOURCES’; вы должны указывать только список исходных файлов.

Промежуточные файлы, созданные yacc (или lex), будут включены в созданный дистрибутив. Таким образом, пользователю не обязательно иметь у себя yacc или lex.

Если был обнаружен исходный текст на `yacc`, то ваш файл '`configure.in`' должен определить переменную '`YACC`'. Это легко делается макросом '`AC_PROG_YACC`' (см. раздел "Проверка отдельных программ" в *Руководство Autoconf*).

Аналогичным образом, если есть исходный текст `lex`, то в '`configure.in`' должна быть определена переменная '`LEX`'. Вы можете использовать для этого макрос '`AC_PROG_LEX`' (см. раздел "Проверка отдельных программ" в *Руководство Autoconf*). Поддержка `lex` в Automake также требует использования макроса '`AC_DECL_YYTEXT`' — automake необходимо знать значение '`LEX_OUTPUT_ROOT`'. Все эти тонкости обрабатываются при использовании макроса `AM_PROG_LEX` (см. Раздел 5.4 [Макросы], с. 15).

Automake делает возможным включение в одну программу нескольких исходных файлов `yacc` (или `lex`). Для запуска `yacc` (или `lex`) в подкаталогах Automake использует небольшую программу, `ylwrap`. Это необходимо, поскольку имя выходного файла yacc является фиксированным, а параллельное выполнение make может одновременно запустить несколько экземпляров `yacc`. Программа `ylwrap` распространяется вместе с Automake. Она должна быть в каталоге, указанном переменной '`AC_CONFIG_AUX_DIR`' (см. раздел "Нахождение ввода 'configure'" в *Руководство Autoconf*) или в текущем каталоге, если данный макрос не используется в '`configure.in`'.

Для `yacc`, недостаточно просто управлять блокировками. Результирующий файл `yacc` всегда использует внутри одни и те же имена символов, так что невозможно скомпоновать два парсера `yacc` в одну и ту же программу.

Мы рекомендуем использование следующего приема с переименованием объектов, который используется в `gdb`:

```
#define yymaxdepth c_maxdepth
#define yyparse c_parse
#define yylex c_lex
#define yyerror c_error
#define yylval c_lval
#define yychar c_char
#define yydebug c_debug
#define yypact c_pact
#define yyr1 c_r1
#define yyr2 c_r2
#define yydef c_def
#define yychk c_chk
#define yypgo c_pgo
#define yyact c_act
#define yyexca c_exca
#define yyerrflag c_errflag
#define yynerrs c_nerrs
#define yyps c_ps
#define yypv c_pv
#define yys c_s
#define yy_yys c_yys
#define yystate c_state
#define yytmp c_tmp
#define yyv c_v
#define yy_yyv c_yyv
```

```
#define yyval c_val
#define yylloc c_lloc
#define yyreds c_reds
#define yytoks c_toks
#define yylhs c_yylhs
#define yylen c_yylen
#define yydefred c_yydefred
#define yydgoto c_yydgoto
#define yysindex c_yysindex
#define yyrindex c_yyrindex
#define yygindex c_yygindex
#define yytable  c_yytable
#define yycheck  c_yycheck
#define yyname   c_yyname
#define yyrule   c_yyrule
```

Для каждого '`#define`' замените префикс '`c_`' на то, что вы хотите использовать. Эти определения работают для программ `bison`, `byacc` и традиционных `yacc`. Если вы обнаружили, что какой-нибудь генератор парсеров использует символы, не указанные в этом списке, то сообщите нам новое имя, чтобы мы добавили его.

7.7 Поддержка C++

Automake полностью поддерживает C++.

Любой пакет, содержащий код на C++, должен определить переменную '`CXX`' в файле '`configure.in`'; самым простым способом сделать это является использование макроса `AC_PROG_CXX` (см. раздел "Проверка отдельных программ" в *Руководство Autoconf*).

Несколько дополнительных переменных определяются при обнаружении исходных файлов на C++:

`CXX` Имя компилятора C++.

`CXXFLAGS` Флаги, передаваемые компилятору C++.

`CXXCOMPILE`
: Команда, используемая для компиляции исходных текстов на языке C++. Имя файла с исходным текстом добавляется к ней для формирования полной командной строки.

`CXXLINK` Эта команда используется для компоновки программы на C++.

7.8 Поддержка Fortran 77

Automake полностью поддерживает Fortran 77.

Любой пакет, содержащий исходные тексты на языке Fortran 77, должен определить выходную переменную '`F77`' в файле '`configure.in`'; самым простым способом является использование макроса `AC_PROG_F77` (см. раздел "Проверка отдельных программ" в *Руководство Autoconf*). См. Раздел 7.8.4 [Fortran 77 и Autoconf], с. 28.

При использовании исходных текстов на Fortran 77 определяются несколько дополнительных переменных:

`F77` Имя компилятора Fortran 77.

`FFLAGS` Флаги, передаваемые компилятору Fortran 77.

`RFLAGS` Флаги, передаваемые компилятору Ratfor.

`F77COMPILE`
Команда, используемая для компиляции исходных текстов на Fortran 77. Имя файла исходных текстов добавляется к ней чтобы получить полную командную строку.

`FLINK` Команда, используемая для компоновки программы на Fortran 77 или разделяемой библиотеки.

Automake может вдобавок к компиляции выполнять предварительную обработку исходных файлов на Fortran 77 и Ratfor[1]. Automake также содержит некоторую поддержку для создания программ и разделяемых библиотек, которые написаны на смеси Fortran 77 и других языков (см. Раздел 7.8.3 [Использование Fortran 77 с C и C++], с. 26).

Это описывается в следующих разделах.

7.8.1 Предварительная обработка файлов Fortran 77

Файл '`N.f`' автоматически создается из файла '`N.F`' или '`N.r`'. Это правило запускает препроцессор для преобразования исходных текстов Fortran 77 или Ratfor с директивами препроцессора в строгий исходный текст Fortran 77. Вот точные команды, которые используются для этого:

'`.F`' `$(F77) -F $(DEFS) $(INCLUDES) $(AM_CPPFLAGS) $(CPPFLAGS) $(AM_FFLAGS) $(FFLAGS)`

'`.r`' `$(F77) -F $(AM_FFLAGS) $(FFLAGS) $(AM_RFLAGS) $(RFLAGS)`

7.8.2 Компиляция файлов Fortran 77

'`N.o`' автоматически создается из '`N.f`', '`N.F`' или '`N.r`' запуском компилятора Fortran 77. Для компиляции используются следующий команды:

'`.f`' `$(F77) -c $(AM_FFLAGS) $(FFLAGS)`

'`.F`' `$(F77) -c $(DEFS) $(INCLUDES) $(AM_CPPFLAGS) $(CPPFLAGS) $(AM_FFLAGS) $(FFLAGS)`

'`.r`' `$(F77) -c $(AM_FFLAGS) $(FFLAGS) $(AM_RFLAGS) $(RFLAGS)`

[1] Много, если не большая часть информации в следующих разделах, имеющая отношение к предварительной обработке программ на Fortran 77, была взята как есть из раздел "Каталог правил" в *Руководство GNU Make*.

7.8.3 Использование Fortran 77 с C и C++

В настоящее время Automake предоставляет *ограниченную* поддержку создания программ и разделяемых библиотек, которые являются смесью Fortran 77 и C и/или C++. Однако существует много других вопросов, возникающих при смешивании кода на Fortran 77 с кодом на других языках, которые в настоящее время *не* обрабатываются Automake, но обрабатываются другими пакетами[2].

[2] например, пакет cfortran, обрабатывает все вопросы взаимодействия языков, и работает почти со всеми компиляторами Fortran 77, C и C++ на почти всех платформах. Однако, сейчас `cfortran` не является Free Software, но будет являться таковым в следующей версии

Automake может предоставить вам помощь двумя способами:

1. Автоматический выбор компоновщика в зависимости от комбинации исходного кода.
2. Автоматический выбор флагов компоновщика (например, '`-L`' и '`-l`') для передачи автоматически выбранному компоновщику для компоновки с соответствующими внутренними библиотеками и библиотеками времени исполнения Fortran 77.

   Эти дополнительные флаги компоновщика Fortran 77 выдаются в выходную переменную `FLIBS` макросом Autoconf `AC_F77_LIBRARY_LDFLAGS`, который поставляется со свежими версиями Autoconf (Autoconf версии 2.13 и выше). См. раздел "Характеристики компилятора Fortran 77" в *Autoconf*.

Если Automake определяет, что программа или разделяемая библиотека (упомянутые в каких-либо основных переменных `_PROGRAMS` или `_LTLIBRARIES`) содержит исходный код, который является смесью Fortran 77 и C и/или C++, то он требует вызова макроса `AC_F77_LIBRARY_LDFLAGS` в файле '`configure.in`', и чтобы в соответствующей переменной `_LDADD` (для программ) или `_LIBADD` (для разделяемых библиотек) появились ссылки либо на `$(FLIBS)`, либо на `@FLIBS@`. От человека, пишущего '`Makefile.am`', требуется убедиться, что переменные `$(FLIBS)` или `@FLIBS@` находятся в соответствующих переменных `_LDADD` или `_LIBADD`.

Например, рассмотрим следующий '`Makefile.am`':

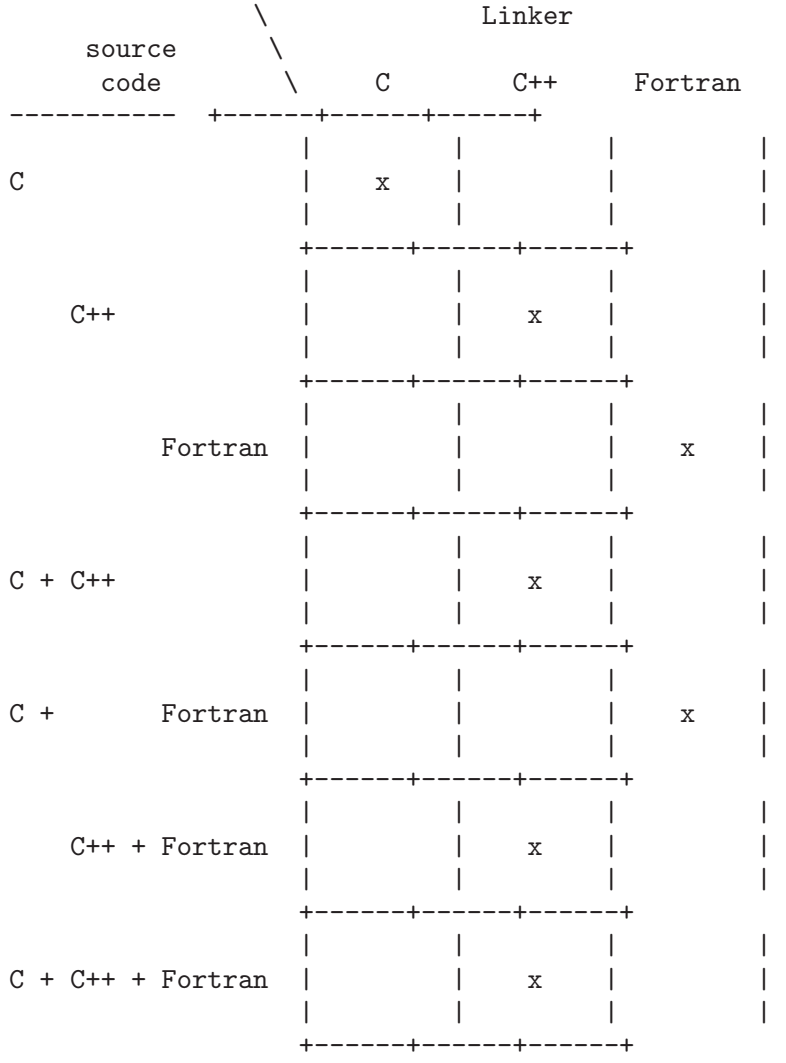
```
bin_PROGRAMS = foo
foo_SOURCES  = main.cc foo.f
foo_LDADD    = libfoo.la @FLIBS@

pkglib_LTLIBRARIES = libfoo.la
libfoo_la_SOURCES  = bar.f baz.c zardoz.cc
libfoo_la_LIBADD   = $(FLIBS)
```

В этом случае Automake будет настаивать, чтобы макрос `AC_F77_LIBRARY_LDFLAGS` был упомянут в '`configure.in`'. Более того, если переменная `@FLIBS@` не была упомянута в переменной `foo_LDADD` и `libfoo_la_LIBADD`, то Automake выдаст предупреждение.

7.8.3.1 Как выбирается компоновщик

Следующая диаграмма показывает, как Automake производит выбор соответствующего компоновщика.

Например, если используемый код на Fortran 77, C и C++ компонуется в одну программу, то выбирается компоновщик C++. В этом случае, если компоновщики C или Fortran 77 требуют какие-либо специальные библиотеки, которые не подключаются компоновщиком C++, то они должны быть вручную добавлены пользователем в переменные `_LDADD` или `_LIBADD` файла '`Makefile.am`'.



7.8.4 Использование Fortran 77 с Autoconf

Имеющаяся в Automake поддержка Fortran 77 требует наличия свежей версии Autoconf, которая поддерживает Fortran 77. Полная поддержка Fortran 77 была добавлена в Autoconf 2.13, так что вы можете использовать эту версию Autoconf или более позднюю.

7.9 Поддержка других языков

В настоящее время Automake включает в себя полную поддержку только C, C++ (см. Раздел 7.7 [Поддержка C++], с. 24) и Fortran 77 (см. Раздел 7.8 [Поддержка Fortran 77], с. 24). Поддержка других языков находится в зачаточном состоянии и будет улучшена по требованию пользователей.

7.10 Автоматическая де-ANSI-фикация

Хотя стандарты GNU позволяют использование ANSI C, это может привести к ограничению переносимости пакета на некоторые старые компиляторы (особенно SunOS).

Automake позволяет вам обойти проблему с такими машинами путем *де-ANSI-фикации* каждого исходного файла перед компиляцией.

Если в ‘`Makefile.am`’ переменная `AUTOMAKE_OPTIONS` (см. Глава 15 [Ключи], с. 38) содержит ключ `ansi2knr`, то в генерируемый файл ‘`Makefile.in`’ будет вставлен код для де-ANSI-фикации.

Это заставит считать каждый исходный текст на языке C соответствующим ANSI C. Если доступен компилятор, соответствующий ANSI C, то он будет использован. В противном случае для преобразования исходных файлов в стандарт K&R будет использована программа `ansi2knr`, а затем преобразованные файлы будут скомпилированы.

Программа `ansi2knr` совершенно бесхитростна. Она предполагает, что исходный текст будет отформатирован определенным способом; подробное описание находится на странице руководства по `ansi2knr`.

Поддержка де-ANSI-фикации требует наличия файлов ‘`ansi2knr.c`’ и ‘`ansi2knr.1`’ в том же пакете, где находятся и исходные тексты на ANSI C; эти файлы поставляются в комплекте Automake. Файл ‘`configure.in`’ должен также содержать вызов макроса `AM_C_PROTOTYPES` (см. Раздел 5.4 [Макросы], с. 15).

Automake также работает в тех случаях, когда файлы `ansi2knr` находятся в другом подкаталоге текущего пакета. Это делается добавлением в опцию `ansi2knr` относительного пути к соответствующему каталогу. Например, предположим, что исходные тексты на ANSI C располагаются в подкаталогах ‘`src`’ и ‘`lib`’. Файлы ‘`ansi2knr.c`’ и ‘`ansi2knr.1`’ находятся в подкаталоге ‘`lib`’. Вот что должно быть написано в ‘`src/Makefile.am`’:

```
AUTOMAKE_OPTIONS = ../lib/ansi2knr
```

Если никакой префикс не задан, то считается, что файлы находятся в текущем каталоге.

Файлы, перечисленные в переменной `LIBOBJS` и нуждающиеся в де-ANSI-фикации, не будут обрабатываться автоматически. Это происходит из-за того, что `configure` будет генерировать имена объектных файлов в виде ‘`regex.o`’, в то время как `make` будет искать ‘`regex_.o`’ (при выполнении де-ANSI-фикации). В конечном счете эта проблема может быть решена с помощью методов `autoconf`, но для этого вы должны поместить следующий код в ваш файл ‘`configure.in`’, как раз перед вызовом макроса `AC_OUTPUT`:

```
# This is necessary so that .o files in LIBOBJS are also built via
# the ANSI2KNR-filtering rules.
LIBOBJS=‘echo $LIBOBJS|sed ’s/\.o /\$U.o /g;s/\.o$/\$U.o/’‘
```

7.11 Автоматическое отслеживание зависимостей

Для разработчика зачастую мучительно бывает постоянно обновлять файл ‘`Makefile.in`’ при изменении зависимостей включаемых в проект файлов. Automake предоставляет возможность автоматического отслеживания изменения зависимостей и записи информации о них в сгенерированный ‘`Makefile.in`’.

В настоящее время эта поддержка требует использования GNU `make` и `gcc`. В будущем может быть возможным поставка другой программы генерации зависимостей, если это будет требоваться. По умолчанию этот режим разрешен, если в текущем каталоге определена любая программа или библиотека на C, так что вы можете получить от не-GNU make ошибку ‘`Должен быть разделитель`’.

Когда вы решаете создать дистрибутив, то цель `dist` перезапустит `automake` с ключом ‘`-include-deps`’ и другими ключами. См. Глава 4 [Запуск Automake], с. 8, и Глава 15 [Ключи], с. 38. При этом предварительно сгенерированные зависимости будут помещены в созданный ‘`Makefile.in`’ и, таким образом, они окажутся в дистрибутиве. При этом в дистрибутив код генерации зависимостей не будет включен в дистрибутив, так что человек, загрузивший ваш дистрибутив и не использующий GNU `make` и `gcc`, не получит ошибки.

При добавлении зависимостей в ‘`Makefile.in`’, из них автоматически удаляются все специфические для данной системы зависимости. Это может быть сделано перечислением файлов в переменной ‘`OMIT_DEPENDENCIES`’. Например, Automake удаляет все ссылки на системные заголовочные файлы. Иногда полезно указать, чтобы были удалены отдельные заголовочные файлы. Например, если ваш файл ‘`configure.in`’ использует макрос ‘`AM_WITH_REGEX`’, то любая зависимость от файла ‘`rx.h`’ или ‘`regex.h`’ должны быть удалена, потому что правильное значение не может быть известно до того, как пользователь выполнит конфигурацию пакета.

Оказывается, Automake достаточно умен для обработки именно этого случая использования заголовочных файлов библиотеки регулярных выражений. Он также автоматически убирает зависимость от ‘`libintl.h`’ при использовании ‘`AM_GNU_GETTEXT`’.

Автоматическое отслеживание зависимостей может быть запрещено помещением `no-dependencies` в переменную `AUTOMAKE_OPTIONS`.

Если вы распаковываете дистрибутив, созданный `make dist`, и хотите включить отслеживание зависимостей, то просто перезапустите `automake`.

Файлы зависимостей помещаются в подкаталог с именем ‘`.deps`’ каталога, где происходит построение. Эти зависимости являются специфическими для машины. Можете удалить их, если хотите; они будут автоматически пересозданы при следующей сборке.

8 Другие порожденные объекты

Automake может обрабатывать порожденные объекты, которые не являются программами на C. Иногда поддержка построения таких объектов должна быть предоставлена явно, но Automake все равно будет автоматически отрабатывать процесс установки и создания дистрибутива.

8.1 Исполняемые скрипты

Существует также возможность определить и установить программы, которые являются скриптами. Эти программы перечисляются с использованием основной переменной 'SCRIPTS'. Automake не определяет зависимости для скриптов; файл 'Makefile.am' должен явно включать в себя соответствующие правила.

Automake не считает, что скрипты являются унаследованными объектами; такие скрипты должны удаляться вручную (см. Глава 12 [Очистка], с. 36).

Сама программа automake является скриптом на Perl, так что она генерируется на этапе конфигурации из 'automake.in'. Вот как это обрабатывается:

```
bin_SCRIPTS = automake
```

Поскольку automake появляется в макросе AC_OUTPUT, то для нее цель создается автоматически.

Скрипты могут быть установлены в каталоги bindir, sbindir, libexecdir или pkgdatadir.

8.2 Заголовочные файлы

Заголовочные файлы определяются семейством переменных 'HEADERS'. Обычно заголовочные файлы не устанавливаются, так что в большинстве случаев будет определена переменная noinst_HEADERS.

Все заголовочные файлы должны быть перечислены; отсутствующие файлы не будут включены в дистрибутив. Часто лучше всего перечислить неустанавливаемые заголовочные файлы вместе с другими исходными текстами программы. См. Глава 7 [Программы], с. 18. Заголовочные файлы, перечисленные в переменных '_SOURCES', не надо указывать ни в одной из переменных '_HEADERS'.

Заголовочные файлы могут быть установлены в каталоги includedir, oldincludedir или pkgincludedir.

8.3 Файлы данных, не зависимые от архитектуры машины

Automake поддерживает установку различных файлов данных, используя семейство переменных 'DATA'.

Такие данные могут быть установлены в каталоги datadir, sysconfdir, sharedstatedir, localstatedir или pkgdatadir.

По умолчанию файлы данных *не* включаются в дистрибутив.

Вот как Automake устанавливает свои вспомогательные файлы данных:

```
pkgdata_DATA = clean-kr.am clean.am ...
```

8.4 Построение исходных текстов

Время от времени файлы, которые могли бы быть названы *исходными* (например, файлы '`.h`' в C), в действительности порождаются из других файлов. Такие файлы должны быть перечислены в переменной `BUILT_SOURCES`.

Построенные исходные тексты по умолчанию не компилируются. Для компиляции исходных текстов вы должны явно указать их в других переменных '`_SOURCES`'.

Заметьте, что в некоторые случаях, `BUILT_SOURCES` будет работать достаточно странным образом. Для того, чтобы построение исходных текстов работало с автоматическим отслеживанием зависимостей, файл '`Makefile`' должен зависеть от `$(BUILT_SOURCES)`. При этом такие исходные тексты могут начать пересобираться в самый неудобный момент.

9 Другие утилиты GNU

Поскольку Automake в основном предназначен для генерации файлов '`Makefile.in`' для использования в программах проекта GNU, то он старается взаимодействовать с другими утилитами GNU.

9.1 Emacs Lisp

Automake предоставляет некоторую поддержку Emacs Lisp. Основная переменная '`LISP`' используется для хранения списка файлов '`.el`'. Возможными префиксами являются '`lisp_`' и '`noinst_`'. Заметьте, что если определена переменная `lisp_LISP`, то в '`configure.in`' должен использоваться макрос `AM_PATH_LISPDIR` (см. Раздел 5.4 [Макросы], с. 15).

По умолчанию Automake будет производить байт-компиляцию всех исходных текстов Emacs Lisp, используя Emacs, который найден при выполнении макроса `AM_PATH_LISPDIR`. Если вы не хотите производить байт-компиляцию, то просто определите переменную `ELCFILES` с пустым значением. Байт-скомпилированные файлы Emacs Lisp не переносимы между разными версиями Emacs, так что отключите компиляцию, если ожидаете, что целевые машины будут иметь несколько разных версий Emacs. К тому же, многие пакеты на самом деле работают после байт-компиляции не лучше. Однако мы рекомендуем вам оставить эту возможность разрешенной. Серверам с такими странными установками лучше дать возможность справиться самим, чем затруднять установку для остальных людей.

9.2 Gettext

Если в файле '`configure.in`' есть макрос `AM_GNU_GETTEXT`, то Automake включает поддержку GNU gettext, системы каталогов сообщений для интернационализации (см. раздел "GNU Gettext" в *Утилиты GNU gettext*).

Поддержка `gettext` в Automake требует добавления в пакет двух подкаталогов, '`intl`' и '`po`'. Automake проверяет, что эти подкаталоги существуют и упомянуты в переменной `SUBDIRS`.

Также Automake проверяет, что определение переменной `ALL_LINGUAS` в файле '`configure.in`' соответствует в точности всем файлам '`.po`', ни больше, ни меньше.

9.3 Guile

Automake обеспечивает некоторую автоматическую поддержку написания модулей Guile. Automake включит поддержку Guile, если в '`configure.in`' используется макрос `AM_INIT_GUILE_MODULE`.

В настоящее время поддержка Guile означает, что при выполнении макроса `AM_INIT_GUILE_MODULE` будет:

- Запущен макрос `AM_INIT_AUTOMAKE`.
- Запущен макрос `AC_CONFIG_AUX_DIR` с параметром '`..`'.

Когда Guile станет лучше поддерживать модули, нет никаких сомнений, что их поддержка в Automake будет развиваться.

9.4 Libtool

Automake предоставляет поддержку GNU Libtool (см. раздел "Introduction" в *The Libtool Manual*) с основной переменной '`LTLIBRARIES`'. См. Раздел 7.4 [Разделяемые библиотеки], с. 21.

9.5 Java

Automake предоставляет минимальную поддержку компиляции файлов Java, используя основную переменную '`JAVA`'.

Все файлы '`.java`', перечисленные в переменной '`_JAVA`', будут скомпилированы с помощью `JAVAC`. По умолчанию, файлы с расширением '`.class`' не включаются в дистрибутив.

В настоящее время Automake принуждает к тому, что в каждом '`Makefile.am`' может быть использована только одна переменная '`_JAVA`'. Причиной этого ограничения является то, что невозможно узнать, какие файлы '`.class`' будут сгенерированы из файлов '`.java`' — так что может быть невозможным узнать, какие файлы и куда необходимо устанавливать.

10 Построение документации

В настоящее время Automake обеспечивает поддержку Texinfo и страниц руководства.

10.1 Texinfo

Если текущий каталог содержит исходный текст Texinfo, то вы должны указать это с помощью основной переменной 'TEXINFOS'. В общем случае файлы Texinfo могут быть преобразованы в формат info, и поэтому макрос info_TEXINFOS является наиболее часто используемым. Заметьте, что имена файлов исходных текстов Texinfo должны заканчиваться на '.texi' или '.texinfo'.

Если файл '.texi' включает в себя с помощью '@include' файл 'version.texi', то он будет сгенерирован автоматически. Файл 'version.texi' определяет три макроса Texinfo, на которые вы можете ссылаться: EDITION, VERSION и UPDATED. Первые два макроса содержат номер версии вашего пакета (но для ясности они разделены на две части); последний макрос содержит дату изменения основного файла. Поддержка 'version.texi' требует наличия программы mdate-sh; эта программа поставляется с Automake и автоматически включается при запуске automake с ключом --add-missing.

Иногда файл info зависит от более чем одного файла '.texi'. Например, в пакете GNU Hello, файл 'hello.texi' включает файл 'gpl.texi'. Вы можете сообщить об этих зависимостях Automake, используя переменную *texi*_TEXINFOS. Вот как это делается в GNU Hello:

```
info_TEXINFOS = hello.texi
hello_TEXINFOS = gpl.texi
```

По умолчанию Automake требует наличия файла 'texinfo.tex' в том же каталоге, что и исходные файлы Texinfo. Однако, если вы используете в файле 'configure.in' макрос AC_CONFIG_AUX_DIR (см. раздел "Нахождение ввода 'configure'" в *Руководство Autoconf*), то 'texinfo.tex' ищется в заданном каталоге. Automake устанавливает файл 'texinfo.tex', если задан ключ '-add-missing'.

Если в вашем пакете файлы Texinfo находятся в нескольких каталогах, то вы можете использовать переменную TEXINFO_TEX для того, чтобы сообщить Automake, где найти файл 'texinfo.tex'. Значением этой переменной должен быть относительный путь от текущего файла 'Makefile.am' к файлу 'texinfo.tex':

```
TEXINFO_TEX = ../doc/texinfo.tex
```

Ключ 'no-texinfo.tex' может быть использован для отмены требования наличия файла 'texinfo.tex'. Однако предпочтительней использование переменной TEXINFO_TEX, поскольку это позволяет работать цели dvi.

Automake создает цель install-info; она, очевидно, используется некоторыми. По умолчанию страницы info устанавливаются при выполнении 'make install'. Это поведение может быть изменено, используя ключ no-installinfo.

10.2 Страницы руководства

Пакет также может включать в свой состав справочные страницы (но взгляните на стандартны GNU за информацией о них, раздел "Man Pages" в *The GNU Coding Standards*). Страницы руководства объявляются с помощью основной переменной 'MANS'. Обычно используется макрос man_MANS. Справочные страницы автоматически устанавливаются в зависимости от расширения файлов в соответствующие подкаталоги mandir. Они не включаются в состав дистрибутива автоматически.

По умолчанию страницы руководства устанавливаются при выполнении ‘`make install`’. Однако, поскольку проекты GNU не требуют наличия справочных страниц, то многие разработчики проектов не расходуют время на поддержание справочных страниц в обновленном состоянии. В этих случаях опция `no-installman` отключит установку справочных страниц по умолчанию. Пользователь все равно будет иметь возможность явно установить справочные страницы, используя команду ‘`make install-man`’.

Вот как документация обрабатывается в пакете GNU `cpio` (который включает в себя как документацию в Texinfo, так и справочные страницы):

```
info_TEXINFOS = cpio.texi
man_MANS = cpio.1 mt.1
EXTRA_DIST = $(man_MANS)
```

Исходные тексты Texinfo и страницы info считаются исходными файлами и включаются в состав дистрибутива.

В настоящее время страницы руководства не рассматриваются как исходные файлы, потому что зачастую они генерируются автоматически. Именно поэтому они и не включаются автоматически в состав дистрибутива.

11 Что будет установлено

Automake обрабатывает установку вашей программы после компиляции. Всё перечисленное в `PROGRAMS`, `SCRIPTS`, `LIBRARIES`, `LISP`, `DATA` и `HEADERS` автоматически устанавливается на соответствующие места.

Automake также обрабатывает установку указанных страниц info и страниц руководства.

В том случае, когда программа установки устанавливает пакет на несколько машин с общей структурой каталогов, Automake создает отдельные цели `install-data` и `install-exec` — они позволяют установить машино-независимые части только один раз. Цель `install` зависит от обоих этих целей.

Automake также создает цель `uninstall`, цель `installdirs` и цель `install-strip`.

Также можно расширить этот механизм определением цели `install-exec-local` или `install-data-local`. Если эти цели определены, то они будут запущены при выполнении ‘`make install`’.

Переменные, использующие стандартные префиксы каталогов ‘`data`’, ‘`info`’, ‘`man`’, ‘`include`’, ‘`oldinclude`’, ‘`pkgdata`’ или ‘`pkginclude`’ (например, ‘`data_DATA`’) будут устанавливаться при выполнении цели ‘`install-data`’.

Переменные, использующие стандартные префиксы каталогов ‘`bin`’, ‘`sbin`’, ‘`libexec`’, ‘`sysconf`’, ‘`localstate`’, ‘`lib`’ или ‘`pkglib`’ (например, ‘`bin_PROGRAMS`’), устанавливаются целью ‘`install-exec`’.

Любые переменные, использующие определенные пользователем префиксы каталогов со словом ‘`exec`’ в имени (например, ‘`myexecbin_PROGRAMS`’ устанавливаются целью ‘`install-exec`’. Все другие определенные пользователем префиксы устанавливаются целью ‘`install-data`’.

Automake генерирует поддержку переменной ‘DESTDIR’ во всех правилах установки. Переменная ‘DESTDIR’ используется в процессе выполнения ‘make install’ для перемещения устанавливаемых объектов в область установки. К каждому объекту и пути добавляется значение переменной ‘DESTDIR’ до того, как быть скопированным в область установки. Вот пример типичного использования DESTDIR:

```
make DESTDIR=/tmp/staging install
```

Это помещает устанавливаемые объекты в дерево каталогов, которое создано в каталоге ‘/tmp/staging’. Если устанавливаются файлы ‘/gnu/bin/foo’ и ‘/gnu/share/aclocal/foo.m4’, то вышеприведенная команда установит ‘/tmp/staging/gnu/bin/foo’ и ‘/tmp/staging/gnu/share/aclocal/foo.m4’.

Это свойство часто используется для построения пакетов и установок. Для получения дополнительной информации смотрите раздел “Makefile Conventions” в *The GNU Coding Standards*.

12 Что будет удалено

Стандарты GNU Makefile определяют несколько правил удаления временных файлов.

Обычно Automake сам может определить, какие файлы могут быть удалены. Конечно, Automake также распознает некоторые переменные, определенные для указания дополнительных файлов, которые должны быть удалены. Этими переменными являются MOSTLYCLEANFILES, CLEANFILES, DISTCLEANFILES и MAINTAINERCLEANFILES.

13 Что войдет в дистрибутив

Цель dist, создаваемая в генерируемом файле ‘Makefile.in’, может быть использована для создания сжатого файла tar с дистрибутивом. Имя tar-файла основывается на переменных ‘PACKAGE’ и ‘VERSION’; а точнее, он называется ‘*package*-*version*.tar.gz’. Вы можете использовать переменную make с именем ‘GZIP_ENV’ для того, чтобы управлять запуском gzip. Значением по умолчанию является строка ‘-best’.

В большинстве случаев файлы, необходимые для дистрибутива, автоматически находятся Automake: все файлы исходных текстов автоматически включаются в состав дистрибутива, так же как и все файлы ‘Makefile.am’ и ‘Makefile.in’. Automake также имеет встроенный список часто используемых файлов, которые автоматически включаются в состав дистрибутива, если они существуют в текущем каталоге. Этот список показывается при выполнении ‘automake -help’. Также автоматически включаются файлы, которые читает программа configure (например, файлы исходных текстов, относящиеся к файлам, указанным при запуске макроса AC_OUTPUT).

Все равно, иногда существуют файлы, которые должны входить в состав дистрибутива, но которые не смогли попасть в автоматически созданный список. Эти файлы должны быть перечислены в переменной EXTRA_DIST. Вы можете указывать в переменной EXTRA_DIST файлы из подкаталогов. Вы можете также указывать каталоги: в этом случае весь каталог будет рекурсивно скопирован в дистрибутив.

Если вы определили переменную SUBDIRS, то Automake будет рекурсивно включать подкаталоги в состав дистрибутива. Если SUBDIRS определен условно (см. Глава 18 [Условные операторы], с. 41), то Automake включит в дистрибутив все подкаталоги, которые могут появиться в SUBDIRS. Если вам необходимо указать список каталогов условно, то вы можете задать в переменной DIST_SUBDIRS точный список подкаталогов, которые необходимо включить в дистрибутив.

Время от времени полезно иметь возможность изменить дистрибутив до того, как он будет упакован. Если существует цель dist-hook, то она запускается после создания каталога с дистрибутивом, но до того, как создается файл '.tar' (или '.shar'). Это применяется для распространения файлов из подкаталогов, в которых было бы избыточным создавать файл 'Makefile.am':

```
dist-hook:
        mkdir $(distdir)/random
        cp -p $(srcdir)/random/a1 $(srcdir)/random/a2 $(distdir)/random
```

Automake также создает цель distcheck, которая может помочь убедиться в том, что дистрибутив работает. distcheck создает дистрибутив и пытается его построить с помощью VPATH.

14 Поддержка комплектов тестирования

Automake поддерживает два вида комплектов тестирования.

Если определена переменная TESTS, то ее значение является списком программ, которые надо запустить для проведения тестирования. Программы могут быть либо унаследованными объектами, либо исходными объектами; сгенерированное правило будет искать их и в srcdir, и в '.'. Программы, которые нуждаются в файлах данных должны искать их в каталоге srcdir (который указан в одноименных переменных среды и make), так что они будут работать при построении в отдельном каталоге (см. раздел "Каталоги сборки " в *Руководство Autoconf*), и в частности для цели distcheck (см. Глава 13 [Дистрибутив], с. 36).

Количество сбоев будет напечатано в конце запуска. Если заданная тестовая программа заканчивает работу с кодом 77, то ее результаты игнорируется в завершающем подсчете. Это свойство позволяет игнорировать непереносимые тесты, в тех случаях когда они не имеют значения.

Переменная TESTS_ENVIRONMENT может быть использована для установки переменных среды для запускаемых тестов; при выполнении этого правила переменная среды srcdir устанавливается. Если все ваши тестовые программы являются скриптами, то вы также можете установить переменную TESTS_ENVIRONMENT для вызова командного процессора (например, '$(SHELL) -x'); это свойство может быть полезно при отладке тестов.

Если в переменной AUTOMAKE_OPTIONS указано 'dejagnu', то предполагается использования комплекта тестов на базе dejagnu. Значение переменной DEJATOOL передается как аргумент ключа --tool программы runtest; по умолчанию это имя пакета.

Переменная RUNTESTDEFAULTFLAGS содержит флаги для ключей --tool и --srcdir, которые по умолчанию передаются dejagnu; в случае необходимости это поведение может быть изменено.

Переменные `EXPECT`, `RUNTEST` и `RUNTESTFLAGS` могут быть переопределены для подстановки специфичных для проекта значений. Например, если вам необходимо сделать это для тестирования toolchain компилятора, поскольку значения по умолчанию не должны содержать имена машины и цели.

В других случаях тестирование производится через цель '`make check`'.

15 Изменение поведения Automake

Различные возможности Automake могут контролироваться ключами в файле '`Makefile.am`'. Такие ключи перечислены в специальной переменной с именем `AUTOMAKE_OPTIONS`. В настоящее время распознаются следующие ключи:

`gnits`
`gnu`
`foreign`

`cygnus` Устанавливает соответствующий уровень ограничений. Ключ `gnits` также предполагает наличие ключей `readme-alpha` и `check-news`.

`ansi2knr`
`path/ansi2knr`
Включает автоматическую де-ANSI-фикацию. См. Раздел 7.10 [ANSI], с. 29. Если в начале строки указан путь, то сгенерированный '`Makefile.in`' будет искать программу '`ansi2knr`' в указанном каталоге. В общем случае путь должен быть относительным путем к другому каталогу в данном пакете (хотя в настоящее время Automake не делает проверку этого пути).

`check-news`
Вызывает сбой `make dist` до тех пор, пока номер текущей версии не появится в нескольких первых строках файла '`NEWS`'.

`dejagnu` Заставляет генерировать специфичные для `dejagnu` правила. См. Глава 14 [Тесты], с. 37.

`dist-shar`
Создает цель `dist-shar` также как и обычную цель `dist`. Эта новая цель будет создавать shar-архив дистрибутива.

`dist-zip` Создает цель `dist-zip` также как и обычную цель `dist` Эта новая цель будет создавать zip-архив дистрибутива.

`dist-tarZ`
Создает цель `dist-tarZ` также как и обычную цель `dist` target. Эта новая цель будет создавать сжатый tar-архив дистрибутива; предполагается использование традиционных программ `tar` и `compress`. Предупреждение: Если вы в действительности используете `GNU tar`, то созданный архив может содержать непереносимые конструкции.

`no-dependencies`
Этот ключ похож на ключ командной строки '`-include-deps`', но полезен в тех ситуациях, где вы не имеете необходимости в автоматическом отсле-

живание зависимостей См. Раздел 7.11 [Зависимости], с. 30. В этом случае можно запретить автоматическое отслеживание зависимостей.

`no-installinfo`
: Сгенерированный ‘`Makefile.in`’ не будет по умолчанию обрабатывать и устанавливать страницы info. Однако, цели `info` и `install-info` все равно будут доступны. Этот ключ запрещен при уровне ограничения ‘`GNU`’ и выше.

`no-installman`
: Сгенерированный ‘`Makefile.in`’ не будет по умолчанию устанавливать справочные страницы. Однако, цель `install-man` все равно будет доступна для использования. Этот ключ запрещен при уровне ограничения ‘`GNU`’ и выше.

`no-texinfo.tex`
: Отменяет требования на наличие файла ‘`texinfo.tex`’, даже если файлы texinfo присутствуют в этом каталоге.

`readme-alpha`
: Если этот выпуск является выпуском в стадии альфа и существует файл ‘`README-alpha`’, то он будет добавлен в дистрибутив. Если задан этот ключ, то номер версии может быть представлен в одной из двух форм. Первая форма выглядит следующим образом: ‘*MAJOR*.*MINOR*.*ALPHA*’, где каждый элемент является числом; заключительная точка и номер должны быть опущены для не-альфа выпусков. Вторая форма выглядит следующим образом: ‘*MAJOR*.*MINORALPHA*’, где *ALPHA* это буква; они должны быть убраны для не-альфа выпусков.

version
: Может быть указан номер версии (например, ‘`0.30`’). Если Automake не новее указанной версии, то будет запрещено создание ‘`Makefile.in`’.

Нераспознанные ключи оцениваются `automake`.

16 Различные правила

Существует несколько правил, которые нельзя отнести к вышеперечисленным пунктам.

16.1 Взаимодействие с `etags`

Automake при некоторых обстоятельствах будет генерировать правила для генерации файлов ‘`TAGS`’, которые используются с GNU Emacs.

Если присутствует любой исходный код на C, C++ или Fortran 77, то для каталога будут созданы цели `tags` и `TAGS`.

В каталоге верхнего уровня пакета, состоящего из нескольких каталогов цель `tags` создаст файл, при выполнении которой будет создавать файл ‘`TAGS`’, включающий все файлы ‘`TAGS`’ из подкаталогов.

Также, если определена переменная ETAGS_ARGS, то будет сгенерирована цель tags. Эта переменная предназначена для каталогов, которые содержат исходные файлы, тип которых не понимает etags, но которые можно обработать.

Вот как Automake создает тэги для своих исходных файлов, а также для узлов файла Texinfo:

```
ETAGS_ARGS = automake.in -lang=none \
 -regex='/^@node[ \t]+\([^,]+\)/\1/' automake.texi
```

Если вы добавили имена файлов к переменной 'ETAGS_ARGS', то вы вероятно захотите установить переменную 'TAGS_DEPENDENCIES'. Содержимое этой переменной будет полностью добавлено к зависимости для цели tags.

Automake также сгенерирует цель ID, которая будет запускать программу mkid на исходных файлах. Это поддерживается при использовании покаталожной основы.

16.2 Обработка новых расширений файлов

Иногда полезно ввести новое неявное правило для обработки новых типов файлов, о которых Automake ничего не знает. Если вы сделали это, то вы должны уведомить GNU Make о новых суффиксах. Это может быть сделано помещением списка новых суффиксов в переменную SUFFIXES.

Например, в настоящее время Automake не обеспечивает никакой поддержки Java. Если вы напишите макрос для генерации файлов '.class' из файлов с исходными текстами '.java', то вы также должны добавить эти суффиксы в список.

```
SUFFIXES = .java .class
```

17 Включение дополнительных файлов

Для включения других файлов (может быть для подключения общих правил) поддерживается следующий синтаксис:

```
include ($(srcdir)|$(top_srcdir))/filename
```

Используя файлы в текущем каталоге:

```
include $(srcdir)/Makefile.extra
```

```
include Makefile.generated
```

Используя файлы в каталоге верхнего уровня:

```
include $(top_srcdir)/filename
```

18 Условные операторы

Automake поддерживает простой тип условных операторов.

До использования условного оператора, вы должны определить его в файле `configure.in` используя макрос `AM_CONDITIONAL` (см. Раздел 5.4 [Макросы], с. 15). Макросу `AM_CONDITIONAL` передается два аргумента.

Первым аргументом для `AM_CONDITIONAL` является имя условного оператора. Им должны быть простая строка, начинающаяся с буквы и содержащая только буквы, цифры и знаки подчеркивания.

Вторым аргументом `AM_CONDITIONAL` является условие для командного процессора, которое можно использовать в операторе `if`. Условие оценивается при запуске `configure`.

Условные операторы обычно зависят от ключей, которые использует пользователь при запуске скрипта `configure`. Вот пример того, как написать условный оператор, который возвращает истинное выражение, если пользователь использовал ключ '`-enable-debug`'.

```
AC_ARG_ENABLE(debug,
[  -enable-debug    Turn on debugging],
[case "${enableval}" in
  yes) debug=true ;;
  no)  debug=false ;;
  *) AC_MSG_ERROR(bad value ${enableval} for -enable-debug) ;;
esac],[debug=false])
AM_CONDITIONAL(DEBUG, test x$debug = xtrue)
```

Вот пример использования этого условного оператора в файле '`Makefile.am`':

```
if DEBUG
DBG = debug
else
DBG =
endif
noinst_PROGRAMS = $(DBG)
```

Этот тривиальный пример также мог быть создан используя макрос EXTRA_PROGRAMS (см. Раздел 7.1 [Программа], с. 18).

В операторе `if` вы можете тестировать только одну переменную. Оператор `else` может не использоваться. Условные операторы могут быть вложены на любую глубину.

Заметьте, что условные операторы в Automake не похожи на условные операторы в GNU Make. Условные операторы Automake проверяются во время конфигурации, при выполнении скрипта '`configure`', и воздействуют на преобразование файла '`Makefile.in`' в файл '`Makefile`'. Они основываются на ключах, передаваемых скрипту '`configure`' и на результатах, определяемых во время выполнения '`configure`'. Условные операторы GNU Make проверяются при выполнении `make` и основываются на переменных, передаваемых программе make, или определенных в '`Makefile`'.

Условные операторы Automake будут работать с любой программой make.

19 Эффект использования ключей -gnu и -gnits

Ключ '-gnu' (или 'gnu' в переменной 'AUTOMAKE_OPTIONS') заставляет automake выполнить проверку следующих вещей:

- Файлы 'INSTALL', 'NEWS', 'README', 'COPYING', 'AUTHORS' и 'ChangeLog' должны находится в каталоге верхнего уровня пакета.
- Ключи 'no-installman' и 'no-installinfo' запрещены.

Заметьте, что в будущем этот ключ будет расширен для проведения дополнительных проверок; рекомендуется ознакомиться с точными требованиями стандартов GNU. Также ключ '-gnu' может требовать наличия нестандартных программ GNU для использования в целях, используемых человеком, который сопровождает данный пакет; например, в будущем может потребоваться программа pathchk для работы цели 'make dist'.

Ключ '-gnits' делает то же самое, что и ключ '-gnu', а также проверяет следующие вещи:

- 'make dist' выполнит проверку того, что файл 'NEWS' обновлен для новой версии.
- Наличие файла 'COPYING.LIB' запрещено. По видимому LGPL считается несостоявшимся экспериментом.
- Проверяется 'VERSION' на то, что его формат соответствует стандартам Gnits.
- Если 'VERSION' указывает на то, что этот выпуск является альфа-версией, и в каталоге верхнего уровня находится файл 'README-alpha', то этот файл будет включен в дистрибутив. Это делается в режиме '-gnits', и ни в каких других режимах, поскольку только в этом режиме есть ограничения формата номера версии, и только в этом режиме Automake может автоматически определить, нужно ли включать файл 'README-alpha'.
- Требуется наличие файла 'THANKS'.

20 Эффект использования ключа -cygnus

Cygnus Solutions применяет немного другие правила о том как будет конструироваться файл 'Makefile.in'. Передача ключа '-cygnus' для automake вызовет то, что сгенерированный 'Makefile.in' будет соответствовать правилам Cygnus.

Вот точное описания действия ключа '-cygnus':

- Файлы Info всегда создаются в каталоге, где происходит построение, а не в каталоге с исходными текстами.
- Не требуется наличие файла 'texinfo.tex', если есть файлы с исходными текстами Texinfo. Предполагается, что файл будет предоставлен, но в том месте, где Automake не сможет найти его. Это предположение является следствием того, как обычно поставляются пакеты Cygnus.
- При выполнении 'make dist' файлы будут искаться в каталоге, где происходило построение, а также в каталоге с исходными текстами. Это требуется для того, чтобы файлы info были помещены в дистрибутив.

- Поиск некоторые утилиты будет производится в каталогах. где происходит сборка, а также в каталогах, указанных в переменной среды ‘PATH’ пользователя. Этими утилитами являются runtest, expect, makeinfo и texi2dvi.
- Подразумевается использование ключа --foreign.
- Подразумевается использование ключей ‘no-installinfo’ и ‘no-dependencies’.
- Требуются макросы ‘AM_MAINTAINER_MODE’ и ‘AM_CYGWIN32’.
- Цель check не зависит от цели all.

Люди, сопровождающие программы GNU рекомендуют использовать уровень ограничений ‘gnu’, а не режим Cygnus.

21 Когда не хватает возможностей Automake

Неявная семантика копирования Automake означает, что много проблем может быть решено простым добавлением некоторых целей для make и правил для ‘Makefile.in’. Automake будет игнорировать эти добавления.

Есть некоторые предостережения для этих работ. Хотя вы можете переопределить цели, уже используемые Automake, но часто это просто неразумно, особенно в каталоге верхнего уровня пакета не относящегося к типу flat. Однако, вы можете указать в вашем файле ‘Makefile.in’ различные полезные цели, имеющие суффикс ‘-local’. Automake дополнит стандартные цели этими целями пользователя.

К целям, поддерживающим локальную версию относятся: all, info, dvi, check, install-data, install-exec, uninstall и разные цели clean (mostlyclean, clean, distclean и maintainer-clean). Заметьте, что в этом списке нет целей uninstall-exec-local или uninstall-data-local; есть только uninstall-local. Это не имеет значения для удаления только данных или исполняемых файлов.

Например, вот один из способов установить файл в каталог ‘/etc’:

```
install-data-local:
        $(INSTALL_DATA) $(srcdir)/afile /etc/afile
```

Некоторые цели также имеют способ запускать другие цели после выполнения всех своих действий, это называется *ловушка (hook)*. Ловушка называется по имени цели, с добавлением суффикса ‘-hook’. Целями, разрешающими использование ловушек являются install-data, install-exec, dist и distcheck.

Например, вот как создать жесткую ссылку на установленную программу:

```
install-exec-hook:
        ln $(bindir)/program $(bindir)/proglink
```

22 Распространение файлов ‘Makefile.in’

Automake не налагает никакие ограничения на распространение готовых файлов ‘Makefile.in’. Мы поощряем авторов к распространению их работ под действием правил подобных GPL, но не требуем использовать Automake.

Некоторые из файлов, которые могут быть автоматически установлены при использовании ключа командной строки --add-missing могут вызвать ошибку при использовании GPL; просмотрите каждый файл для получения информации.

23 Некоторые идеи на будущее

Возможности, которые могут появиться в будущем:

- Поддержка HTML.
- Вывод будет очищен. Например, в файл '`Makefile.in`' будут вставляться только используемые переменные.
- Будет добавлена поддержка для автоматической перекодировки дистрибутива. Эта возможность предназначена для того, чтобы сопровождающий мог использовать удобный для него набор символов, а дистрибутив использовал бы Unicode или ISO 10646 с кодировкой UTF-8.
- Изменения в Guile. Этого не случится в ближайшем будущем, но когда-нибудь обязательно произойдет.

Индекс переменных и макросов

default

A

B

C

D

E

F

H

I

L

M

N

O

P

R

S

T

V

W

Y

Общий индекс

default

A

B

C

D

E

F

G

H

I

J

L

M

N

P

R

S

T

U

V

Y

А

Б

Г

Д

Е

З

И

К

М

Н

О

П

Р

С

Т

У

Ф

Ц

Оглавление