Autoconf

Создание скриптов для автоматической конфигурации
Редакция 2.13, Autoconf версии 2.13
Декабрь 1998

David MacKenzie и Ben Elliston

Copyright © 1992, 93, 94, 95, 96, 98 Free Software Foundation, Inc.

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions, except that this permission notice may be stated in a translation approved by the Foundation.

Перевод Отт Алексей © 2000.

Разрешается создавать и распространять не измененные копии этого руководства, сохраняя во всех копиях уведомление об авторских правах и данное уведомление.

Разрешается копировать и распространять измененные версии этого руководства на условиях, применяемых для не измененных копий, если результирующая работа целиком распространяется на условиях, идентичных данным условиям.

Разрешается копировать и распространять переводы этого руководства на другие языки, на условиях, приведенных для измененных версий этого руководства, за исключением уведомления о правах, которое может быть приведено в переводе, утвержденном Фондом.

1 Введение

> Физик, инженер и специалист по компьютерам спорили о природе Бога. Конечно, он был Физиком, сказал физик, поскольку в начале Творения Бог создал Свет; и вы знаете, есть уравнения Максвелла, двойная природа электромагнитных волн, уравнения относительности. . . "Он был Инженером!" – сказал инженер, поскольку до создания света, Бог разделил Хаос на Землю и Воду; нужно быть настоящим инженером, чтобы обработать огромное количество грязи и последовательно разделить твердые вещества от жидких. . . Компьютерный специалист воскликнул: А как вы думаете, откуда взялся хаос?
>
> —Anonymous

Autoconf— это утилита для создания скриптов командного процессора, которые автоматически конфигурируют пакеты с исходным кодом так, чтобы они могли работать на множестве UNIX-подобных систем. Скрипты настройки, созданные с помощью Autoconf, при выполнении не зависят от Autoconf, так что Autoconf не обязательно должен быть установлен у пользователя.

Скрипты конфигурации, созданные Autoconf, при работе не требуют вмешательства пользователя; обычно они даже не требуют, чтобы были заданы аргументы, указывающие тип системы. Вместо этого, такие скрипты тестируют наличие каждого средства, которое может понадобиться данному пакету. (До выполнения каждой из проверок, скрипты печатают однострочное сообщение о том, какую именно возможность они проверяют, так что пользователь не будет сильно скучать, ожидая конца работы скриптов). В результате эти скрипты хорошо справляются с системами, которые являются гибридами или специализированными вариантами большинства видов UNIX. Таким образом, пропадает необходимость в сопровождении файлов со списком всех возможностей всех версий каждого варианта UNIX.

Для каждого пакета программного обеспечения, который использует Autoconf, из шаблона создается скрипт настройки, который перечисляет системные возможности, в которых нуждается данный пакет или которые он может использовать. После того, как код на языке командного процессора, распознающий и обрабатывающий ту или иную возможность, написан, Autoconf позволяет использовать этот код во всех пакетах, которые могу использовать (или нуждаются) в соответствующей возможности. Если позже по каким-то причинам понадобится изменить код командного процессора, то изменения необходимо будет внести только в одно место; все скрипты настройки могут быть автоматически пересозданы, чтобы отразить изменения кода.

Пакет Metaconfig предназначен практически для тех же целей, что и Autoconf, но создаваемые скрипты требуют интерактивного взаимодействия с пользователем, что довольно неудобно при конфигурации больших программных проектов. В отличие от скриптов Metaconfig, скрипты Autoconf могут поддерживать кросс-компиляцию, если кто-то позаботится о ее поддержке на данной системе.

Существует несколько разных задач, относящихся к созданию переносимого программного обеспечения, которые в настоящее время нельзя решить средствами Autoconf. Среди них— автоматическое создание файлов ‘`Makefile`’ со всеми необходимыми

стандартными целями, а также предоставление замены стандартных библиотечных функций и заголовочных файлов в тех системах, в которых эти функции или файлы отсутствуют. Однако работа в этом настроении ведется и эти возможности могут появиться в будущих версиях.

Autoconf навязывает некоторые ограничения на имена макросов, которые используются в директивах `#ifdef` программ на языке C (см. [Индекс символов препроцессора], с. 100).

Для создания скриптов Autoconf требует наличия программы GNU `m4`. Он пользуется возможностями, которых нет в некоторых UNIX-версиях программы `m4`. Он также превышает внутренние ограничения некоторых версий `m4`, включая GNU `m4` версии 1.0. Вам необходимо использовать версию 1.1 (или более позднюю) программы GNU `m4`. Версии 1.3 и более поздние будут работать гораздо быстрее, чем версии 1.1 или 1.2.

См. Глава 13 [Обновление], с. 86, где описано обновление с версии 1. См. Глава 14 [История], с. 90, где описана история разработки Autoconf. См. Глава 12 [Вопросы], с. 83, где даются ответы на некоторые общие вопросы об Autoconf.

Сообщения об ошибках и свои пожелания для Autoconf посылайте по адресу `bug-gnu-utils@prep.ai.mit.edu`. Включите в сообщение номер версии Autoconf, который вы можете получить, выполнив команду ‘`autoconf -version`’.

2 Создание скриптов configure

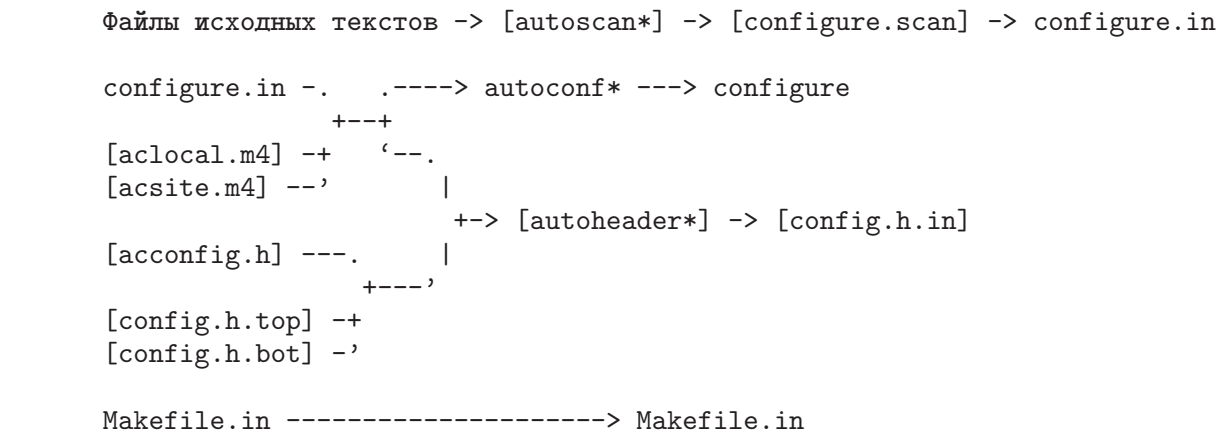
Скрипты конфигурации, создаваемые Autoconf, по принятым соглашениям называются `configure`. При запуске `configure` создает несколько файлов, заменяя в них параметры конфигурации на соответствующие системе значения. `configure` создает следующие файлы:

- один или несколько файлов '`Makefile`', по одному на каждый подкаталог пакета (см. Раздел 3.3 [Подстановки в Makefile], с. 12);
- если задано, создается заголовочный файл для языка C, имя которого можно задать при создании скрипта и который содержит директивы `#define` (см. Раздел 3.4 [Заголовочные файлы конфигурации], с. 16);
- скрипт командного процессора с именем '`config.status`', который при запуске заново создаст вышеперечисленные файлы (см. Глава 11 [Запуск config.status], с. 81);
- скрипт командного процессора с именем '`config.cache`', который сохраняет результаты выполнения многих тестов (см. Раздел 6.3.2 [Кэш-файлы], с. 58);
- файл с именем '`config.log`', который содержит все сообщения, выданные компиляторами. Этот файл может использоваться при отладке, если `configure` работает неправильно.

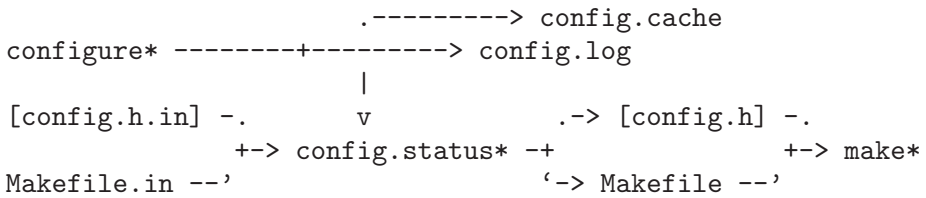
Для того, чтобы с помощью Autoconf создать скрипт `configure`, вам необходимо написать входной файл с именем '`configure.in`' и выполнить команду `autoconf`. Если вы напишите собственный код тестирования возможностей системы, в дополнение к поставляемым с Autoconf, то вам придется записать его в файлы с именами '`aclocal.m4`' и '`acsite.m4`'. Если вы используете заголовочный файл, который содержит директивы `#define`, то вы также должны создать файл '`acconfig.h`', и вы сможете распространять с пакетом созданный с помощью Autoconf файл '`config.h.in`'.

Вот диаграмма, показывающая, как создаются файлы, используемые при конфигурации. Выполняемые программы обозначены суффиксом '`*`'. Необязательные файлы взяты в квадратные скобки ('`[]`'). Программы `autoconf` и `autoheader` также читают установленные файлы с макросами Autoconf (обрабатывая файл '`autoconf.m4`').

Файлы, используемые при подготовке программного пакета к распространению:



Файлы, используемые при конфигурации программного пакета:



2.1 Написание ‘configure.in’

Для создания скрипта configure для программного пакета, создайте файл с именем ‘configure.in’, который содержит вызовы макросов Autoconf, которые проверяют системные возможности, которые нужны вашему пакету или которые он может использовать. Для многих таких возможностей макросы Autoconf уже написаны; См. Глава 4 [Существующие тесты], с. 22, где находится их описание. Для большинства других возможностей вы можете использовать шаблонные макросы Autoconf, на базе которых можно создать специальные проверки; См. Глава 5 [Написание тестов], с. 45, где это описано. Для особо хитроумных или специализированных возможностей, в файл ‘configure.in’ может понадобиться включить специально написанные скрипты командного процессора. Программа autoscan может оказать вам хорошую помощь на первых порах, при создании файла ‘configure.in’ (см. Раздел 2.2 [Запуск autoscan], с. 6, где описана эта программа).

За некоторыми исключениями, порядок вызовов макросов Autoconf в ‘configure.in’ не важен. Каждый файл ‘configure.in’ должен в самом начале содержать вызов макроса AC_INIT, а также вызов макроса AC_OUTPUT в самом конце (см. Раздел 3.2 [Вывод], с. 10). Также некоторые макросы полагаются на то, что другие макросы были вызваны первыми, поскольку для того, чтобы принять решение, они проверяют уже установленные значения переменных. Такие макросы отдельно отмечены в описании (см. Глава 4 [Существующие тесты], с. 22), а при создании скрипта configure выдается предупреждение, если вы нарушили порядок вызова макросов.

Для того, чтобы ваши файлы были последовательны и единообразны, мы приведем желательный порядок вызова макросов Autoconf. Вообще говоря, то, что находится ближе к концу списка, может зависеть от того, что находится в начале списка. Например, библиотечные функции могут зависеть от определений типов и библиотек.

AC_INIT(*file*)
Проверка программ
Проверка библиотек
Проверка заголовочных файлов
Проверка определений типов
Проверка структур
Проверка характеристик компилятора
Проверка библиотечных функций
Проверка системных сервисов
AC_OUTPUT([*file*...])

Лучше всего помещать каждый вызов макроса на отдельную строку файла ‘configure.in’. Большинство макросов не добавляют дополнительных переводов строк; они полагают, что после каждого вызова макроса находится новая строка. Это

позволяет сделать скрипт configure читабельнее, не добавляя ненужных пустых строк. Можно спокойно устанавливать переменные окружения в той же строке, что и вызов макроса, потому что командные процессоры позволяют выполнять присваивание в одной строке, без дополнительных пустых строк.

При вызове макросов с аргументами между открывающей скобкой и названием макроса не должно быть пробелов. Аргументы могут занимать несколько строк если они заключены в "кавычки" языка m4— '[' и ']'. Если у вас есть длинная строка, например, список имен файлов, то можно использовать символ обратного слэша в конце строки для указания, что список продолжается на следующей строке (эта возможность реализуется командным процессором, без привлечения возможностей Autoconf).

Некоторые макросы отрабатывают два случая— когда заданное условие выполняется и когда условие не выполняется. В некоторых местах вы можете захотеть сделать что-либо, если условие выполняется, и ничего не делать в противном случае, и наоборот. Для того, чтобы пропустить действие при выполнении условия, передайте пустое значение аргументу *action-if-found* данного макроса. Для пропуска действия при невыполнении условия уберите аргумент *action-if-not-found* данного макроса, включая предшествующую ему запятую.

В файл 'configure.in' можно включать комментарии, начиная их со встроенного макроса m4— dnl, который отбрасывает текст вплоть до начала новой строки. Эти комментарии не появятся в созданных скриптах configure. Например, полезно начинать файлы 'configure.in' со строки, которая может выглядеть так:

```
dnl для создания скрипта configure обработайте этот файл программой autoconf.
```

2.2 Использование программы autoscan для создания 'configure.in'

Программа autoscan может помочь вам в создании файла 'configure.in' для программного пакета. autoscan выполняет анализ дерева исходных текстов, корень которого указан в командной строке или совпадает с текущим каталогом. Программа ищет в исходных текстах следы обычных проблем с переносимостью и создает файл 'configure.scan', который является заготовкой для 'configure.in' для данного пакета.

Вы должны сами просмотреть файл 'configure.scan' перед тем, как переименовать его в 'configure.in': скорее всего, он будет нуждаться в некоторых исправлениях. Иногда autoscan выдает макросы в неправильном порядке, и поэтому autoconf будет выдавать предупреждения; вам необходимо вручную передвинуть эти макросы. Также, если вы хотите, чтобы пакет использовал заголовочный файл настроек, то вы должны сами добавить вызов макроса AC_CONFIG_HEADER (см. Раздел 3.4 [Заголовочные файлы конфигурации], с. 16). Вам также необходимо добавить или изменить некоторые директивы препроцессора #if в вашей программе, для того, чтобы заставить ее работать с Autoconf (см. Раздел 2.3 [Запуск ifnames], с. 7, где описана программа, которая поможет вам выполнить эту работу).

Программа autoscan использует несколько файлов данных, чтобы определить, какие макросы следует использовать при обнаружении определенных символов в исходных файлах пакета. Эти файлы данных устанавливаются вместе с дистрибутивными

макро-файлами Autoconf и имеют одинаковый формат. Каждая строка состоит из символа, пробелов и имени макроса Autoconf, которое выдается в том случае, если заданный символ имеется в исходных текстах. Строки, начинающиеся с символа '`#`' являются комментариями.

`autoscan` устанавливается только в том случае, если у вас установлена программа Perl. `autoscan` распознает следующие ключи командной строки:

`-help` Выдает список ключей командной строки и прекращает работу.

`-macrodir=`*dir*
: Заставляет программу искать файлы данных в каталоге *dir*, а не в каталоге, куда производилась установка. Вы также можете установить значение переменной окружения `AC_MACRODIR` равным пути к этому каталогу; данный ключ командной строки переопределяет значение переменной окружения.

`-verbose` Выдает имена исследуемых файлов и потенциально интересные символы, обнаруженные в этих файлах. Выдача может быть довольно обширной.

`-version` выдает номер версии Autoconf и прекращает работу.

2.3 Использование программы `ifnames` для перечисления условных выражений

Программа `ifnames` может помочь при создании файла '`configure.in`' для программного пакета. Она выдает список идентификаторов, которые пакет уже использует в условных выражениях препроцессора языка C. Если ваша программа уже была написана с учетом возможного переноса на другие платформы, то данная программа может помочь вам определить, какие проверки необходимо выполнить в `configure`. Эта программа может помочь заполнить некоторые пробелы в файле '`configure.in`', который был создан программой `autoscan` (см. Раздел 2.2 [Запуск autoscan], с. 6).

Программа `ifnames` обрабатывает все исходные тексты на C, перечисленные в командной строке (или же принимает текст со стандартного ввода, если ни один файл не был указан) и выдает на стандартный вывод сортированный список идентификаторов, которые используются в директивах `#if`, `#elif`, `#ifdef` или `#ifndef`. Программа выдает каждый идентификатор на отдельной строке, за которым через пробел следует список файлов, в которых этот идентификатор встречается.

Программа `ifnames` распознает следующие ключи командной строки:

`-help`
`-h` выдает список ключей командной строки и прекращает работу.

`-macrodir=`*dir*
: Заставляет программу искать файлы данных в каталоге *dir*, а не в каталоге, куда производилась установка. Вы также можете установить значение переменной окружения `AC_MACRODIR` равным пути к этому каталогу; данный ключ командной строки переопределяет значение переменной окружения.

`-version` выдает номер версии Autoconf и прекращает работу.

2.4 Использование программы autoconf для создания скрипта configure

Для того, чтобы создать скрипт `configure` из файла '`configure.in`', просто запустите программу `autoconf` без аргументов. `autoconf` обработает файл '`configure.in`' с помощью макропроцессора `m4`, используя макросы Autoconf. Если вы зададите аргумент программе `autoconf`, то программа будет выполнять чтение заданного файла, а не файла '`configure.in`' и вывод будет производиться на стандартный вывод, не в в файл `configure`. Если вы дадите программе `autoconf` аргумент '`-`', то она будет читать со стандартного ввода, а не из файла '`configure.in`', а результаты будут выдаваться на стандартный вывод.

Макросы Autoconf определены в нескольких файлах. Некоторые из них распространяются вместе с Autoconf; `autoconf` читает их в первую очередь. Затем ищется необязательный файл '`acsite.m4`' в каталоге, который содержит распространяемые с Autoconf файлы макросов, и необязательный файл '`aclocal.m4`' в текущем каталоге. Эти файлы могут содержать макросы, специфические для вашей машины или макросы для конкретных пакетов программного обеспечения (см. Глава 7 [Создание макросов], с. 62, где приведена дополнительная информация). Если определение макроса существует в нескольких файлах, которые считывает `autoconf`, то последнее макроопределение переопределяет все предыдущие.

`autoconf` распознает следующие ключи командной строки:

`-help`
`-h` выдает список ключей командной строки и прекращает работу.

`-localdir=`*dir*
`-l` *dir* Ищет файл '`aclocal.m4`' для данного пакета в каталоге *dir*, а не в текущем каталоге.

`-macrodir=`*dir*
Заставляет программу искать файлы данных в каталоге *dir*, а не в каталоге, куда производилась установка. Вы также можете установить значение переменной окружения `AC_MACRODIR` равным пути к этому каталогу; данный ключ командной строки переопределяет значение переменной окружения.

`-version` выдает номер версии Autoconf и прекращает работу.

2.5 Использование autoreconf для обновления ваших скриптов configure

Если у вас много скриптов `configure`, созданных с помощью Autoconf, то программа `autoreconf` может облегчить вашу работу. Она запускает программы `autoconf` (и, если необходимо, `autoheader`) для повторного создания скриптов `configure` и шаблонов заголовочных файлов настройки для исходных текстов, корневой каталог которых находится в текущем каталоге. По умолчанию, эти программы создают заново только те файлы, которые старше, чем соответствующий файл '`configure.in`' или (если имеется) '`aclocal.m4`'. Поскольку программа `autoheader` не изменяет время модификации выходного файла в случае, если файл не изменялся, то не обязательно будет проделано минимальное количество работы. Если вы установили новую версию Autoconf, то

вы можете заставить `autoreconf` заново создать *все* файлы, задав ключ командной строки '`-force`'.

Если вы зададите программе `autoreconf` ключи командной строки '`-macrodir=`*dir*' или '`-localdir=`*dir*', то она передаст их программам `autoconf` и `autoheader` (с правильно настроенными относительными путями).

`autoreconf` не поддерживает нахождение в одном дереве как каталогов, которые являются частями большого проекта (и которые используют одни и те же файлы '`aclocal.m4`' и '`acconfig.h`'), так и каталогов, которые являются независимыми пакетами (которые имеют собственные файлы '`aclocal.m4`' и '`acconfig.h`'). Программа предполагает, что все каталоги являются частями одного пакета, если вы используете ключ командной строки '`-localdir`', или что каждый каталог является отдельным пакетом, если вы не используете этот ключ. Это ограничение может быть убрано в будущем.

См. Раздел 3.3.3 [Автоматическая пересборка], с. 16, где описаны правила '`Makefile`' для автоматического пересоздания скриптов `configure`, если изменяются исходные тексты этих скриптов. Этот метод корректно обрабатывает изменение шаблонов заголовочных файлов конфигурации, но не передает команде ключи командной строки '`-macrodir=`*dir*' или '`-localdir=`*dir*'.

`autoreconf` распознает следующие ключи командной строки:

`-help`
`-h`
: выдает список ключей командной строки и прекращает работу.

`-force`
`-f`
: Пересоздать даже те скрипты '`configure`' и заголовочные файлы настройки, которые новее, чем соответствующие входные файлы ('`configure.in`' и, если есть, '`aclocal.m4`').

`-localdir=`*dir*
`-l` *dir*
: Заставляет программы `autoconf` и `autoheader` искать файлы '`aclocal.m4`' и (для `autoheader`) '`acconfig.h`' (но не '*file*`.top`' и '*file*`.bot`') данного пакета в каталоге *dir* вместо каталога, который содержит отдельный файл '`configure.in`'.

`-macrodir=`*dir*
`-m` *dir*
: Заставляет программу искать файлы данных в каталоге *dir*, а не в каталоге, куда производилась установка. Вы также можете установить значение переменной окружения `AC_MACRODIR` равным пути к этому каталогу; данный ключ командной строки переопределяет значение переменной окружения.

`-verbose`
: Выдает имена каждого каталога, в котором `autoreconf` запускает `autoconf` (и если необходимо то и `autoheader`).

`-version`
: Выдает номер версии Autoconf и прекращает работу.

3 Файлы инициализации и выходные файлы

Скриптам, созданным Autoconf, нужна некоторая инициализационная информация, как то: где найти исходные тексты пакета; какие выходные файлы создавать. В нижеследующей главе описана инициализация и создание выходных файлов.

3.1 Нахождение ввода `configure`

Каждый скрипт `configure` должен первым делом вызвать макрос `AC_INIT`. Единственный обязательный макрос – `AC_OUTPUT` (см. Раздел 3.2 [Вывод], с. 10).

AC_INIT (*unique-file-in-source-dir*) Macro

Обрабатывает аргументы командной строки и ищет каталог с исходными текстами. *unique-file-in-source-dir*— это некоторый файл в каталоге с исходными текстами пакета; `configure` проверяет существование этого файла, чтобы убедиться, что это именно тот каталог с исходными текстами, какой нужно. Иногда люди указывают неверный каталог с исходными текстами, используя ключ командной строки '`-srcdir`'; эта проверка позволяет не допускать таких инцидентов. Для детальной информации См. Глава 10 [Запуск configure], с. 77.

Пакетам, которые выполняют ручную настройку или используют программу `install`, может понадобиться указать скрипту `configure`, где можно найти другие скрипты командного процессора. Это выполняется с помощью вызова макроса `AC_CONFIG_AUX_DIR`, хотя используемые по умолчанию значения в большинстве случаев будут правильными.

AC_CONFIG_AUX_DIR(*dir*) Macro

Использует скрипты '`install-sh`', '`config.sub`', '`config.guess`' и Cygnus-версию `configure`, которые располагаются в каталоге *dir*. Эти вспомогательные файлы используются при конфигурировании. Значение *dir* может быть задано либо абсолютным путем, либо путем относительно '*srcdir*'. Значением по умолчанию является первый из каталогов '*srcdir*', '*srcdir*`/..`' или '*srcdir*`/../..`', в котором будет найден файл '`install-sh`'. Проверка наличия других файлов не производится, так что использование `AC_PROG_INSTALL` не требует включения в дистрибутив других вспомогательных файлов. Также проверяется наличие файла '`install.sh`', но это имя является устаревшим, поскольку некоторые программы `make` имеют правило, которое создает файл '`install`' из этого файла, в случае если '`Makefile`' отсутствует.

3.2 Создание выходных файлов

Каждый скрипт `configure`, созданный Autoconf, должен заканчиваться вызовом макроса `AC_OUTPUT`. Этот макрос создает файлы '`Makefile`' и, может быть, дополнительные файлы, которые являются результатом конфигурации. Еще одним обязательным макросом является `AC_INIT` (см. Раздел 3.1 [Ввод], с. 10).

AC_OUTPUT ([*file*... [, *extra-cmds* [, *init-cmds*]]]) Macro

Создает выходные файлы. Вызовите этот макрос один раз в конце файла ‘`configure.in`’. Аргумент *file*... является списком выходных файлов через пробел; этот список может быть пустым. Этот макрос создает каждый из файлов ‘*file*’, копируя входной файл (который по умолчанию называется ‘*file*`.in`’) и подставляя значения выходных переменных. Для более детального описания использования выходных переменных См. Раздел 3.3 [Подстановки в Makefile], с. 12. Для детального описания того, как создавать такие переменные См. Раздел 6.2 [Установка выходных переменных], с. 56. Этот макрос создает каталог, в котором находится файл, если этот каталог не существует (но не создает родительские каталоги для этого каталога). Обычно таким образом создаются файлы ‘`Makefile`’, но можно указать также и другие файлы, такие как ‘`.gdbinit`’.

Если вызывались макросы `AC_CONFIG_HEADER`, `AC_LINK_FILES` или `AC_CONFIG_SUBDIRS`, то этот макрос также создает файлы, указанные в аргументах этих макросов.

Типичный вызов `AC_OUTPUT` выглядит примерно так:

```
AC_OUTPUT(Makefile src/Makefile man/Makefile X/Imakefile)
```

Вы можете переопределить имена входных файлов, добавив к *file* список входных файлов, который разделен двоеточием. Например:

```
AC_OUTPUT(Makefile:templates/top.mk lib/Makefile:templates/lib.mk)
AC_OUTPUT(Makefile:templates/vars.mk:Makefile.in:templates/rules.mk)
```

Это позволит вам сохранить имена файлов в формате MS-DOS, или для добавления стандартных кусков кода кода в начало или конец файла.

В параметре *extra-cmds* можно указать команды, которые будут вставлены в файл ‘`config.status`’ и сработают после того, как было сделано все остальное. В параметре *init-cmds* можно указать команды, которые будут вставлены непосредственно перед *extra-cmds*, причем `configure` выполнит в них подстановку переменных, команды и обратных слэшей. Аргумент *init-cmds* можно использовать для передачи переменных из `configure` в *extra-cmds*. Если был вызван макрос `AC_OUTPUT_COMMANDS`, то команды, переданные ему в качестве аргумента, выполняются прямо перед командами, переданными макросу `AC_OUTPUT`.

AC_OUTPUT_COMMANDS (*extra-cmds* [, *init-cmds*]) Macro

Задает дополнительные команды командного процессора, которые выполняются в конце ‘`config.status`’, а также команды для инициализации переменных в `configure`. Этот макрос можно вызывать несколько раз. Вот нереальный пример:

```
fubar=27
AC_OUTPUT_COMMANDS([echo this is extra $fubar, and so on.], fubar=$fubar)
AC_OUTPUT_COMMANDS([echo this is another, extra, bit], [echo init bit])
```

Если вам нужно запустить `make` в подкаталогах, то это следует делать с помощью переменной `MAKE`. Большинство версий программы `make` устанавливают значение переменной `MAKE` равным имени программы `make` с дополнительно заданными ключами.

(Но многие версии не включаются сюда значения переменных, заданных в командной строке, поэтому они не передаются автоматически). Некоторые старые версии команды `make` не устанавливают эту переменную. Следующий макрос позволяет вам использовать переменную `MAKE` даже таких старых версий.

AC_PROG_MAKE_SET Macro

Если `make` определяет переменную `MAKE`, то переменная `SET_MAKE` получает пустое значение. Иначе, определяется переменная `SET_MAKE` со значением '`MAKE=make`'. Для переменной `SET_MAKE` вызывается макрос `AC_SUBST`.

Для использования данного макроса поместите следующую строку в каждый из файлов '`Makefile.in`', в котором производится запуск `MAKE` для подкаталогов:

```
@SET_MAKE@
```

3.3 Подстановки в файлах Makefile

Каждый подкаталог дистрибутива, который содержит что либо, что должно компилироваться или устанавливаться, должен поставляться с файлом '`Makefile.in`', из которого `configure` создаст файл '`Makefile`' для данного каталога. Для создания '`Makefile`', `configure` выполнит простую подстановку переменных, заменяя вхождения '`@`*variable*`@`' в файле '`Makefile.in`' на значения, которые определены `configure` для данной переменной. Переменные, которые подставляются в выходных файлах таким способом, называются *выходными переменными (output variables)*. Они являются обычными переменными командного процессора, которые устанавливаются в `configure`. Для того, чтобы `configure` подставлял в выходных файлах определенную переменную, необходимо вызвать макрос `AC_SUBST` с именем переменной в качестве аргумента. Любое вхождение '`@`*variable*`@`' для других переменных остается неизменным. Для получения дополнительной информации о создании выходных переменных с помощью макроса `AC_SUBST` См. Раздел 6.2 [Установка выходных переменных], с. 56.

Пакеты программного обеспечения, использующие скрипт `configure`, должны распространяться с файлами '`Makefile.in`', но не с файлами '`Makefile`'; таким образом, пользователь должен перед компиляцией сконфигурировать программный пакет так, чтобы он соответствовал используемой системе.

См. раздел "Makefile Conventions" в *The GNU Coding Standards*, для получения информации о том, что можно помещать в файлы '`Makefile`'.

3.3.1 Предварительная установка выходных переменных

Некоторые выходные переменные заранее устанавливаются макросами Autoconf. Некоторые макросы Autoconf устанавливают дополнительные выходные переменные, которые упомянуты в описаниях этих макросов. Полный список выходных переменных находится в [Индекс выходных переменных], с. 98. См. раздел "Variables for Installation Directories" в *The GNU Coding Standards*, для дополнительной информации о переменных с именами, которые заканчиваются на '`dir`'.

bindir Variable

Каталог в который устанавливаются исполняемые программы, которые запускают пользователи.

configure_input Variable

Комментарий, сообщающий что файл был автоматически создан скриптом `configure` и выдает имя входного файла. `AC_OUTPUT` добавляет строку комментария, содержащую эту переменную, в начало каждого создаваемого файла '`Makefile`'. Для других файлов вы должны сослаться на эту переменную в комментарии в заголовке каждого входного файла. Например, входной скрипт командного процессора должен начинаться примерно так:

```
#! /bin/sh
# @configure_input@
```

Наличие этой строки также напоминает людям, редактирующим этот файл, что перед использованием его необходимо обработать с помощью `configure`.

datadir Variable

Каталог, куда устанавливаются файлы данных, не зависящие от архитектуры.

exec_prefix Variable

Префикс указывающий, куда будут устанавливаться файлы, зависящие от архитектуры.

includedir Variable

Каталог, куда будут устанавливаться заголовочные файлы C.

infodir Variable

Каталог, куда будет устанавливаться документация в формате Info.

libdir Variable

Каталог куда будут устанавливаться скомпилированные библиотеки.

libexecdir Variable

Каталог, в который устанавливаются исполняемые файлы, запускаемые другими программами.

localstatedir Variable

Каталог, куда будут устанавливаться изменяемые файлы данных для данной машины.

mandir Variable

Каталог верхнего уровня, в который будут устанавливаться страницы руководства.

oldincludedir Variable

Каталог, куда будут устанавливаться заголовочные файлы, для не-gcc компиляторов.

prefix Variable

Префикс для установки файлов, не зависящих от архитектуры.

sbindir Variable
Каталог для установки исполняемых файлов, запускаемых администратором.

sharedstatedir Variable
Каталог в который устанавливаются изменяемые, независящие от архитектуры файлы данных.

srcdir Variable
Каталог, который содержит исходный код для данного ‘`Makefile`’.

sysconfdir Variable
Каталог, в который устанавливаются неизменяемые файлы данных для данной машины.

top_srcdir Variable
Каталог верхнего уровня, содержащий исходный код пакета. В каталоге верхнего уровня эта переменная совпадает с `srcdir`.

CFLAGS Variable
Ключи оптимизации и отладочной информации для компилятора C. Если эта переменная не установлена в среде при запуске `configure`, то значение по умолчанию устанавливается при вызове макроса `AC_PROG_CC` (или равно пустому значению, если вы не вызываете этот макрос). `configure` использует эту переменную при компиляции программ для тестирования возможностей компилятора C.

CPPFLAGS Variable
Каталоги поиска заголовочных файлов (‘`-I`*dir*’) и любые другие ключи для препроцессора и компилятора C. Если эта переменная не установлена в среде при запуске `configure`, то значение по умолчанию равно пустому значению. `configure` использует эту переменную при компиляции программ или обработке препроцессором для тестирования возможностей компилятора C.

CXXFLAGS Variable
Ключи оптимизации и отладочной информации для компилятора C++. Если эта переменная не установлена в среде при запуске `configure`, то значение по умолчанию устанавливается при вызове макроса `AC_PROG_CXX` (или равно пустому значению, если вы не вызываете этот макрос). `configure` использует эту переменную при компиляции программ для тестирования возможностей компилятора C++.

FFLAGS Variable
Ключи оптимизации и отладочной информации для компилятора Fortran 77. Если эта переменная не установлена в среде при запуске `configure`, то значение по умолчанию устанавливается при вызове макроса `AC_PROG_F77` (или равно пустому значению, если вы не вызываете этот макрос). `configure` использует эту переменную при компиляции программ для тестирования возможностей компилятора Fortran 77.

DEFS Variable

Ключи ‘-D’, передаваемые компилятору C. Если вызывается макрос AC_CONFIG_HEADER, то configure заменяет вхождения ‘@DEFS@’ на ‘-DHAVE_CONFIG_H’ (см. Раздел 3.4 [Заголовочные файлы конфигурации], с. 16). Эта переменная не определена во время выполнения тестов configure, она определяется только при создании выходных файлов. См. Раздел 6.2 [Установка выходных переменных], с. 56, для описания того, как получить результаты предыдущих тестов.

LDFLAGS Variable

Ключ для удаления отладочной информации (‘-s’), а также другие ключи для компоновщика. Если не установлена в среде при запуске configure, то по умолчанию имеет пустое значение. configure использует эту переменную при компоновке программ для тестирования возможностей C.

LIBS Variable

Ключи ‘-l’ и ‘-L’, передаваемые компоновщику.

3.3.2 Каталоги сборки программ

Вы можете поддерживать одновременную компиляцию пакета программного обеспечения для различных архитектур из одной и той же копии исходного кода. Объектные файлы для каждой из архитектур хранятся в отдельных каталогах.

Для поддержки этого make использует переменную VPATH для поиска файлов, которые находятся в каталоге с исходными текстами. Такая возможность поддерживается GNU make и большинством других программ make свежих версий. Старые версии программы make не поддерживают переменную VPATH; при их использовании исходные тексты должны находиться в том же каталоге, что и объектные файлы.

Для поддержки VPATH каждый файл ‘Makefile.in’ должен содержать две строки, которые могут выглядеть следующим образом:

```
srcdir = @srcdir@
VPATH = @srcdir@
```

Не надо устанавливать VPATH в значение другой переменной, например ‘VPATH = $(srcdir)’, поскольку некоторые версии make не выполняют подстановку переменных для VPATH.

configure подставляет правильное значение srcdir при создании файла ‘Makefile’.

Не используйте переменную $< программы make, которая разворачивается в имя файла с полным путем (найденным с помощью VPATH), причем только в явных правилах. (Неявные правила, например, ‘.c.o’, сообщают, как создать файл ‘.o’ из файла ‘.c’.) Некоторые версии make не устанавливают $< в явных правилах; они подставляют вместо него пустое значение.

Вместо этого командные строки‘Makefile’ всегда должны ссылаться на файлы с исходными текстами, с добавлением к ним префикса ‘$(srcdir)/’. Например:

```
time.info: time.texinfo
        $(MAKEINFO) $(srcdir)/time.texinfo
```

3.3.3 Автоматическая пересборка

Вы можете поместить правила, упомянутые ниже, в файл ‘Makefile.in’ верхнего уровня пакета, для того чтобы автоматически обновлять информацию о конфигурации при изменении файлов конфигурации. Этот пример использует все дополнительные файлы, такие как ‘aclocal.m4’, а также то, что относятся к заголовочным файлам конфигурации. Уберите из правила для ‘Makefile.in’ файлы, не использующиеся в вашем пакете.

Префикс ‘${srcdir}/’ добавлен из-за ограничений механизма использования VPATH.

Файлы ‘stamp-’ являются необходимыми, поскольку время последнего изменения файлов ‘config.h.in’ и ‘config.h’ останется прежним, если пересоздание этих файлов не изменит их содержимого. Эта возможность позволяет избежать ненужной перекомпиляции. Вы должны включить файл ‘stamp-h.in’ в дистрибутив вашего пакета, так что make будет считать ‘config.h.in’ обновленным. На некоторых старых системах BSD, команда touch или любая другая, создающая файл нулевой длины, не обновляет время изменения этого файла, так что используйте для правильной работы команду echo.

```
${srcdir}/configure: configure.in aclocal.m4
        cd ${srcdir} && autoconf

# autoheader мог не изменить config.h.in, так что обновить дату stamp-файла.
${srcdir}/config.h.in: stamp-h.in
${srcdir}/stamp-h.in: configure.in aclocal.m4 acconfig.h \
    config.h.top config.h.bot
        cd ${srcdir} && autoheader
        echo timestamp > ${srcdir}/stamp-h.in

config.h: stamp-h
stamp-h: config.h.in config.status
        ./config.status

Makefile: Makefile.in config.status
        ./config.status

config.status: configure
        ./config.status -recheck
```

Вдобавок вы должны передать ‘echo timestamp > stamp-h’ в аргументе *extra-cmds* макросу AC_OUTPUT, так что ‘config.status’ будет гарантировать, что файл ‘config.h’ будет рассматриваться как обновленный. См. Раздел 3.2 [Вывод], с. 10, для дополнительной информации о AC_OUTPUT.

См. Глава 11 [Запуск config.status], с. 81, где описаны дополнительные примеры обработки конфигурационных зависимостей.

3.4 Заголовочные файлы конфигурации

Когда пакет производит тестирование большого количества символов препроцессора C, то командная строка ключей ‘-D’, передаваемых компилятору, может получиться

достаточно длинной. Это вызывает две проблемы. Первая заключается в том, что вывод результатов команды `make` будет тяжело читать в поисках ошибок. Вторая и более серьезная заключается в том, что длина командной строки может превысить предельную длину командной строки в некоторых операционных системах. В качестве альтернативы передаче компилятору ключей '`-D`', скрипты `configure` могут создавать заголовочные файлы C, которые содержат директивы '`#define`'. Макрос `AC_CONFIG_HEADER` выбирает этот способ выдачи результатов. Макрос должен быть вызван сразу после вызова `AC_INIT`.

Пакет должен подключить с помощью `#include` заголовочный файл настройки до подключения остальных заголовочных файлов, чтобы избежать несовместимости в объявлениях (например, если этот файл переопределяет `const`). Используйте директиву '`#include <config.h>`' вместо '`#include "config.h"`', и передайте компилятору C ключ '`-I.`' (или '`-I..`', смотря где находится файл '`config.h`'). Таким образом, даже если сам каталог с исходными текстами сконфигурирован (например, для создания дистрибутива), то другие сборочные каталоги можно будет сконфигурировать, не используя при этом файл '`config.h`' и каталога с исходными текстами.

AC_CONFIG_HEADER (*header-to-create ...*) Macro

Заставляет `AC_OUTPUT` создать файлы с именами из разделенного пробелами списка *header-to-create*, которые будут содержать директивы `#define` препроцессора C, и заменить '`@DEFS@`' в созданных файлах на '`-DHAVE_CONFIG_H`' вместо значения `DEFS`. Обычным значением для *header-to-create* является '`config.h`'.

Если *header-to-create* уже существует и его содержимое не отличается от того, что в него хотят поместить, то он остается неизмененным. Это позволяет вносить некоторые изменения в конфигурацию без ненужной перекомпиляции объектных файлов, которые зависят от данных заголовочных файлов.

Обычно входной файл называется '*header-to-create*`.in`'; однако вы можете переопределить имя входного файла, добавив к *header-to-create* список входных файлов, разделенный двоеточием. Примеры:

```
AC_CONFIG_HEADER(defines.h:defines.hin)
AC_CONFIG_HEADER(defines.h:defs.pre:defines.h.in:defs.post)
```

Это позволяет вам сохранить имена в виде, приемлемом для использования в MS-DOS, а также для добавления стандартных кусков кода к файлам.

3.4.1 Шаблоны заголовочных файлов

Ваш дистрибутив должен содержать файл шаблона, который должен выглядеть так, как будет выглядеть окончательный заголовочный файл, включая комментарии, но при этом все значения директив `#define` в нем будут установлены по умолчанию. Например, предположим, что ваш файл '`configure.in`' производит следующие вызовы:

```
AC_CONFIG_HEADER(conf.h)
AC_CHECK_HEADERS(unistd.h)
```

Для этого примера необходимо вставить в '`conf.h.in`' нижеследующий код. В системах, в которых есть '`unistd.h`', `configure` заменит 0 на 1. В других системах эта строка останется неизмененной.

```
/* Определить со значением 1 если у вас есть unistd.h.  */
#define HAVE_UNISTD_H 0
```

Если ваш код проверяет конфигурацию, используя директиву препроцессора `#ifdef` вместо `#if`, то значение по умолчанию может быть удалено директивой `#undef` вместо определения значения. В системах в которых имеется файл '`unistd.h`', `configure` изменит вторую строку на '`#define HAVE_UNISTD_H 1`'. В других системах эта строка будет закомментирована (в случае, если система предопределяет этот символ).

```
/* Определяется, если в системе есть unistd.h.  */
#undef HAVE_UNISTD_H
```

3.4.2 Использование `autoheader` для создания '`config.h.in`'

Программа `autoheader` может создать шаблон файла, содержащего директивы '`#define`', для использования с `configure`. Если '`configure.in`' использует `AC_CONFIG_HEADER(`*file*`)`, то `autoheader` создает '*file*`.in`'; если в качестве аргумента задано несколько имен файлов, то используется только первое имя. В противном случае `autoheader` создаст файл '`config.h.in`'.

Если вы зададите аргумент программе `autoheader`, то она будет считывать данные из этого файла, а не из файла '`configure.in`' и будет выводить данные в поток стандартного вывода вместо '`config.h.in`'. Если вы зададите `autoheader` аргумент '`-`', то программа будет считывать данные со стандартного ввода вместо '`configure.in`' и выдавать результаты на стандартный вывод.

`autoheader` сканирует файл '`configure.in`' и определяет, какие символы препроцессора C могут быть определены в нем. Программа копирует комментарии и директивы `#define` и `#undef` из файла с именем '`acconfig.h`', который поставляется вместе с Autoconf. Программа также использует файл с именем '`acconfig.h`' из текущего каталога, если он присутствует. Если вы определите с помощью макроса `AC_DEFINE` дополнительные символы, то вы должны создать этот файл с записями для них. Для символов, определенных макросами `AC_CHECK_HEADERS`, `AC_CHECK_FUNCS`, `AC_CHECK_SIZEOF` или `AC_CHECK_LIB`, программа `autoheader` сама создает комментарии и директивы `#undef`, а не копирует их из файла, поскольку количество возможных символов фактически бесконечно.

Файл, который создается `autoheader`, содержит в основном директивы `#define` и `#undef` и комментарии к ним. Если '`./acconfig.h`' содержит строку '`@TOP@`', то `autoheader` копирует строки, которые находятся выше строки '`@TOP@`', прямо в заголовок создаваемого файла. Аналогично, если '`./acconfig.h`' содержит строку '`@BOTTOM@`', то `autoheader` скопирует строки, расположенные сразу после этой строки, в конец создаваемого файла. Можно не использовать как '`@TOP@`', так и '`@BOTTOM@`'.

Другой способ добиться точно такого же результата – создать в текущем каталоге файлы '*file*`.top`' (обычно '`config.h.top`') и/или '*file*`.bot`'. Если эти файлы существуют, то `autoheader` копирует их содержимое в начало и в конец выводимых данных. Их использование не рекомендуется, поскольку имена этих файлов содержат две точки и не могут применяться в MS-DOS; также это увеличивает содержимое каталога еще на два файла. Но если вы воспользуетесь ключом '`-localdir=`*dir*' для использования '`acconfig.h`' находящегося в другом каталоге, то эти файлы позволят вам вставлять произвольные куски кода в каждый конкретный файл '`config.h.in`'.

`autoheader` распознает следующие ключи командной строки:

`-help`
`-h` Выдает список ключей командной строки и прекращает работу.

`-localdir=`*dir*
`-l` *dir* Искать файлы '`aclocal.m4`' и '`acconfig.h`' (но не файлы '*file*`.top`' и '*file*`.bot`') в каталоге *dir* вместо текущего каталога.

`-macrodir=`*dir*
`-m` *dir* Искать инсталлированные файлы макросов и '`acconfig.h`' в каталоге *dir*. Вы можете также установить переменную среды `AC_MACRODIR`, указывающую на этот каталог; этот ключ переопределяет значение переменной среды.

`-version` Выдает номер версии Autoconf и прекращает работу.

3.5 Настройка других пакетов, находящихся в подкаталогах

В большинстве ситуаций для создания файлов '`Makefile`' в подкаталогах достаточно вызова макроса `AC_OUTPUT`. Однако скрипты `configure`, которые контролируют более чем один независимый пакет, могут использовать макрос `AC_CONFIG_SUBDIRS` для запуска скриптов `configure` для других пакетов, находящихся в подкаталогах.

AC_CONFIG_SUBDIRS (*dir ...*) Macro

Заставляет `AC_OUTPUT` запустить `configure` в каждом подкаталоге *dir*, которые заданы в списке через пробел. Если заданный каталог *dir* не найден, то сообщение об ошибке не выдается, поэтому скрипт `configure` может производить конфигурацию, даже если часть подкаталогов отсутствует. Если заданный каталог *dir* содержит файл '`configure.in`', но не содержит `configure`, то будет использоваться Cygnus-версия скрипта `configure`, местонахождение которой определяется макросом `AC_CONFIG_AUXDIR`.

Скриптам `configure`, находящимся в подкаталогах, передается та же командная строка, что задана текущему скрипту `configure`, только с некоторыми изменениями, когда это необходимо (например, исправление относительных путевых имен для кэш-файла или каталога с исходными текстами). Этот макрос также устанавливает выходную переменную `subdirs` равной списку каталогов '*dir ...*'. Правила '`Makefile`' могут использовать эту переменную для определения того, в какие подкаталоги будет осуществляться рекурсивный переход. Этот макрос может вызываться много раз.

3.6 Префикс по умолчанию

По умолчанию `configure` задает префикс для устанавливаемых файлов равным '`/usr/local`'. Пользователь `configure` может выбрать другой префикс, используя ключи командной строки '`-prefix`' и '`-exec-prefix`'. Есть два способа изменения значения по умолчанию: при создании `configure` и при его запуске.

Некоторые пакеты программного обеспечения могут требовать установки по умолчанию в каталог, отличный от '/usr/local'. Чтобы изменить значение по умолчанию, используйте макрос AC_PREFIX_DEFAULT.

AC_PREFIX_DEFAULT (*prefix*) Macro

Устанавливает значение префикса установки по умолчанию в значение *prefix* вместо '/usr/local'.

Для пользователей может быть удобным, чтобы configure попытался угадать префикс для установки на основе расположения некоторых программ, которые уже установлены в системе. Если вы хотите именно этого, используйте макрос AC_PREFIX_PROGRAM.

AC_PREFIX_PROGRAM (*program*) Macro

Если пользователь не указал префикс для установки (используя ключ '-prefix'), то попробовать определить значение префикса на основе поиска *program* в списке каталогов из PATH. Если *program* найдена, то установить префикс равным родительскому каталогу каталога, в котором находится *program*; иначе оставить неизмененным значение префикса, указанного 'Makefile.in'. Например, если значением *program* является gcc, а в путях найдена программа '/usr/local/gnu/bin/gcc', то значением префикса будет '/usr/local/gnu'.

3.7 Номера версий в configure

Следующие макросы используются для работы с номерами версий в скриптах configure. Их использование не обязательно.

AC_PREREQ (*version*) Macro

Обеспечивает проверку того, что используется достаточно свежая версия Autoconf. Если версия Autoconf, используемая для создания configure, является более старой, чем *version*, то в стандартный поток сообщений об ошибках выдается сообщение и configure не создается. Например:

```
AC_PREREQ(1.8)
```

Этот макрос полезен в том случае, если ваш 'configure.in' полагается на неочевидное поведение, которое изменилось между версиями Autoconf. Если вам необходимы только недавно добавленные макросы, то AC_PREREQ чуть менее полезен, поскольку программа autoconf и так сообщит пользователю о том, какие макросы не найдены. То же самое случится в том случае, если файл 'configure.in' будет обрабатываться версией Autoconf, более старой, чем та, в которой был добавлен макрос AC_PREREQ.

AC_REVISION (*revision-info*) Macro

Копирует метки ревизий *revision-info* в скрипт configure, удаляя знаки доллара и двойные кавычки. Этот макрос позволяет вам помещать метки версий из файла 'configure.in' в configure, но при этом RCS или CVS не станут изменять их при помещении configure в репозиторий. Таким образом, вы можете легко определить, какая версия 'configure.in' соответствует конкретному configure.

Хорошей идеей является вызов этого макроса перед `AC_INIT`, чтобы номер ревизии располагался в начале и ‘`configure.in`’, и `configure`. Для поддержки этого выдача `AC_REVISION` начинается с ‘`#! /bin/sh`’, подобно обычному началу скрипта `configure`.

Вот пример этой строки в ‘`configure.in`’:

```
AC_REVISION($Revision: 1.30 $)dnl
```

это создает в `configure` строки:

```
#! /bin/sh
# From configure.in Revision: 1.30
```

4 Существующие тесты

Эти макросы выполняют проверку отдельных возможностей системы, в которых пакет нуждается или которые он может использовать. Если вам необходимо протестировать возможность, которую не проверяет ни один из имеющихся макросов, то, скорее всего, вы сможете это сделать путем вызова примитивных макросов с соответствующими аргументами (см. Глава 5 [Написание тестов], с. 45).

Эти тесты сообщают пользователю, что именно они проверяют и каков результат проверки. Результаты кэшируются для ускорения последующих запусков `configure` (см. Раздел 6.3 [Кэширование результатов], с. 57).

Некоторые из этих макросов устанавливают выходные переменные. См. Раздел 3.3 [Подстановки в Makefile], с. 12, для того, чтобы узнать о том, как получить значения этих переменных. Фраза “определить *name*” ниже используется как сокращение, обозначающее “определить символ *name* препроцессора C в значение 1”. См. Раздел 6.1 [Определение символов], с. 55, для того, чтобы узнать о том, как получить определения этих символов в вашей программе.

4.1 Альтернативные программы

Эти макросы проверяют наличие или поведение определенных программ. Они используются для выбора между несколькими различными программами и для решения того, что делать, когда нужная программа выбрана. Если для проверки наличия необходимой вам программы нет отдельного макроса, и вам не нужно выполнять проверку специальных возможностей этой программы, то можно использовать один из стандартных макросов проверки программ.

4.1.1 Проверка отдельных программ

Эти макросы выполняют проверку отдельных программ— существуют ли они, а также, в некоторых случаях, проверку того, поддерживают ли эти программы определенные свойства.

AC_DECL_YYTEXT Macro

Определяет `YYTEXT_POINTER`, если `yytext` имеет тип ‘`char *`’, а не ‘`char []`’. Также устанавливает значение выходной переменной `LEX_OUTPUT_ROOT` равным основе имени файла, создаваемого лексическим генератором; обычно это значение равно ‘`lex.yy`’, но иногда используется что-то другое. Эти результаты различаются в зависимости от того, используется ли `lex` или `flex`.

AC_PROG_AWK Macro

Проверяет наличие `mawk`, `gawk`, `nawk` и `awk`, в таком порядке и устанавливает выходную переменную `AWK`, равную имени найденной программы. Сначала пытается найти `mawk`, который считается самой быстрой реализацией.

AC_PROG_CC Macro

Определяет компилятор C, который надо использовать. Если переменная среды `CC` не установлена, то проверить наличие `gcc` и использовать `cc`, если `gcc` не

найден. Устанавливает выходную переменную `CC`, равную имени найденного компилятора.

Если используется компилятор GNU C, то значение переменной `GCC` устанавливается в значение '`yes`', в противном случае оно остается пустым. Если выходная переменная `CFLAGS` еще не была установлена, то установить ее равной '`-g -O2`' для компилятора GNU C ('`-O2`' на системах, в которых GCC не понимает ключа '`-g`') или равной '`-g`' для других компиляторов.

Если используемый компилятор C не создает исполняемых файлов, которые могут запускаться в той системе, где исполняется `configure`, то переменной командного процессора `cross_compiling` присваивается значение '`yes`', в противном случае она получает значение '`no`'. Другими словами, здесь проверяется, отличается ли тип системы, для которой производится сборка, от системы, на которой производится сборка (тип целевой системы не относится к этому тесту). Для получения дополнительной информации о кросс-компиляции См. Глава 8 [Ручная настройка], с. 67.

AC_PROG_CC_C_O Macro

Если компилятор C не может запускаться одновременно с ключами '`-c`' и '`-o`', то определяется переменная `NO_MINUS_C_MINUS_O`.

AC_PROG_CPP Macro

Значение выходной переменной `CPP` устанавливается равным имени команды, которая запускает препроцессор C. Если '`$CC -E`' не работает, то используется '`/lib/cpp`'. Переносимым решением является запуск `CPP` только для обработки файлов с расширением '`.c`'.

Если текущим языком является C (см. Раздел 5.8 [Выбор языка], с. 53), то многие специфические тесты косвенно используют значение переменной `CPP`, вызывая макросы `AC_TRY_CPP`, `AC_CHECK_HEADER`, `AC_EGREP_HEADER` или `AC_EGREP_CPP`.

AC_PROG_CXX Macro

Определяет имя используемого компилятора C++. Проверяется, установлены ли переменные среды `CXX` или `CCC` (именно в таком порядке); если одна из них установлена, то значение выходной переменной `CXX` устанавливается равным значению этой переменной. В противном случае производится поиск компилятора C++, используя вероятные имена (`c++`, `g++`, `gcc`, `CC`, `cxx` и `cc++`). Если ни одна из этих проверок не прошла успешно, то в качестве последнего шанса значение переменной `CXX` устанавливается равным `gcc`.

Если используется компилятор GNU C++, то переменная командного процессора `GXX` получает значение '`yes`', иначе ей присваивается пустое значение. Если выходная переменная `CXXFLAGS` еще не была установлена, то ей присваивается значение '`-g -O2`' для компилятора GNU C++ ('`-O2`' на системах, где G++ не распознает ключ '`-g`') или значение '`-g`' для других систем.

Если используемый компилятор C++ не создает исполняемых файлов, которые могут запускаться в системе, где выполняется `configure`, то переменной командного процессора `cross_compiling` присваивается значение '`yes`', в противном случае устанавливается значение '`no`'. Другими словами, здесь проверяется,

отличается ли тип системы, для которой производится сборка, от системы, на которой производится сборка (тип целевой системы не относится к этому тесту). Для получения дополнительной информации о кросс-компиляции См. Глава 8 [Ручная настройка], с. 67.

AC_PROG_CXXCPP Macro

Значение выходной переменной `CXXCPP` устанавливается равным имени команды, которая запускает препроцессор C++. Если '`$CXX -E`' не работает, то используется '`/lib/cpp`'. Переносимым решением является запуск `CXXCPP` только для обработки файлов с расширениями '`.c`', '`.C`' или '`.cc`'.

Если текущим языком является C++ (см. Раздел 5.8 [Выбор языка], с. 53), то многие специфические тесты косвенно используют значение переменной `CXXCPP`, вызывая `AC_TRY_CPP`, `AC_CHECK_HEADER`, `AC_EGREP_HEADER` или `AC_EGREP_CPP`.

AC_PROG_F77 Macro

Определяет имя используемого компилятора Fortran 77. Если переменная среды `F77` не установлена, то производится проверка наличия программ `g77`, `f77` and `f2c`, в описанном порядке. Имя найденной программы присваивается выходной переменной `F77`.

Если используется программа `g77` (компилятор GNU Fortran 77), то макрос `AC_PROG_F77` установит переменную `G77` равной значению '`yes`', а в противном случае ей будет присвоено пустое значение. Если выходная переменная `FFLAGS` не была установлена в среде, то для `g77` данной переменной присваивается значение '`-g -O2`' (или '`-O2`' в тех случаях когда `g77` не принимает ключ '`-g`'). Иначе, для всех остальных компиляторов Fortran 77, переменной `FFLAGS` присваивается значение '`-g`'.

AC_PROG_F77_C_O Macro

Выполняет проверку того, что компилятор Fortran 77 может запускаться одновременно с ключами '`-c`' и '`-o`'. Если компилятор не принимает эти ключи одновременно, то определяется переменная `F77_NO_MINUS_C_MINUS_O`.

AC_PROG_GCC_TRADITIONAL Macro

Добавляет строку '`-traditional`' к выходной переменной `CC` в том случае, если используемый компилятор GNU C и функции `ioctl` неправильно работают без нее. Обычно это случается, если в старой системе не были установлены исправленные заголовочные файлы. Поскольку свежие версии компилятора GNU C при установке исправляют заголовочные файлы, это становится менее распространенной проблемой.

AC_PROG_INSTALL Macro

Устанавливает выходную переменную `INSTALL`, равной полному пути к совместимой с BSD программе `install`, если она найдена в текущей переменной `PATH`. Иначе, переменная `INSTALL` получает значение '*dir*`/install-sh -c`', проверяя каталоги, указанные в `AC_CONFIG_AUX_DIR` (или каталоги по умолчанию) для определения *dir* (см. Раздел 3.2 [Вывод], с. 10). Этот макрос также устанавливает пе-

ременные INSTALL_PROGRAM и INSTALL_SCRIPT равными значениям '${INSTALL}', а INSTALL_DATA значение '${INSTALL} -m 644'.

Этот макрос не замечает версии install о которых известно, что они не работают. Этот макрос также предпочитает использовать программу на языке C вместо скриптов командного процессора. Вместо 'install-sh', он также может использовать 'install.sh', но это имя устарело, поскольку некоторые программы make имеют правило, которое создает файл 'install' из этого файла, если нет файла 'Makefile'.

Копия 'install-sh', которую вы можете использовать, поставляется с Autoconf. Если вы используете AC_PROG_INSTALL, то вы должны включить в свой дистрибутив либо 'install-sh', либо 'install.sh', иначе configure выдаст ошибку, сообщающую о том, что он не может найти эти файлы— даже если система имеет нормальную программу install. Это мера безопасности, чтобы вы случайно не забыли про этот файл, тем самым лишив пользователя возможности установить ваш пакет в системе, в которой нет BSD-совместимой программы install.

Если вам необходимо использовать вашу собственную программу установки (поскольку она имеет возможности, отсутствующие в стандартных программах install), то нет никакой надобности в использовании макроса AC_PROG_INSTALL; просто поместите путь к вашей программе в ваши файлы 'Makefile.in'.

AC_PROG_LEX Macro

Если найдена программа flex, то выходная переменная LEX получает значение 'flex', а LEXLIB — значение '-lfl', в случае, если библиотека располагается в стандартном месте. Иначе переменная LEX получает значение 'lex', а LEXLIB — значение '-ll'.

AC_PROG_LN_S Macro

Если команда 'ln -s' работает в текущей файловой системе (и операционная, и файловая системы поддерживают символьные ссылки), то выходная переменная LN_S получает значение 'ln -s', в противном случае значение равно 'ln'.

Если ссылка помещается в другой, отличный от текущего, каталог, то смысл этой ссылки зависит от того, какая команда будет использована: 'ln' или 'ln -s'. Чтобы безбоязненно создавать ссылки, используя '$(LN_S)', либо определите, какая форма команды используется и соответственно измените ее аргументы, либо всегда запускайте ln в том каталоге, где будет создаваться ссылка.

Другими словами, не делайте

```
$(LN_S) foo /x/bar
```

Вместо этого выполняйте

```
(cd /x && $(LN_S) foo bar)
```

AC_PROG_RANLIB Macro

Если команда ranlib найдена, то выходная переменная RANLIB получает значение равное 'ranlib', в противном случае используется значение ':' (не делать ничего).

AC_PROG_YACC Macro

Если найдена программа `bison`, то выходная переменная `YACC` получает значение '`bison -y`'. В противном случае, если найдена команда `byacc`, то переменная `YACC` получит значение '`byacc`'. В противном случае `YACC` устанавливается в '`yacc`'.

4.1.2 Общие программы и проверки файлов

Эти макросы используются для обнаружения программ, для которых нет отдельных макросов. Если вам необходимо проверить не только присутствие программы, но и ее поведение, то вам необходимо написать свой тест для данной программы (см. Глава 5 [Написание тестов], с. 45). По умолчанию эти макросы используют переменную среды `PATH`. Если вам необходимо проверить наличие программы, которая может находится в каталогах пользовательской переменной `PATH`, то вы можете передать макросу измененную переменную `PATH`, вот как в этом случае:

```
AC_PATH_PROG(INETD, inetd, /usr/libexec/inetd,
  $PATH:/usr/libexec:/usr/sbin:/usr/etc:etc)
```

AC_CHECK_FILE (*file* [, *action-if-found* [, *action-if-not-found*]]) Macro

Выполняет проверку, существует ли в системе файл *file*. Если он найден, то выполняются команды *action-if-found*, в противном случае выполняется *action-if-not-found*, если задано.

AC_CHECK_FILES (*files*[, *action-if-found* [, *action-if-not-found*]]) Macro

Выполняет макрос `AC_CHECK_FILE` для каждого из файлов в списке *files*. Дополнительно определяет переменную '`HAVE`*file*' для каждого из найденных файлов и устанавливает ее равной 1.

AC_CHECK_PROG (*variable, prog-to-check-for, value-if-found* [, *value-if-not-found* [, *path*, [*reject*]]]) Macro

Проверяет, находится ли программа *prog-to-check-for* в каталогах, перечисленных в переменной `PATH`. Если эта программа найдена, то переменная *variable* устанавливается равным значению *value-if-found*, в противном случае равным значению *value-if-not-found* (если оно задано). Никогда не использует *reject* (имя файла с абсолютным путем), даже если такая программа была найдена в путях поиска; в этом случае переменная *variable* устанавливается, используя абсолютное имя найденной программы *prog-to-check-for*, которая не является *reject*. Если переменная *variable* уже установлена, то ничего не делается. Вызывает макрос `AC_SUBST` для *variable*.

AC_CHECK_PROGS (*variable, progs-to-check-for* [, *value-if-not-found* [, *path*]]) Macro

Проверяет наличие в `PATH` каждой программы из списка через пробел *progs-to-check-for*. Если программа найдена, то переменная *variable* устанавливается в значение, равное имени найденной программы. В противном случае продолжается проверка наличия следующей программы. Если ни одна из программ не

найдена, то переменная *variable* получает значение *value-if-not-found*; если *value-if-not-found* не указано, то значение *variable* не изменяется. Вызывает макрос `AC_SUBST` для *variable*.

AC_CHECK_TOOL (*variable, prog-to-check-for* [, *value-if-not-found* [, *path*]]) Macro

Работает подобно `AC_CHECK_PROG`, но сначала проверяет наличие *prog-to-check-for* с префиксом типа системы, который определяется макросом `AC_CANONICAL_HOST`, за которым следует тире (см. Раздел 8.2 [Канонизация], с. 68). Например, если пользователь запустит '`configure -host=i386-gnu`', то этот вызов:

```
AC_CHECK_TOOL(RANLIB, ranlib, :)
```

установит переменную `RANLIB` в значение '`i386-gnu-ranlib`', если эта программа находится в каталогах, перечисленных в `PATH`, или в '`ranlib`', если эта программа находится в `PATH`, или в '`:`', если ни одна из программ не существует.

AC_PATH_PROG (*variable, prog-to-check-for* [, *value-if-not-found* [, *path*]]) Macro

Работает подобно `AC_CHECK_PROG`, но устанавливает *variable* равной полному пути к найденной программе *prog-to-check-for*.

AC_PATH_PROGS (*variable, progs-to-check-for* [, *value-if-not-found* [, *path*]]) Macro

Подобен макросу `AC_CHECK_PROGS`, но если найдена любая из программ *progs-to-check-for*, то переменная *variable* получает значение, равное полному пути к найденной программе.

4.2 Файлы библиотек

Нижеописанные макросы проверяют наличие определенных библиотек C, C++ или Fortran 77.

AC_CHECK_LIB (*library, function* [, *action-if-found* [, *action-if-not-found* [, *other-libraries*]]]) Macro

В зависимости от текущего языка (см. Раздел 5.8 [Выбор языка], с. 53), макрос пытается убедиться, что функция C, C++ или Fortran 77 с именем *function* доступна (путем проверки, что тестовая программа компонуется с библиотекой *library* для получения доступа к функции). *library* является базовым именем библиотеки; например, для '`-lmp`', используйте '`mp`' в качестве аргумента *library*.

action-if-found является списком команд командного процессора, которые запускаются в случае, если процесс компоновки прошел удачно; *action-if-not-found* является списком команд, которые запускаются, если процесс компоновки потерпел неудачу. Если аргумент *action-if-found* не указан, то действие по умолчанию добавит '`-l`*library*' в переменную `LIBS` и определит переменную '`HAVE_LIB`*library*' (все буквы заглавные).

Если при компоновке с *library* выдаются сообщения о ненайденных символах, которые могут быть найдены, компонуя программы с дополнительными библиотеками, то вы должны передать список этих библиотек через пробелы в аргументе *other-libraries*: '`-lXt -lX11`'. В противном случае этот макрос не сможет определить, что библиотека *library* присутствует, поскольку компоновка тестовой программы всегда будет аварийно завершаться с сообщениями о ненайденных символах.

AC_HAVE_LIBRARY (*library,* [, *action-if-found* [, *action-if-not-found* [, *other-libraries*]]]) Macro

Этот макрос аналогичен вызову `AC_CHECK_LIB` с аргументом *function*, равным `main`. Вдобавок, *library* может быть указана как '`foo`', '`-lfoo`' или '`libfoo.a`'. Во всех этих случаях компилятору передается строка '`-lfoo`'. Однако *library* не может быть переменной командного процессора; ее значение должно быть символьным именем. Этот макрос считается устаревшим.

AC_SEARCH_LIBS (*function, search-libs* [, *action-if-found* [, *action-if-not-found* [, *other-libraries*]]]) Macro

Производит поиск библиотеки, определяющей функцию *function*, если она еще не доступна. Это подобно вызову макроса `AC_TRY_LINK_FUNC` сначала без указания библиотек, а затем для каждой из библиотек, перечисленных в списке *search-libs*.

Если функция найдена, то выполняются команды *action-if-found*, в противном случае выполняются *action-if-not-found*.

Если при компоновке с *library* выдаются сообщения о ненайденных символах, которые могут быть найдены, компонуя программы с дополнительными библиотеками, то вы должны передать список этих библиотек через пробел, используя аргумент *other-libraries*: '`-lXt -lX11`'. В противном случае этот макрос не сможет определить, что библиотека *library* присутствует, поскольку компоновка тестовой программы всегда будет аварийно завершаться с сообщениями о ненайденных символах.

AC_SEARCH_LIBS (*function, search-libs*[, *action-if-found* [, *action-if-not-found*]]) Macro

Этот макрос эквивалентен вызову `AC_TRY_LINK_FUNC` для каждой из библиотек, перечисленных в списке *search-libs*. Макрос добавляет '`-l`*library*' к содержимому переменной `LIBS` для первой библиотеки, которая содержит *function* и выполняет *action-if-found*. В противном случае выполняется *action-if-not-found*.

4.3 Библиотечные функции

Следующие макросы проверяют отдельные функции библиотеки C. Если для функции, которая вам нужна, нет отдельного макроса, и вам не нужно проверять специальные возможности этой функции, то можно использовать один из общих макросов проверки функций.

4.3.1 Проверка отдельных функций

Эти макросы выполняют проверку отдельных функций: существуют ли они и, в отдельных случаях, как они работают при задании определенных аргументов.

AC_FUNC_ALLOCA Macro

Проверяет, как получить `alloca`. Макрос пробует получить встроенную версию, проверяя наличие файла '`alloca.h`' или предопределенных макросов препроцессора C `__GNUC__` и `_AIX`. Если этот макрос находит '`alloca.h`', то определяется переменная `HAVE_ALLOCA_H`.

Если эти попытки оканчиваются неудачей, то макрос будет искать функцию в стандартной библиотеке C. Если любой из этих методов закончится успешно, то будет определена переменная `HAVE_ALLOCA`. В противном случае выходная переменная `ALLOCA` получит значение '`alloca.o`' и будет определена переменная `C_ALLOCA` (так что программы смогут периодически вызывать '`alloca(0)`' для сборки мусора). Эта переменная отделена от `LIBOBJS`, так что несколько программ смогут использовать одно и то же значение `ALLOCA`, без необходимости создания настоящей библиотеки, если лишь некоторые из них используют код в `LIBOBJS`.

Эти макросы не пытаются получить `alloca` из библиотеки System V R3 '`libPW`' или из библиотеки System V R4 '`libucb`', поскольку эти библиотеки содержат некоторые несовместимые функции, что может в дальнейшем вызвать проблемы. Некоторые версии библиотек даже не содержат `alloca` или содержат версию с ошибками. Если вы все таки хотите использовать `alloca` из этих библиотек, то вместо компиляции файла '`alloca.c`' используйте `ar` для извлечения из них '`alloca.o`'.

Для правильного объявления этой функции исходные тексты, использующие `alloca`, должны начинаться примерно с нижеизложенного кода. В некоторых версиях AIX, объявление `alloca` должно предшествовать всему, за исключением комментариев и директив препроцессора. Директива `#pragma` специальным образом выровнена (перед ней стоит несколько пробелов), чтобы старые не-ANSI компиляторы C игнорировали ее, а не выдавали ошибку.

```
/* AIX требует, чтобы это было первым кодом в файле.  */
#ifndef __GNUC__
# if HAVE_ALLOCA_H
#  include <alloca.h>
# else
#  ifdef _AIX
 #pragma alloca
#  else
#   ifndef alloca /* предопределено в cc +Olibcalls фирмы HP */
char *alloca ();
#   endif
#  endif
# endif
#endif
```

AC_FUNC_CLOSEDIR_VOID Macro

Если значение, возвращаемое функцией `closedir`, не несет полезной информации, то определяется `CLOSEDIR_VOID`. В противном случае тот, кто вызывает эту функцию, может проверить возвращаемое значение на наличие ошибки.

AC_FUNC_FNMATCH Macro

Если функция `fnmatch` доступна и работает (в отличие от имеющейся в SunOS 5.4), то определяется переменная `HAVE_FNMATCH`.

AC_FUNC_GETLOADAVG Macro

Проверка того, как получить данные о загрузке системы. Если система имеет функцию `getloadavg`, то этот макрос определяет переменную `HAVE_GETLOADAVG`, и добавляет к `LIBS` библиотеки, необходимые для получения этой функции.

В противном случае макрос добавляет '`getloadavg.o`' к выходной переменной `LIBOBJS` и, возможно, определяет другие макросы препроцессора C и выходные переменные:

1. Он определяет `SVR4`, `DGUX`, `UMAX` или `UMAX4_3` на соответствующих системах.
2. Если он находит '`nlist.h`', то он определяет переменную `NLIST_STRUCT`.
3. Если '`struct nlist`' имеет поле '`n_un`', то определяется переменная `NLIST_NAME_UNION`.
4. Если компиляция '`getloadavg.c`' определяет `LDAV_PRIVILEGED`, то программы необходимо специальным образом устанавливать на эту систему, чтобы `getloadavg` работала, и этот макрос определяет `GETLOADAVG_PRIVILEGED`.
5. Этот макрос устанавливает выходную переменную `NEED_SETGID`. Ее значением является '`true`', если требуется специальная установка, и '`false`' в противном случае. Если `NEED_SETGID` равен '`true`', то этот макрос устанавливает `KMEM_GROUP` в значение, равное названию группы, которая должна владеть установленной программой.

AC_FUNC_GETMNTENT Macro

Проверяет наличие `getmntent` в библиотеках '`sun`', '`seq`' и '`gen`' для Irix 4, PTX и Unixware, соответственно. Затем, если функция `getmntent` доступна, определяется переменная `HAVE_GETMNTENT`.

AC_FUNC_GETPGRP Macro

Если `getpgrp` запускается без аргументов (версия POSIX.1), то определяется `GETPGRP_VOID`. В противном случае функция является BSD-версией, которая принимает в качестве аргумента идентификатор процесса. Этот макрос не выполняет проверку наличия `getpgrp`; если вам необходимо работать в такой ситуации, то сначала вызовите `AC_CHECK_FUNC` для `getpgrp`.

AC_FUNC_MEMCMP Macro

Если функция `memcmp` недоступна, или не работает с восьмибитными данными (как функция в SunOS 4.1.3), то '`memcmp.o`' добавляется к выходной переменной `LIBOBJS`.

AC_FUNC_MMAP Macro

Если функция `mmap` существует и работает правильно, то определяется переменная `HAVE_MMAP`. Проверяется только фиксированное приватное отображение уже отображенной памяти.

AC_FUNC_SELECT_ARGTYPES Macro

Определяет правильный тип, передаваемый каждому из аргументов функции `select`, и определяет эти типы в переменных `SELECT_TYPE_ARG1`, `SELECT_TYPE_ARG234` и `SELECT_TYPE_ARG5`. Значением по умолчанию для `SELECT_TYPE_ARG1` является '`int`', для `SELECT_TYPE_ARG234` типом по умолчанию является '`int *`' и для `SELECT_TYPE_ARG5` типом по умолчанию является '`struct timeval *`'.

AC_FUNC_SETPGRP Macro

Если `setpgrp` запускается без аргументов (версия POSIX.1), то определяется `SETPGRP_VOID`. В противном случае, функция является BSD-версией, которая принимает в качестве аргумента идентификатор процесса. Этот макрос не выполняет проверку наличия `setpgrp`; если вам необходимо работать в такой ситуации, то сначала вызовите `AC_CHECK_FUNC` для `setpgrp`.

AC_FUNC_SETVBUF_REVERSED Macro

Если `setvbuf` принимает тип буферизации как второй аргумент, а указатель на буфер как третий аргумент, а не наоборот, то определяется переменная `SETVBUF_REVERSED`. Это справедливо для System V до выпуска 3.

AC_FUNC_STRCOLL Macro

Если функция `strcoll` существует и работает правильно, то определяется переменная `HAVE_STRCOLL`. Этот макрос выполняет больше проверок, чем просто вызов '`AC_CHECK_FUNCS(strcoll)`', потому что некоторые системы имеют неправильные определения `strcoll`, которыми не следует пользоваться.

AC_FUNC_STRFTIME Macro

Проверка наличия `strftime` в библиотеке '`intl`', для SCO UNIX. Затем, если `strftime` доступна, определяется переменная `HAVE_STRFTIME`.

AC_FUNC_UTIME_NULL Macro

Если вызов '`utime(`*file*`, NULL)`' устанавливает время модификации файла *file* в текущее время, то определить переменную `HAVE_UTIME_NULL`.

AC_FUNC_VFORK Macro

Если найден файл '`vfork.h`', то определяется переменная `HAVE_VFORK_H`. Если работающая версия `vfork` не найдена, то определить `vfork` как `fork`. Этот макрос проверяет несколько известных ошибок в реализации `vfork` и если найдена одна из таких реализаций, то считается, что система не имеет работающей версии `vfork`. Макрос не считает, ошибкой реализации, если при вызове потомком функции `signal` изменяются обработчики сигналов родителя, поскольку потомки редко изменяют обработчики сигналов родительского процесса.

AC_FUNC_VPRINTF Macro

Если найдена функция `vprintf`, то определяется переменная `HAVE_VPRINTF`. В противном случае, если найдена функция `_doprnt`, то определяется переменная `HAVE_DOPRNT`. (Если функция `vprintf` доступна, то вы можете считать, что функции `vfprintf` и `vsprintf` тоже доступны).

AC_FUNC_WAIT3 Macro

Если функция `wait3` найдена, и заполняет содержимое своего третьего аргумента (‘`struct rusage *`’), чего не делает HP-UX, то определяется переменная `HAVE_WAIT3`.

4.3.2 Проверка базовых функций

Эти макросы используются для нахождения функций, которые не имеют специальных макросов проверки. Если функции могут находиться в других библиотеках, а не в стандартной библиотеке C, то сначала вызовите макрос `AC_CHECK_LIB` для проверки наличия нужных библиотек. Если вам нужно не только проверить, существует ли функция, но и уточнить ее поведение, то вам придется написать свой собственный тест для этой функции (см. Глава 5 [Написание тестов], с. 45).

AC_CHECK_FUNC (*function*, [*action-if-found* [, *action-if-not-found*]]) Macro

Если функция C с именем *function* доступна, то запускаются команды командного процессора *action-if-found*, в противном случае запускаются *action-if-not-found*. Если вы просто хотите определить символ препроцессора, если функция существует, то вместо этого макроса попробуйте использовать `AC_CHECK_FUNCS`. Этот макрос проверяет компоновку с библиотекой C, даже если был вызван макрос `AC_LANG_CPLUSPLUS`, поскольку C++ является более стандартизованным, чем C. (см. Раздел 5.8 [Выбор языка], с. 53, для дополнительной информации о выборе языка, для которого проводятся проверки).

AC_CHECK_FUNCS (*function*... [, *action-if-found* [, *action-if-not-found*]]) Macro

Для каждой из заданных *function* в списке, разделенном пробелами, в случае если она доступна, определить переменную `HAVE_`*function* (все буквы заглавные). Если задан аргумент *action-if-found*, то выполняется дополнительный код командного процессора, если одна из функций найдена. Вы можете задать значение ‘`break`’ для прекращения цикла при нахождении первой функции. Если задан аргумент *action-if-not-found*, то эти команды выполняются, когда одна из функций не найдена.

AC_REPLACE_FUNCS (*function*...) Macro

Этот макрос подобен вызову макроса `AC_CHECK_FUNCS`, используя код *action-if-not-found*, который добавляет ‘*function*`.o`’ к выходной переменной `LIBOBJS`. Вы можете объявить функцию, для которой будет использована ваша замена, поместив ее прототип между директивами ‘`#ifndef HAVE_`*function*’. Если система имеет нужную функцию, то эта функция, вероятно, будет объявлена в заголовочном файле, который вы должны включить в свою программу, так что вы не должны повторно объявлять ее, во избежание конфликта объявлений.

4.4 Заголовочные файлы

Следующие макросы проверяют наличие определенных заголовочных файлов языка C. Если для нужного вам заголовочного файла нет специального макроса, и при этом вам не нужно проверять специальные особенности этого файла, то можно использовать один из стандартных макросов проверки заголовочных файлов.

4.4.1 Проверка отдельных заголовочных файлов

Эти макросы выполняют проверку отдельных заголовочных файлов— существуют ли они и, в некоторых случаях, объявлены ли в них какие-либо символы.

AC_DECL_SYS_SIGLIST Macro

Определяет переменную `SYS_SIGLIST_DECLARED`, если переменная `sys_siglist` объявлена в системном заголовочном файле— либо в ‘`signal.h`’, либо в ‘`unistd.h`’.

AC_DIR_HEADER Macro

Подобен вызову макросов `AC_HEADER_DIRENT` и `AC_FUNC_CLOSEDIR_VOID`, но определяет немного другой набор макросов препроцессора C, для указания того, какой заголовочный файл найден. Этот макрос и имена, которые он определяет, считаются устаревшими. Макрос определяет следующие имена:

‘`dirent.h`’
: `DIRENT`

‘`sys/ndir.h`’
: `SYSNDIR`

‘`sys/dir.h`’
: `SYSDIR`

‘`ndir.h`’ `NDIR`

Вдобавок, если функция `closedir` не возвращает информативного значения, то определяется переменная `VOID_CLOSEDIR`.

AC_HEADER_DIRENT Macro

Проверка следующих заголовочных файлов, и для первого файла, который найден и определяет ‘`DIR`’, определить нижеследующие макросы препроцессора C:

‘`dirent.h`’
: `HAVE_DIRENT_H`

‘`sys/ndir.h`’
: `HAVE_SYS_NDIR_H`

‘`sys/dir.h`’
: `HAVE_SYS_DIR_H`

‘`ndir.h`’ `HAVE_NDIR_H`

В исходном тексте объявления библиотеки каталогов должны выглядеть примерно так:

```
#if HAVE_DIRENT_H
# include <dirent.h>
# define NAMLEN(dirent) strlen((dirent)->d_name)
#else
# define dirent direct
# define NAMLEN(dirent) (dirent)->d_namlen
# if HAVE_SYS_NDIR_H
#  include <sys/ndir.h>
# endif
# if HAVE_SYS_DIR_H
#  include <sys/dir.h>
# endif
# if HAVE_NDIR_H
#  include <ndir.h>
# endif
#endif
```

Используя нижеследующие объявления, программа должна объявить переменные с типом `struct dirent`, а не `struct direct`, а доступ к полю длины имени каталога она должна получать путем передачи указателя на `struct dirent` макросу `NAMLEN`.

Этот макрос также проверяет наличие библиотек ‘`dir`’ и ‘`x`’ в SCO Xenix.

AC_HEADER_MAJOR Macro

Если ‘`sys/types.h`’ не определяет `major`, `minor` и `makedev`, но это делается в ‘`sys/mkdev.h`’, то определяется переменная `MAJOR_IN_MKDEV`; в противном случае, если эти функции определяются в ‘`sys/sysmacros.h`’, то определяется переменная `MAJOR_IN_SYSMACROS`.

AC_HEADER_STDC Macro

Определяет `STDC_HEADERS`, если система имеет заголовочные файлы ANSI C. Это макрос проверяет наличие ‘`stdlib.h`’, ‘`stdarg.h`’, ‘`string.h`’ и ‘`float.h`’; если система имеет эти файлы, то, скорее всего, имеются и остальные заголовочные файлы ANSI C. Этот макрос также проверяет, что ‘`string.h`’ объявляет `memchr` (и поэтому, скорее всего, еще и другие функции `mem`), объявляется ли в ‘`stdlib.h`’ функция `free` (и, по видимому, `malloc` и другие относящиеся к ним функции), и будут ли макросы из ‘`ctype.h`’ работать с символами с установленным старшим битом, как этого требует ANSI C.

Используйте `STDC_HEADERS` вместо of `__STDC__` для определения, имеются ли в системе совместимые с ANSI заголовочные файлы (и, вероятно, функции библиотеки C), поскольку многие системы, имеющие GCC, не имеют заголовочные файлы ANSI C.

На системах без заголовочных файлов ANSI C существует так много вариантов, что, вероятно, легче объявить используемые вами функции, чем точно определять, какой из заголовочные файлов определяет эти функции. Некоторые си-

стемы содержат смесь функций ANSI и BSD; некоторые из них по большей части совместимы с ANSI, но не имеют ‘memmove’; некоторое определяют функции BSD как макросы в файлах ‘string.h’ или ‘strings.h’; некоторые из них имеют только функции BSD, но с ‘string.h’; некоторые объявляют функции работы с памятью в ‘memory.h’, некоторые в ‘string.h’; и т. п. Скорее всего, достаточно проверить наличие одной функции работы со строками и одной функции работы с памятью; если библиотека имеет ANSI-версии этих функций, то, скорее всего, она имеет и большинство других функций. Вы должны поместить следующий код в ‘configure.in’:

```
AC_HEADER_STDC
AC_CHECK_FUNCS(strchr memcpy)
```

а затем, в вашем коде вы можете поместить следующие строки:

```
#if STDC_HEADERS
# include <string.h>
#else
# ifndef HAVE_STRCHR
#  define strchr index
#  define strrchr rindex
# endif
char *strchr (), *strrchr ();
# ifndef HAVE_MEMCPY
#  define memcpy(d, s, n) bcopy ((s), (d), (n))
#  define memmove(d, s, n) bcopy ((s), (d), (n))
# endif
#endif
```

Если вы используете функции, которые не имеют эквивалентов в BSD, такие как memchr, memset strtok или strspn, то просто макросов будет недостаточно; вы должны предоставить реализацию каждой из функций. Простой способ подключить ваши реализации только если они действительно нужны (потому что функции из системной библиотеки могут быть вручную оптимизированы) — это, например, поместить функцию memchr в файл ‘memchr.c’, и использовать макрос ‘AC_REPLACE_FUNCS(memchr)’.

AC_HEADER_SYS_WAIT Macro

Если ‘sys/wait.h’ существует и совместим с POSIX.1, то определяется переменная HAVE_SYS_WAIT_H. Несовместимость может возникнуть, если файла ‘sys/wait.h’ не существует, или для сохранения значения статуса он использует старую BSD-версию union wait вместо int. Если ‘sys/wait.h’ не является совместимым с POSIX.1, то вместо его включения определяется макрос POSIX.1 с его обычной реализацией. Вот пример:

```
#include <sys/types.h>
#if HAVE_SYS_WAIT_H
# include <sys/wait.h>
#endif
#ifndef WEXITSTATUS
# define WEXITSTATUS(stat_val) ((unsigned)(stat_val) >> 8)
#endif
#ifndef WIFEXITED
# define WIFEXITED(stat_val) (((stat_val) & 255) == 0)
#endif
```

AC_MEMORY_H Macro

Определяет NEED_MEMORY_H, если memcpy, memcmp, и т. п. не объявлены в файле 'string.h' и существует файл 'memory.h'. Этот макрос является устаревшим; вместо него используйте вызов AC_CHECK_HEADERS(memory.h). Смотрите пример для AC_HEADER_STDC.

AC_UNISTD_H Macro

Определяет переменную HAVE_UNISTD_H, если в системе имеется файл 'unistd.h'. Этот макрос является устаревшим; вместо него используйте вызов 'AC_CHECK_HEADERS(unistd.h)'.

Для проверки того, что система поддерживает POSIX.1, можно использовать следующий код:

```
#if HAVE_UNISTD_H
# include <sys/types.h>
# include <unistd.h>
#endif

#ifdef _POSIX_VERSION
/* Код для систем POSIX.1 */
#endif
```

_POSIX_VERSION определяется, когда 'unistd.h' подключен в системах, совместимых с POSIX.1. Если файла 'unistd.h' не существует, то, скорее всего, эта система не относится к POSIX.1. Однако некоторые не-POSIX.1 системы имеют файл 'unistd.h'.

AC_USG Macro

Определяет USG, если система не имеет файла 'strings.h', rindex, bzero и т. п. Это означает, что система имеет 'string.h', strrchr, memset и т. п.

Символ USG является устаревшим. Вместо этого макроса смотрите пример для AC_HEADER_STDC.

4.4.2 Базовые проверки заголовочных файлов

Эти макросы используются для нахождения системных заголовочных файлов, для которых не существует отдельного теста. Если вам надо проверить не только наличие заголовочного файла, но и его содержимое, то придется написать для этого собственный тест (см. Глава 5 [Написание тестов], с. 45).

AC_CHECK_HEADER (*header-file*, [*action-if-found* [, *action-if-not-found*]]) Macro

Если системный заголовочный файл *header-file* существует, то исполняются команды командного процессора *action-if-found*, в противном случае выполняются *action-if-not-found*. Если вы просто хотите определить символ, если заголовочный файл доступен, то лучше используйте макрос `AC_CHECK_HEADERS`.

AC_CHECK_HEADERS (*header-file*... [, *action-if-found* [, *action-if-not-found*]]) Macro

Для каждого системного заголовочного файла *header-file*, заданного в списке через пробел, в случае его существования определить переменную `HAVE_`*header-file* (все буквы заглавные). Если задан аргумент *action-if-found*, то выполняется дополнительный код командного процессора в случае, когда файл найден. Вы можете задать аргумент '`break`' для прекращения итераций, когда найден первый файл. Если задан аргумент *action-if-not-found*, то он выполняется, когда заголовочный файл не найден.

4.5 Структуры

Следующие макросы проверяют наличие определенных структур или полей структур. Для проверки структур, не перечисленных в этом разделе, используйте макрос `AC_EGREP_CPP` (см. Раздел 5.1 [Исследование объявлений], с. 45) или `AC_TRY_COMPILE` (см. Раздел 5.2 [Проверка синтаксиса], с. 46).

AC_HEADER_STAT Macro

Если макросы `S_ISDIR`, `S_ISREG` и т. п., определенные в '`sys/stat.h`', работают неправильно (возвращая неверные положительные результаты), то определяется переменная `STAT_MACROS_BROKEN`. Это происходит на системах Tektronix UTekV, Amdahl UTS и Motorola System V/88.

AC_HEADER_TIME Macro

Если программа может подключать как '`time.h`', так и '`sys/time.h`', то определяется переменная `TIME_WITH_SYS_TIME`. В некоторых старых системах '`sys/time.h`' подключает '`time.h`', но '`time.h`' не защищен от многократного подключения, так что программы не должны явно подключать оба файла. Этот макрос полезен для программ, которые, например, используют структуры `struct timeval` или `struct timezone`, вместе с `struct tm`. Этот макрос лучше всего использовать вместе с `HAVE_SYS_TIME_H`, который может быть проверен с помощью `AC_CHECK_HEADERS(sys/time.h)`.

```
#if TIME_WITH_SYS_TIME
# include <sys/time.h>
# include <time.h>
#else
# if HAVE_SYS_TIME_H
#  include <sys/time.h>
# else
#  include <time.h>
# endif
#endif
```

AC_STRUCT_ST_BLKSIZE Macro

Если `struct stat` содержит поле `st_blksize`, то определяется переменная `HAVE_ST_BLKSIZE`.

AC_STRUCT_ST_BLOCKS Macro

Если `struct stat` содержит поле `st_blocks`, то определяется переменная `HAVE_ST_BLOCKS`. В противном случае, '`fileblocks.o`' добавляется к выходной переменной `LIBOBJS`.

AC_STRUCT_ST_RDEV Macro

Если `struct stat` содержит поле `st_rdev`, то определяется переменная `HAVE_ST_RDEV`.

AC_STRUCT_TM Macro

Если '`time.h`' не определяет `struct tm`, то определяется символ `TM_IN_SYS_TIME`, что означает, что '`sys/time.h`' следовало бы определить `struct tm`.

AC_STRUCT_TIMEZONE Macro

Определяет, как получить текущую временную зону. Если `struct tm` имеет поле `tm_zone`, то определяется переменная `HAVE_TM_ZONE`. В противном случае, если найден внешний массив `tzname`, то определяется переменная `HAVE_TZNAME`.

4.6 Объявления типов

Следующие макросы проверяют определения типов (typedef) языка C. Если для нужного вам определения типа нет специального макроса, и вам не нужно выполнять проверку специальных возможностей, то можно использовать общие макросы проверки объявлений типов.

4.6.1 Проверка отдельных объявлений типов

Эти макросы проверяют конкретные объявления типов C в файлах '`sys/types.h`' и '`stdlib.h`' (если он существует).

AC_TYPE_GETGROUPS Macro

Определяет `GETGROUPS_T` равным `gid_t` или `int`, в зависимости от того, что именно является базовым типом массива-аргумента функции `getgroups`.

AC_TYPE_MODE_T Macro

Если `mode_t` не определен, то определить тип `mode_t` равным `int`.

AC_TYPE_OFF_T Macro

Если `off_t` не определен, то определить `off_t` равным `long`.

AC_TYPE_PID_T Macro

Если `pid_t` не определен, то определить `pid_t` равным `int`.

AC_TYPE_SIGNAL Macro

Если '`signal.h`' определяет `signal` как возвращающий указатель на функцию, возвращающую `void`, то переменная `RETSIGTYPE` становится равной `void`; в противном случае она определяется с типом `int`.

Определить обработчики сигналов как возвращающие тип `RETSIGTYPE`:

```
RETSIGTYPE
hup_handler ()
{
...
}
```

AC_TYPE_SIZE_T Macro

Если `size_t` не определен, то определить `size_t` как `unsigned`.

AC_TYPE_UID_T Macro

Если `uid_t` не определен, то определить `uid_t` равным `int` и `gid_t` равным `int`.

4.6.2 Базовые проверки объявлений типов

Эти макросы используются для проверки определений типов (typedef), которые не были описаны в разделе конкретных макросов проверок.

AC_CHECK_TYPE (*type, default*) Macro

Если тип *type* не определен в '`sys/types.h`', или в '`stdlib.h`', или в '`stddef.h`' (если они существуют), то определить этот тип равным встроенному типу C (или C++) *default*; например, '`short`' или '`unsigned`'.

4.7 Характеристики компилятора C

Следующие макросы выполняют проверку свойств компилятора C или архитектуры машины. Для проверки характеристик, не перечисленных в этом разделе, используйте макросы `AC_TRY_COMPILE` (см. Раздел 5.2 [Проверка синтаксиса], с. 46) или `AC_TRY_RUN` (см. Раздел 5.4 [Время выполнения], с. 48).

AC_C_BIGENDIAN Macro

Если слова хранятся в порядке, когда самый значимый байт хранится первым (подобно процессорам Motorola и SPARC, но не Intel и VAX), то определяется переменная `WORDS_BIGENDIAN`.

AC_C_CONST Macro

Если компилятор C не поддерживает полностью ключевое слово `const`, то макросу `const` присваивается пустое значение. Некоторые компиляторы C не определяют константу `__STDC__`, но поддерживают `const`; некоторые компиляторы, определяющие `__STDC__`, не полностью поддерживают `const`. Программы могут просто использовать `const`, как будто любой компилятор C поддерживает его; для тех компиляторов, которые не имеют такой поддержки, '`Makefile`' или заголовочный файл настройки определят это слово как имеющее пустое значение.

AC_C_INLINE Macro

Если компилятор C поддерживает `inline`, то ничего не делается. В противном случае, `inline` определяется равным `__inline__` или `__inline`, если компилятор поддерживает один из этих вариантов, иначе `inline` определяется равным пустому значению.

AC_C_CHAR_UNSIGNED Macro

Если тип C `char` является беззнаковым, то определяется переменная `__CHAR_UNSIGNED__` (если компилятор C еще не определил ее).

AC_C_LONG_DOUBLE Macro

Если компилятор C поддерживает тип `long double`, то определяется переменная `HAVE_LONG_DOUBLE`. Некоторые компиляторы C, которые не определяют `__STDC__`, поддерживают `long double`; а некоторые компиляторы, определяющие `__STDC__`, не поддерживают тип `long double`.

AC_C_STRINGIZE Macro

Если препроцессор C поддерживает строковый (stringizing) оператор, то определяется переменная `HAVE_STRINGIZE`. Строковым (stringinzing)оператором является '`#`' и он используется в макросах, например:

```
#define x(y) #y
```

AC_CHECK_SIZEOF (*type* [, *cross-size*]) Macro

Определить `SIZEOF_`*uctype* равным числу байтов во встроенном типе C (или C++) *type*, например, '`int`' или '`char *`'. Если '`type`' неизвестен компилятору, то переменная получает значение 0. *uctype* является *type*, со строчными буквами, преобразованными в прописные, пробелы преобразуются в знаки подчеркивания, и знаки звездочка (`*`) заменяются на '`P`'. Если производится кросс-компиляция, то используется значение *cross-size* (если оно задано), в противном случае `configure` прекращает работу с выдачей сообщения об ошибке.

Например, вызов

```
AC_CHECK_SIZEOF(int *)
```

определяет `SIZEOF_INT_P` равным 8 на системах DEC Alpha AXP.

AC_INT_16_BITS Macro

Если тип `int` имеет размер 16 бит, то определяется переменная `INT_16_BITS`. Этот макрос является устаревшим; вместо него лучше использовать общий макрос '`AC_CHECK_SIZEOF(int)`'.

AC_LONG_64_BITS Macro

Если тип `long int` имеет размер 64 бита, то определяется переменная `LONG_64_BITS`. Этот макрос является устаревшим; вместо него лучше использовать вызов '`AC_CHECK_SIZEOF(long)`'.

4.8 Характеристики компилятора Fortran 77

Следующие макросы используются для проверки характеристик компилятора Fortran 77. Для проверки характеристик, не перечисленных в этом разделе, используйте макросы `AC_TRY_COMPILE` (см. Раздел 5.2 [Проверка синтаксиса], с. 46) или `AC_TRY_RUN` (см. Раздел 5.4 [Время выполнения], с. 48), убедившись, что перед этим вы установили Fortran 77 текущим языком. `AC_LANG_FORTRAN77` (см. Раздел 5.8 [Выбор языка], с. 53).

AC_F77_LIBRARY_LDFLAGS Macro

Определяет ключи командной строки компоновщика (например, '`-L`' и '`-l`') для *внутренних библиотек Fortran 77 и библиотек времени исполнения*, которые требуются для правильной компоновки программ на Fortran 77 или разделяемых библиотек. Выходная переменная `FLIBS` устанавливается равной этим флагам.

Этот макрос предназначен для использования в ситуациях, когда необходимо смешать исходный код, например на C++ и Fortran 77, в одну программу или разделяемую библиотеку (см. раздел "Смешивание кода Fortran 77 с кодом на C и C++" в *GNU Automake*).

Например, если объектные файлы от компиляторов C++ и Fortran 77 должны быть скомпонованы вместе, то для компоновки должен использоваться компилятор/компоновщик C++, поскольку специфические для C++ вещи, такие как вызовы глобальных конструкторов, подстановке шаблонов, разрешении обработки исключений, и т. п., нуждаются в специальных действиях во время компоновки.

Однако в этих случаях должны быть подключены и внутренние библиотеки Fortran 77, а также библиотеки времени исполнения, а компилятор/компоновщик C++ просто не знает, какие библиотеки Fortran 77 должны быть добавлены. Для определения библиотек Fortran 77 и был создан макрос `AC_F77_LIBRARY_LDFLAGS`.

4.9 Системные сервисы

Следующие макросы проверяют наличие сервисов операционной системы или ее возможности.

AC_CYGWIN Macro

Проверяет наличие среды Cygwin. Если она присутствует, то переменная среды `CYGWIN` получает значение '`yes`'. В противном случае переменная `CYGWIN` получает пустое значение.

AC_EXEEXT Macro

Определяет переменную подстановки `EXEEXT`, основанную на расширении файла, выдаваемого компилятором, после исключения файлов с расширениями .c, .o и .obj. Для Unix обычным значением является пустая строка, а для Win32 — '`.exe`' и '`.EXE`'.

AC_OBJEXT Macro

Определяет переменную подстановки `OBJEXT`, основанную на выводе компилятора, после исключения файлов с расширением .c. Обычно имеет значение '`.o`' в Unix, и '`.obj`' на системах Win32.

AC_MINGW32 Macro

Проверяет наличие среды компилятора MingW32. Если она присутствует, то переменная `MINGW32` получает значение '`yes`'. В противном случае переменная `MINGW32` получает пустое значение.

AC_PATH_X Macro

Этот макрос пробует определить расположение заголовочных файлов и библиотек X Window System. Если пользователь задал ключи командной строки '`-x-includes=`*dir*' и '`-x-libraries=`*dir*', то используются заданные каталоги. Если один из ключей или оба не заданы, то пропущенные значения получают запуском `xmkmf` для простого '`Imakefile`' и разбора полученного файла '`Makefile`'. Если произошел сбой (например, если `xmkmf` отсутствует), то производится поиск в нескольких каталогах, где часто располагаются эти файлы. Если один из этих способов был удачен, то переменные командного процессора `x_includes` и `x_libraries` устанавливаются равными найденным каталогам (в том случае, если эти каталоги не входят в пути, в которых компилятор по умолчанию производит поиск).

Если оба этих метода дают сбой, или пользователь задал ключ командной строки '`-without-x`', то переменная командного процессора `no_x` получает значение '`yes`'; в противном случае она получает пустое значение.

AC_PATH_XTRA Macro

Расширенная версия `AC_PATH_X`. Она добавляет к выходной переменной `X_CFLAGS` ключи компилятора C, которые необходимы X, а также флаги X для компоновщика к переменной `X_LIBS`. Если X не доступна, то добавляется '`-DX_DISPLAY_MISSING`' к `X_CFLAGS`.

Этот макрос также выполняет проверки специальных библиотек, в которых нуждаются некоторые системы для того, чтобы скомпилировать программу для X. Он добавляет все, что необходимо для таких систем, к выходной переменной `X_EXTRA_LIBS`. Он также проверяет наличие специальных библиотек X11R6, которые необходимо скомпоновать до использования '`-lX11`', и добавляет найденные библиотеки к выходной переменной `X_PRE_LIBS`.

AC_SYS_INTERPRETER Macro

Проверяет, поддерживает ли система начало скриптов со строки в форме '`#! /bin/csh`' для выбора интерпретатора, который будет использоваться для

данного скрипта. После запуска этого макроса код командного процессора в `configure.in` может проверить переменную `interpval`; она будет равна '`yes`', если система поддерживает '`#!`', и '`no`' в противном случае.

AC_SYS_LONG_FILE_NAMES Macro

Если система поддерживает имена файлов длиннее 14 символов, то будет определена переменная `HAVE_LONG_FILE_NAMES`.

AC_SYS_RESTARTABLE_SYSCALLS Macro

Если система автоматически перезапускает системный вызов, который был прерван сигналом, то определяется переменная `HAVE_RESTARTABLE_SYSCALLS`.

4.10 Варианты UNIX

Следующие макросы проверяют наличие конкретных операционных систем, что может потребовать специальной обработки в программах из-за исключительных странностей в их заголовочных файлах или библиотеках. Эти макросы являются бородавками (наростами); они будут заменены на более систематизированные, разбитые на предоставляемые ими функции или устанавливаемые ими параметры среды.

AC_AIX Macro

На AIX определяет переменную `_ALL_SOURCE`. Позволяет использовать некоторые функции BSD. Должен вызываться до макросов, запускающих компилятор C.

AC_DYNIX_SEQ Macro

На Dynix/PTX (Sequent UNIX) добавляет '`-lseq`' к выходной переменной `LIBS`. Этот макрос является устаревшим; используйте вместо него `AC_FUNC_GETMNTENT`.

AC_IRIX_SUN Macro

На IRIX (Silicon Graphics UNIX) добавляет '`-lsun`' к выходной переменной `LIBS`. Этот макрос является устаревшим. Если вы используете его для проверки наличия `getmntent`, то вместо него используйте макрос `AC_FUNC_GETMNTENT`. Если вы использовали его для NIS-версий функций работы с паролями и группами, то вместо него используйте '`AC_CHECK_LIB(sun, getpwnam)`'.

AC_ISC_POSIX Macro

На POSIX-версии ISC UNIX определяет переменную `_POSIX_SOURCE` и добавляет '`-posix`' (для компилятора GNU C) или '`-Xp`' (для других компиляторов C) к выходной переменной `CC`. Это позволяет использовать возможности POSIX. Макрос должен быть вызван после вызова `AC_PROG_CC` и до вызова любых других макросов, которые запускают компилятор C.

AC_MINIX Macro

На Minix определяет переменные `_MINIX` и `_POSIX_SOURCE` и определяет `_POSIX_1_SOURCE` со значением 2. Это позволяет использовать возможности POSIX. Должен вызываться до вызова других макросов, запускающих компилятор C.

AC_SCO_INTL Macro

На SCO UNIX добавляет ‘`-lintl`’ к выходной переменной `LIBS`. Этот макрос является устаревшим; вместо него используйте макрос `AC_FUNC_STRFTIME`.

AC_XENIX_DIR Macro

На Xenix добавляет ‘`-lx`’ к выходной переменной `LIBS`. Также, если используется ‘`dirent.h`’, то к переменной `LIBS` добавляется ‘`-ldir`’. Этот макрос является устаревшим; вместо него используйте `AC_HEADER_DIRENT`.

5 Написание тестов

Если существующие тесты делают не то, что вам надо, то вы можете написать собственные тесты. Эти макросы являются строительными блоками для этих тестов. Они предоставляют другим макросам возможность проверить доступность различных свойств и сообщить о результатах.

Эта глава содержит некоторые пожелания и описание того, почему существующие тесты написаны так, а не иначе. Вы также можете научиться тому, как писать тесты Autoconf, разглядывая существующие тесты. Если в одном или нескольких тестах Autoconf что-нибудь пойдет не так, то эта информация может помочь вам понять предположения, которые стоят за ними, что, в свою очередь, может помочь вам определить наилучший способ решения проблемы.

Эти макросы проверяют вывод системного компилятора C. Они не кэшируют результаты тестов для последующего использования (см. Раздел 6.3 [Кэширование результатов], с. 57), поскольку для генерации имени переменной кэша они не имеют достаточно информации о том, что они проверяют. Также по некоторым причинам они не печатают никаких сообщений. Проверки отдельных свойств компилятора C вызывают эти макросы и кэшируют свои результаты, а также выводят сообщения о том, что они проверяют.

Когда вы пишете тест свойства, который может быть применим для более чем одного пакета программного обеспечения, то лучше всего будет описать его как новый макрос. Для того, чтобы узнать, как это делается См. Глава 7 [Создание макросов], с. 62.

5.1 Исследование деклараций

Макрос `AC_TRY_CPP` используется для проверки существования конкретных заголовочных файлов. You can check for one at a time, or more than one if you need several header files to all exist for some purpose.

AC_TRY_CPP (*includes*, [*action-if-true* [, *action-if-false*]]) Macro

includes содержит директивы `#include` языков C или C++, а также объявления, над которыми выполняются подстановки переменных командного процессора, обратных кавычек и обратных слэшей. (В действительности, *includes* может быть любой программой на C, но другие выражения, вероятно, бесполезны). Если препроцессор не выдает сообщений об ошибках в течении обработки директивы, то выполняется код командного процессора *action-if-true*. В противном случае выполняется код *action-if-false*.

Этот макрос использует переменную `CPPFLAGS`, а не `CFLAGS`, поскольку ‘`-g`’, ‘`-O`’ и т. п. не являются правильными ключами для многих препроцессоров C.

Вот как узнать, содержит ли конкретный заголовочный файл определенное объявление, например, объявление типа, структуры, члена структуры или функции. Используйте макрос `AC_EGREP_HEADER` вместо прямого запуска команды `grep` для заголовочного файла; в некоторых системах символ может быть объявлен в другом заголовочном файле, а не в том, который вы проверяете в ‘`#include`’.

AC_EGREP_HEADER (*pattern, header-file, action-if-found* [, *action-if-not-found*]) Macro

Если вывод препроцессора, запущенного для системного заголовочного файла *header-file* соответствует регулярному выражению `egrep` *pattern*, то выполняются команды командного процессора *action-if-found*, в противном случае выполняются команды *action-if-not-found*.

Для проверки символов препроцессора C, определенных в заголовочном файле, либо предопределенных препроцессором C, используйте макрос `AC_EGREP_CPP`. Вот пример последнего:

```
AC_EGREP_CPP(yes,
[#ifdef _AIX
  yes
#endif
], is_aix=yes, is_aix=no)
```

AC_EGREP_CPP (*pattern, program,* [*action-if-found* [, *action-if-not-found*]]) Macro

program является текстом программы на C или C++, для которой выполняются подстановки переменных командного процессора, обратных кавычек и обратных слэшей. Если вывод препроцессора, обрабатывавшего *program*, соответствует регулярному выражению команды `egrep` *pattern*, то выполняется код командного процессора *action-if-found*, иначе выполняется *action-if-not-found*.

Этот макрос вызывает `AC_PROG_CPP` или `AC_PROG_CXXCPP` (в зависимости от того, какой из языков является текущим, см. Раздел 5.8 [Выбор языка], с. 53), если эти макросы еще не вызывались.

5.2 Проверка синтаксиса

Для проверки синтаксических возможностей компиляторов C, C++ или Fortran 77, например, распознавания определенных ключевых слов, используется макрос `AC_TRY_COMPILE`, который пробует откомпилировать маленькую программу, которая использует заданную возможность. Вы также можете использовать этот макрос для проверки структур и полей структур, которые присутствуют не во всех системах.

AC_TRY_COMPILE (*includes, function-body,* [*action-if-found* [, *action-if-not-found*]]) Macro

Создает тестовую программу на C, C++ или Fortran 77 (в зависимости от того, какой язык является текущим, см. Раздел 5.8 [Выбор языка], с. 53), для того, чтобы убедиться, что функция, чье тело состоит из *function-body* может быть скомпилирована.

Для C и C++, *includes* является любыми директивами `#include`, в которых нуждается код в *function-body* (параметр *includes* будет проигнорирован, если текущим языком является Fortran 77). Этот макрос при компиляции помимо переменной `CPPFLAGS` также использует переменные `CFLAGS` или `CXXFLAGS`, если текущим языком является C или C++. Переменная `FFLAGS` будет использована при компиляции, если текущим языком является Fortran 77.

Если файл компилируется нормально, то выполняются команды *action-if-found*, иначе выполняется *action-if-not-found*.

Этот макрос не пытается выполнить компоновку программы – для этого вам придется использовать макрос `AC_TRY_LINK` (см. Раздел 5.3 [Проверка библиотек], с. 47).

5.3 Проверка библиотек

Для проверки библиотеки, функции или глобальной переменной скрипт `configure` попытается скомпилировать и скомпоновать небольшую программу, которая использует тестируемые возможности. Этим Autoconf отличается от Metaconfig, который обрабатывает файлы библиотеки C, используя `nm` или `ar`, чтобы определить, какие функции доступны. Попытка скомпоновать программу с функцией – более надежный вариант, поскольку он избавляет от необходимости обрабатывать различные ключи командной строки и форматы выдачи результатов программ `nm` и `ar`, а также выяснять расположение стандартных библиотек. Этот подход также позволяет конфигурировать кросс-компиляцию, а также проверять поведение функции во время выполнения. С другой стороны, этот подход может оказаться значительно более медленным, чем однократное сканирование библиотек.

Компоновщики в нескольких существующих системах не возвращают статус ошибки, если не могут найти какие-либо символы при компоновке. Эта ошибка делает невозможным использование на таких системах скриптов настройки, созданных Autoconf. Однако, некоторым из них могут быть заданы ключи, которые позволяют получить правильный статус завершения работы. Эту проблему в настоящий момент Autoconf не может обработать автоматически. Если пользователь столкнется с таким, то он может решить эту проблему установкой переменной среды `LDFLAGS`, передавая компоновщику необходимые ключи командной строки (например, '`-Wl,-dn`' на MIPS RISC/OS).

Макрос `AC_TRY_LINK` используется для компиляции тестовой программы для проверки функций и глобальных переменных. Он также используется макросом `AC_CHECK_LIB` для проверки библиотек (см. Раздел 4.2 [Библиотеки], с. 27), временно добавляя проверяемую библиотеку в переменную `LIBS` и пытаясь скомпоновать маленькую программу.

AC_TRY_LINK (*includes, function-body,* [*action-if-found* [, *action-if-not-found*]]) Macro

В зависимости от текущего языка (см. Раздел 5.8 [Выбор языка], с. 53), создается тестовая программа, для того чтобы выяснить, может ли быть скомпилирована и скомпонована функция, чье тело состоит из аргумента *function-body*.

Для C и C++, *includes* является любыми директивами `#include`, в которых нуждается код в *function-body* (параметр *includes* будет проигнорирован, если текущим языком является Fortran 77). Этот макрос при компиляции помимо переменной `CPPFLAGS` также использует переменные `CFLAGS` или `CXXFLAGS`, если текущим языком является C или C++. Переменная `FFLAGS` будет использована при компиляции, если текущим языком является Fortran 77. Однако в любом случае при компоновке будут использованы переменные `LDFLAGS` и `LIBS`.

Если файл компилируется и компонуется, то выполняются команды *action-if-found*, в противном случае — *action-if-not-found*.

AC_TRY_LINK_FUNC (*function*, [*action-if-found* [, *action-if-not-found*]]) Macro

В зависимости от текущего языка см. Раздел 5.8 [Выбор языка], с. 53), создается тестовая программа для того, чтобы убедиться, что программа, чье тело состоит их прототипа и вызова *function*, может быть скомпилирована и скомпонована.

Если файл компилируется и компонуется без ошибок, то выполняется код *action-if-found*, в противном случае выполняется *action-if-not-found*.

AC_TRY_LINK_FUNC (*function*, [*action-if-found* [, *action-if-not-found*]]) Macro

Этот макрос пробует скомпилировать и скомпоновать маленькую программу, которая компонуется с *function*. Если файл компилируется и компонуется без ошибок, то запускается код командного процессора *action-if-found*, в противном случае выполняется *action-if-not-found*.

AC_COMPILE_CHECK (*echo-text*, *includes*, *function-body*, *action-if-found* [, *action-if-not-found*]) Macro

Этот макрос является устаревшей версией `AC_TRY_LINK`. Он отличается тем, что выдает сообщение '`checking for` *echo-text*' в поток стандартного вывода, в том случае, если аргумент *echo-text* не является пустым. Вместо этого макроса для выдачи сообщений используйте `AC_MSG_CHECKING` и `AC_MSG_RESULT` (см. Раздел 6.4 [Выдача сообщений], с. 60).

5.4 Проверка поведения во время выполнения

Иногда вам необходимо определить, как система работает во время выполнения программы, например, имеет ли заданная функция определенные возможности или ошибки. Старайтесь выполнять такие проверки непосредственно во время выполнения программы, а не во время конфигурирования. Такие вещи, как расположение байтов в памяти машины также следует проверять при инициализации программы.

Если вам действительно необходимо протестировать поведение программы во время выполнения при конфигурировании, то вы можете написать тестовую программу для определения результатов, откомпилировать и запустить ее с помощью макроса `AC_TRY_RUN`. Если возможно, избегайте запуска тестовых программ, поскольку их использование мешает пользователям настраивать ваш пакет для кросс-компиляции.

5.4.1 Запуск тестовых программ

Используйте нижеописанный макрос, если вам нужно при конфигурировании протестировать поведение системы во время исполнения.

AC_TRY_RUN (*program*, [*action-if-true* [, *action-if-false* [, *action-if-cross-compiling*]]]) Macro

Аргумент *program* является текстом программы на языке C, для которой выполняются подстановки переменных командного процессора, а также обратных кавычек и обратных слэшей. Если она компилируется и компонуется, и при выполнении возвращает код завершения 0, то выполняется код командного процессора *action-if-true*. В противном случае выполняются команды *action-if-false*; код завершения тестовой программы доступен в переменной командного процессора '`$?`'. При компиляции этот макрос использует переменные `CFLAGS` или `CXXFLAGS`, `CPPFLAGS`, `LDFLAGS` и `LIBS`.

Если используемый компилятор C не создает исполняемых файлов, которые запускаются на той же системе, где выполняется скрипт `configure`, то тестовая программа не запускается. Если задан аргумент *action-if-cross-compiling*, то вместо программы запускается код, заданный в этом аргументе. В противном случае `configure` выдает сообщение об ошибке и прекращает работу.

Постарайтесь сделать значения по умолчанию пессимистическими, если кросс-компиляция не позволяет проверить поведение времени выполнения. Это можно сделать, передав макросу `AC_TRY_RUN` необязательный последний аргумент. `autoconf` выдает предупреждающее сообщение при создании `configure` каждый раз, когда встречается вызов макроса `AC_TRY_RUN` с незаданным аргументом *action-if-cross-compiling*. Вы можете игнорировать это предупреждение, хотя пользователи не смогут настроить ваш пакет для кросс-компиляции. Несколько макросов, поставляемых в составе Autoconf, выдают это предупреждающее сообщение.

Для конфигурирования для кросс-компиляции вы также можете выбрать значения параметров, основываясь на каноническом имени системы (см. Глава 8 [Ручная настройка], с. 67). В качестве альтернативы, вы можете установить правильное значение для целевой системы в кэш-файле с результатами тестов (см. Раздел 6.3 [Кэширование результатов], с. 57).

Для задания значений по умолчанию для вызовов макроса `AC_TRY_RUN`, которые включены в другие макросы (включая те, которые поставляются с Autoconf), вы можете вызвать макрос `AC_PROG_CC` до их вызова. Затем, если переменная командного процессора `cross_compiling` имеет значение '`yes`', то используется альтернативный метод для получения результатов, вместо вызова макросов.

AC_C_CROSS Macro

Этот макрос является устаревшим; он ничего не делает.

5.4.2 Рекомендации по написанию тестовых программ

Тестовые программы не должны выдавать никаких сообщений на поток стандартного вывода. Они должны возвращать значение 0 в случае удачи и ненулевое значение — в противном случае, так что удачное выполнение можно легко отличить от выдачи дампа при крахе программы или другого неудачного выполнения; нарушение доступа к памяти и другие сбои возвращают ненулевой статус завершения. Тестовые программы должны завершать работу с помощью вызова функции `exit`, а

не с помощью оператора `return` из подпрограммы `main`, поскольку на некоторых системах (по крайней мере, на старых машинах Sun) в подпрограмме `main` игнорируется аргумент оператора `return`.

Тестовые программы могут использовать директивы `#if` или `#ifdef` для проверки значений макросов препроцессора, определенных уже проведенными тестами. Например, если вы вызовете `AC_HEADER_STDC`, то далее в '`configure.in`' можно использовать тестовую программу, которая в зависимости от условия включает заголовочные файлы ANSI C:

```
#if STDC_HEADERS
# include <stdlib.h>
#endif
```

Если тестовой программе нужно использовать или создать файл данных, то задавайте этому файлу имя, которое начинаются с '`conftest`', например, '`conftestdata`'. Скрипт `configure` после выполнения тестовых программ а также в случае прерывания работы скрипта удаляет эти файлы с помощью команды '`rm -rf conftest*`'.

5.4.3 Тестовые функции

Объявления функций в тестовой программе должны быть с помощью условной компиляции объявлены как для компилятора C++, так и для компилятора C. На практике, однако, тестовые программы редко нуждаются в функциях, которым передаются аргументы.

```
#ifdef __cplusplus
foo(int i)
#else
foo(i) int i;
#endif
```

Функции, которые объявляются в тестовых программах, должны быть также объявлены с применением прототипов '`extern "C"`', для использования с компиляторами C++. Убедитесь, что вы не включаете заголовочные файлы, содержащие конфликтующие прототипы.

```
#ifdef __cplusplus
extern "C" void *malloc(size_t);
#else
char *malloc();
#endif
```

Если тестовая программа вызывает функцию с неправильными параметрами (просто чтобы убедиться, что такая существует), то организуйте программу таким образом, чтобы эта функция никогда не была вызвана. Это можно сделать путем вызова ее в другой функции, которая никогда не вызывается. Вы не можете сделать это, поместив вызов функции после вызова функции `exit`, поскольку GCC версии 2 знает о том, что функция `exit` никогда не возвращается в точку вызова, и оптимизирует любой код, который следует за ней в том же блоке.

Если вы включаете какой-либо заголовочный файл, то убедитесь, что функции, находящиеся в этих файлах, вызываются с правильным числом параметров, даже если все эти параметры равны нулю. Это нужно, чтобы избежать ошибок компиляции из-за

несоответствия прототипов. GCC версии 2 имеет внутренние прототипы нескольких функций, которые он встраивает в код автоматически; например, к таким относится `memcpy`. Для того, чтобы избежать ошибок при их проверке, либо передавайте этим функциям правильное количество аргументов, либо повторно объявите эти функции с другим типом возвращаемого значения (например, как `char`).

5.5 Переносимое программирование на языке командного процессора

Есть определенные техники программирования скриптов командного процессора, которых вам следует избегать, чтобы ваш код был переносим. Bourne shell и совместимые с ним процессора, такие как Bash и Korn, развивались в течении многих лет, но для того, чтобы избежать трудностей, не используйте возможностей, которые были добавлены после выпуска UNIX версии 7, примерно в 1977 году. Вы не должны использовать функции командного процессора, псевдонимы (aliases), отрицательные классы символов и другие возможности, которые присутствуют не во всех версиях командных процессоров, совместимых с процессором Bourne; ограничьте себя общим знаменателем. Даже `unset` не поддерживается всеми командными процессорами! При указании интерпретатора ставьте пробел после символов '`#!`', например,

```
#! /usr/bin/perl
```

Если вы уберете пробел перед путевым именем, то системы типа 4.2BSD, такие как Sequent DYNIX, будут просто игнорировать эту строку, поскольку они интерпретируют '`#! /`' как 4-х байтовое магическое число.

Набор внешних программ, которые можно запускать из скрипта `configure`, довольно мал. См. раздел "Utilities in Makefiles" в *GNU Coding Standards*, ниже приведен список этих программ. Это ограничение позволяет пользователям начать с небольшого количества программ, постепенно компилируя остальные, и избежать слишком большого числа зависимостей между пакетами.

Многие такие внешние утилиты обладают общим подмножеством переносимых возможностей; например, не полагайтесь на то, что команда `ln` имеет ключ '`-f`', а `cat` вообще имеет какие-либо ключи. Скрипты `sed` не должны содержать комментариев или использовать метки длиннее 8 символов. Не используйте '`grep -s`' для запрещения вывода, поскольку '`grep -s`' на System V не запрещает вывод, а запрещает только сообщения об ошибках. Вместо этого ключа лучше перенаправьте стандартные потоки вывода и сообщений об ошибках (сообщения о несуществующих файлах) программы `grep` на устройство '`/dev/null`'. Проверяйте код возврата `grep`, чтобы узнать, произошло ли совпадение.

5.6 Тестирование значений и файлов

Скрипты `configure` должны проверять различные свойства разных файлов и строк. Вот небольшой список проблем с переносимостью, которых нужно избегать при написании проверок.

Программа `test` используется для выполнения многих проверок файлов и строк. Она часто запускается альтернативным способом, через имя '`[`', но использование

этого имени в коде Autoconf приведет к ошибкам, потому что этот символ является символом кавычек в `m4`.

Если вам необходимо выполнить несколько проверок, используя команду `test`, то объединяйте их с помощью операторов командного процессора '`&&`' и '`||`', а не используйте операторы программы `test` '`-a`' и '`-o`'. На System V приоритеты операторов '`-a`' и '`-o`' неправильно соотносятся с приоритетами унарных операторов; из-за этого POSIX не определяет эти операторы, так что их использование приводит к непереносимому коду. Если вы в одном выражении используете как '`&&`', так и '`||`', то помните, что они имеют одинаковый приоритет.

Скрипты `configure`, поддерживающие кросс-компиляцию, не должны не делать ничего, что тестирует свойства системы, на которой выполняется скрипт. Но иногда вам может понадобиться проверить, существует ли определенный файл. Чтобы сделать это используйте команды '`test -f`' или '`test -r`'. Не используйте команду '`test -x`', поскольку 4.3BSD не поддерживает ее.

Другой непереносимой конструкцией программирования командного процессора является

var`=${`*var*`:-`*value*`}`

Она предназначена для установки значения переменной *var* равным *value*, но только в тех случаях, когда переменная еще не имеет значения. Если *var* уже было присвоено значение, даже равное пустой строке, то оно остается неизменным. Старые командные процессоры BSD, включая Ultrix-версию `sh`, не воспринимают символ двоеточия, выдают ошибку и прекращают работу. Переносимым эквивалентом данной конструкции является

`: ${`*var*`=`*value*`}`

5.7 Множество вариантов

Некоторые операции выполняются несколькими разными способами в зависимости от используемого варианта UNIX. Их проверка требует "оператора выбора". Autoconf напрямую не обеспечивает такой оператор, однако достаточно легко эмулировать его, используя переменную командного процессора для запоминания, найден ли уже пригодный способ.

Вот пример, который использует переменную `fstype` для отслеживания того, остались ли варианты, которые необходимо проверить.

```
AC_MSG_CHECKING(как получить тип файловой системы)
fstype=no
# Порядок этих действий является важным.
AC_TRY_CPP([#include <sys/statvfs.h>
#include <sys/fstyp.h>], AC_DEFINE(FSTYPE_STATVFS) fstype=SVR4)
if test $fstype = no; then
AC_TRY_CPP([#include <sys/statfs.h>
#include <sys/fstyp.h>], AC_DEFINE(FSTYPE_USG_STATFS) fstype=SVR3)
fi
if test $fstype = no; then
AC_TRY_CPP([#include <sys/statfs.h>
#include <sys/vmount.h>], AC_DEFINE(FSTYPE_AIX_STATFS) fstype=AIX)
fi
# (остальные варианты пропущены в этом примере)
AC_MSG_RESULT($fstype)
```

5.8 Выбор языка

Пакеты, использующие одновременно и C, и C++, нуждаются в проверке возможностей обоих компиляторов. Созданные Autoconf скрипты `configure` по умолчанию выполняют проверку возможностей компилятора C. Нижеописанные макросы определяют, компилятор какого языка будет использоваться в тестах, которые последуют за вызовом этого макроса в '`configure.in`'.

AC_LANG_C Macro

Выполняет тесты компиляции, используя переменные `CC` и `CPP`, а также используя расширение '`.c`' для тестовых программ. Устанавливает переменную командного процессора `cross_compiling` в значение, вычисленное макросом `AC_PROG_CC`, если он был запущен, и в пустое значение в противном случае.

AC_LANG_CPLUSPLUS Macro

Выполняет тесты компиляции, используя переменные `CXX` и `CXXPP`, а также используя расширение '`.C`' для тестовых программ. Устанавливает переменную командного процессора `cross_compiling` в значение, вычисленное макросом `AC_PROG_CXX`, если он был запущен, и в пустое значение в противном случае.

AC_LANG_FORTRAN77 Macro

Выполняет тесты компиляции, используя переменную `F77`, а также используя расширение '`.f`' для тестовых программ. Устанавливает переменную командного процессора `cross_compiling` в значение, вычисленное макросом `AC_PROG_F77`, если он был запущен, и в пустое значение в противном случае.

AC_LANG_SAVE Macro

Запоминает в стеке значение текущего языка (установленное макросами `AC_LANG_C`, `AC_LANG_CPLUSPLUS` или `AC_LANG_FORTRAN77`). Не изменяет значение текущего языка. Используйте этот макрос и `AC_LANG_RESTORE` в макросах, которым необходимо временно переключиться на конкретный язык.

AC_LANG_RESTORE Macro

Выбирает язык, который был сохранен на вершине стека, где он был сохранен макросом `AC_LANG_SAVE`, и удаляет его со стека. Этот макрос эквивалентен вызову `AC_LANG_C`, `AC_LANG_CPLUSPLUS` или `AC_LANG_FORTRAN77`, в зависимости от того, который из них действовал во время последнего вызова макроса `AC_LANG_SAVE`.

Не вызывайте этот макрос больше раз, чем было вызовов `AC_LANG_SAVE`.

AC_REQUIRE_CPP Macro

Убеждается, что препроцессор, который должен сейчас использоваться, был найден. Вызывает макрос `AC_REQUIRE` (см. Раздел 7.4.1 [Требуемые макросы], с. 64) с аргументом, равным либо `AC_PROG_CPP`, либо `AC_PROG_CXXCPP`, в зависимости от того, какой язык был выбран.

6 Результаты тестов

После того, как скрипт `configure` выяснил существование какой-либо возможности, что можно сделать, чтобы записать эту информацию? Есть четыре варианта: определить символ препроцессора C, установить переменную в выходном файле, сохранить результат в кэш-файле для использования при последующих запусках `configure` и выдать сообщение, позволяющее пользователю узнать результат теста.

6.1 Определение символов препроцессора C

Обычно после проверки какой-либо возможности устанавливается символ препроцессора, отражающий результат проверки. Это происходит при вызове макроса `AC_DEFINE` или `AC_DEFINE_UNQUOTED`.

По умолчанию макрос `AC_OUTPUT` помещает символы, определенные этими макросами в выходную переменную `DEFS`, которая по одному ключу '`-D`*symbol*`=`*value*' на каждый символ. В отличии от Autoconf версии 1, переменная `DEFS` не определена в течении работы `configure`. Для проверки того, определен ли уже какой-либо символ препроцессора C, проверьте значение соответствующей переменной кэша, как показано в этом примере:

```
AC_CHECK_FUNC(vprintf, AC_DEFINE(HAVE_VPRINTF))
if test "$ac_cv_func_vprintf" != yes; then
AC_CHECK_FUNC(_doprnt, AC_DEFINE(HAVE_DOPRNT))
fi
```

Если был вызван макрос `AC_CONFIG_HEADER`, то `AC_OUTPUT` вместо определения переменной `DEFS` создает заголовочный файл путем подстановки правильных значений в директивы `#define`, заданные в файле-шаблоне. См. Раздел 3.4 [Заголовочные файлы конфигурации], с. 16, для дополнительной информации об этом способе вывода результатов.

AC_DEFINE (*variable* [, *value* [, *description*]]) Macro

Определяет переменную препроцессора C *variable*. Если аргумент *value* задан, то устанавливает переменную *variable* в это значение (без изменений), в противном случае устанавливает ее равной 1. *value* не должен содержать символов перевода строки, а если вы не используете `AC_CONFIG_HEADER`, то он не должен содержать символы '`#`', поскольку `make` имеет склонен проглатывать их. Для использования переменной командного процессора (которая необходима, если нужно определить значение, содержащее символ, являющийся кавычкой в `m4` '`[`' или '`]`') вам следует использовать `AC_DEFINE_UNQUOTED`. Аргумент *description* полезен только в том случае, если вы используете макрос `AC_CONFIG_HEADER`. В этом случае *description* будет помещен в созданный файл '`config.h.in`' в качества комментария к определению символа; макросу не нужно быть упомянутым в '`acconfig.h`'. Следующий пример определяет переменную препроцессора C `EQUATION` со значением, равным строковой константе '`"$a > $b"`':

```
AC_DEFINE(EQUATION, "$a > $b")
```

AC_DEFINE_UNQUOTED (*variable* [, *value* [, *description*]]) Macro

Подобно `AC_DEFINE`, но над переменными *variable* и *value* выполняется ряд преобразований: подстановка переменных (‘`$`’), подстановка результатов выполнения команд (‘``’) и экранирование символов обратной косой черты (‘`\`’). Символы одиночных и двойных кавычек в *value* не имеют специального смысла. Используйте этот макрос вместо `AC_DEFINE`, когда *variable* или *value* являются переменными командного процессора. Примеры:

```
AC_DEFINE_UNQUOTED(config_machfile, "${machfile}")
AC_DEFINE_UNQUOTED(GETGROUPS_T, $ac_cv_type_getgroups)
AC_DEFINE_UNQUOTED(${ac_tr_hdr})
```

Из-за синтаксических странностей командного процессора Bourne не следует использовать точку с запятой для разделения вызовов макросов `AC_DEFINE` или `AC_DEFINE_UNQUOTED` от вызова других макросов или кода командного процессора; это может привести к синтаксическим ошибкам в результирующем скрипте `configure`. Вместо этого используйте пробелы или переводы строк. То есть, следует писать так:

```
AC_CHECK_HEADER(elf.h, AC_DEFINE(SVR4) LIBS="$LIBS -lelf")
```

либо так:

```
AC_CHECK_HEADER(elf.h,
  AC_DEFINE(SVR4)
  LIBS="$LIBS -lelf")
```

но не так:

```
AC_CHECK_HEADER(elf.h, AC_DEFINE(SVR4); LIBS="$LIBS -lelf")
```

6.2 Установка выходных переменных

Одним из способов записи результатов тестов является установка *выходных переменных*, то есть переменных командного процессора, чьи значения подставляются в файлы, выводимые `configure`. Нижеприведенные макросы используются для создания новых выходных переменных. См. Раздел 6.2 [Установка выходных переменных], с. 56, где приведен список всегда присутствующих выходных переменных.

AC_SUBST (*variable*) Macro

Создает выходную переменную из переменной командного процессора. Заставляет `AC_OUTPUT` подставлять переменную *variable* в выходные файлы (обычно это один или несколько файлов ‘`Makefile`’). Это означает, что `AC_OUTPUT` будет заменять вхождения ‘`@`*variable*`@`’ во входных файлах на значение переменной командного процессора *variable*, которое она имела при вызове макроса `AC_OUTPUT`. Значение *variable* не должно содержать символы новой строки.

AC_SUBST_FILE (*variable*) Macro

Другой способ создания выходной переменной из переменной командного процессора. Заставляет `AC_OUTPUT` вставить (без подстановок) в выходные файлы содержимое файла, указанного в переменной командного процессора *variable*. Это означает, что `AC_OUTPUT` будет заменять вхождения ‘`@`*variable*`@`’ в выходных файлах (таких как ‘`Makefile.in`’) на содержимое файла, имя которого содержалось

в переменной *variable* в момент вызова макроса `AC_OUTPUT`. Установите значение этой переменной в '`/dev/null`' для случаев, когда вставляемый файл отсутствует.

Этот макрос полезен для вставки фрагментов '`Makefile`', содержащих специальные зависимости или другие директивы `make` для отдельных типов машин и целей в результирующие файлы '`Makefile`'. Например, файл '`configure.in`' может содержать:

```
AC_SUBST_FILE(host_frag)dnl
host_frag=$srcdir/conf/sun4.mh
```

и файл '`Makefile.in`' может содержать:

```
@host_frag@
```

6.3 Кэширование результатов

Чтобы избежать повторяющихся проверок одних и тех же возможностей в различных скриптах `configure` (или при повторных вызовах одного скрипта), `configure` сохраняет результаты многих проверок в *кэш-файле*. Если при запуске скрипта `configure` тот находит кэш-файл, то считывает результаты, полученные при предыдущих запусках, и не выполняет проверки, результат которых уже получен. Благодаря этому `configure` может работать намного быстрее, чем если бы при каждом запуске приходилось заново выполнять все проверки.

AC_CACHE_VAL (*cache-id, commands-to-set-it*) Macro

Проверяет, что доступны результаты проверки, на которые указывает *cache-id*. Если результаты проверки находятся в кэше, и скрипту `configure` не был задан ключ '`-quiet`' или '`-silent`', то выдать сообщение о том, что результаты были взяты из кэша; в противном случае запустить код командного процессора *commands-to-set-it*. Эти команды не должны иметь побочных эффектов, за исключением установки переменной *cache-id*. В частности, они не должны вызывать макрос `AC_DEFINE`; это должен делать код, следующий за вызовом `AC_CACHE_VAL`, основываясь на кэшированном значении. Они также не должны выдавать никаких сообщений, например, с помощью макроса `AC_MSG_CHECKING`; это надо выполнять до вызова `AC_CACHE_VAL`, так что сообщения будут печататься вне зависимости от того, будут ли результаты взяты из кэша или будут определены с помощью выполнения кода командного процессора. Если для определения значения будет запущен код командного процессора, то полученное значение будет сохранено в кэш-файле перед тем, как `configure` будет создавать выходные файлы. См. Раздел 6.3.1 [Имена переменных кэша], с. 58, для того чтобы узнать, как выбрать имя переменной *cache-id*.

AC_CACHE_CHECK (*message, cache-id, commands*) Macro

Обертка для макроса `AC_CACHE_VAL`, которая берет на себя заботу о выдаче сообщений. Этот макрос обеспечивает удобную и короткую форму записи наиболее распространенных способов использования этих макросов. Он вызывает макрос `AC_MSG_CHECKING` для выдачи сообщения *message*, затем вызывает `AC_CACHE_VAL` с аргументами *cache-id* и *commands* и, наконец, вызывает `AC_MSG_RESULT` с аргументом *cache-id*.

AC_CACHE_LOAD Macro

Загружает значения из существующего кэш-файла, или создает новый, если кэш-файл не найден. Автоматически вызывается из макроса `AC_INIT`.

AC_CACHE_SAVE Macro

Записывает все кэшированные значения в кэш-файл. Этот макрос автоматически из макроса `AC_OUTPUT`, но полезно бывает вызывать `AC_CACHE_SAVE` в ключевых точке файла '`configure.in`'. При это кэш сохраняется на тот случай, если работа скрипта '`configure`' будет прервана.

6.3.1 Имена переменных кэша

Имена переменных кэша должны иметь следующий формат:

package-prefix`_cv_`*value-type*`_`*specific-value*[`_`*additional-options*]

Например, '`ac_cv_header_stat_broken`' или '`ac_cv_prog_gcc_traditional`'. Имя переменной состоит из следующих частей:

package-prefix
: Сокращенное название вашего пакета или организации; с такого же префикса вы должны начинать локальные макросы Autoconf, но только здесь этот префикс записывается в нижнем регистре. Макросы, распространяемые с Autoconf, используют префикс '`ac`'.

`_cv_`
: Показывает, что эта переменная командного процессора является кэшированным значением.

value-type
: Соглашение по классификации значений кэша, для создания рациональной системы наименования. Значения, используемые в Autoconf, перечислены в Раздел 7.2 [Имена макросов], с. 62.

specific-value
: Для какого члена класса кэшированных значений применяется данный тест. Например, к какой функции ('`alloca`'), программе ('`gcc`') или выходной переменной ('`INSTALL`').

additional-options
: Конкретное поведение конкретного члена класса, к которому применяется этот тест. Например, '`broken`' ("сломано") или '`set`' ("установлено"). Эта часть имени может быть опущена.

Значения кэшированных переменных не могут содержать переводы строк. Обычно их значения являются логическими значениями ('`yes`' или '`no`') или именами файлов или функций, поэтому это ограничение не критично.

6.3.2 Кэш-файлы

Кэш-файл — это скрипт командного процессора, который хранит результаты тестов конфигурации, проведенных на одной системе, так что они могут совместно использоваться разными скриптами настройки, или при нескольких запусках одного и того же скрипта configure. На других системах этот файл использовать бесполезно.

Если содержимое этого файла по некоторым причинам является неверным, то пользователь может удалить или отредактировать его.

По умолчанию в качестве кэш-файла ‘`configure`’ использует файл ‘`./config.cache`’, создавая его, если он не существует. `configure` распознает ключ командной строки ‘`-cache-file=`*file*’, который позволяет использовать другой кэш-файл; этот ключ используется `configure`, когда он вызывает скрипты `configure`, находящиеся в подкаталогах, так что они используют общий кэш. См. Раздел 3.5 [Подкаталоги], с. 19, где описано, как задавать подкаталоги с помощью макроса `AC_CONFIG_SUBDIRS`.

Использование ключа ‘`-cache-file=/dev/null`’ запрещает кэширование, например, для отладки `configure`. Скрипт ‘`config.status`’ смотрит на кэш-файл только если запустить его с ключом ‘`-recheck`’, чтобы повторно выполнить `configure`. Если вы предчувствуете долгий период отладки, то можете запретить загрузку и сохранение кэша путем переопределения макросов работы с кэшем в начале ‘`configure.in`’:

```
define([AC_CACHE_LOAD], )dnl
define([AC_CACHE_SAVE], )dnl
AC_INIT(whatever)
 ... rest of configure.in ...
```

Попытка распространения кэш-файлов для определенных типов систем неверна в корне. Пытаясь сделать это, вы получаете полную свободу совершать ошибки, а также сталкиваетесь с массой административных проблем, связанных с поддержкой этих файлов. Для возможностей, которые нельзя определить автоматически, используйте стандартный способ канонического типа системы и компоновки файлов (см. Глава 8 [Ручная настройка], с. 67).

На отдельной системе кэш-файл постепенно будет накапливать результаты запусков скрипта `configure`; первоначально он вообще не будет существовать. Запуск `configure` объединяет новые кэшированные результаты с уже существующим кэш-файлом. Для того, чтобы сделать работу скрипта более простой, скрипт инициализации на данной машине, может указать общесистемный кэш-файл, который будет использоваться вместо используемого по умолчанию, поскольку каждый раз используется один и тот же компилятор C (см. Раздел 9.5 [Локальные значения по умолчанию], с. 74).

Если ваш скрипт, или макрос, вызываемые из ‘`configure.in`’, прерывает процесс настройки, то полезно будет сохранить кэшированные данные несколько раз в ключевых точках скрипта. Делая это, вы уменьшите время, затраченное при перезапуске скрипта конфигурации после исправления ошибки, которая вызвала предыдущий останов работы.

```
 ... AC_INIT, etc. ...
dnl проверки программ
AC_PROG_CC
AC_PROG_GCC_TRADITIONAL
 ... дополнительные проверки программ...
AC_CACHE_SAVE

dnl проверка библиотек
AC_CHECK_LIB(nsl, gethostbyname)
```

```
AC_CHECK_LIB(socket, connect)
 ... другие проверки библиотек ...
AC_CACHE_SAVE

dnl Might abort...
AM_PATH_GTK(1.0.2, , exit 1)
AM_PATH_GTKMM(0.9.5, , exit 1)
```

6.4 Выдача сообщений

Скрипты `configure` должны сообщать пользователям различную информацию. Следующие макросы различными способами выдают сообщения. Аргументы для каждого из макросов помещаются в двойные кавычки, используемые командным процессором, так что в этих аргументах будет выполняться подстановка переменных и специальных символов. Вы можете напечатать сообщение, содержащее запятую, поместив сообщение в кавычки, используемые программой `m4`:

```
AC_MSG_RESULT([never mind, I found the BASIC compiler])
```

Все эти макросы являются надстройками над командой `echo`. Скрипты `configure` должны крайне редко использовать команду `echo` для выдачи сообщения пользователю. Использование этих макросов позволяет легко изменить способ, каким выдается каждый из типов сообщений; такие изменения можно будет внести в определение макроса, и все вызовы этого макроса изменят свой вид автоматически.

AC_MSG_CHECKING (*feature-description*) Macro

Уведомляет пользователя о том, что `configure` проверяет конкретную возможность. Этот макрос печатает сообщение, которое начинается с '`checking `' и заканчивается '`...`' без перехода на новую строку. Вслед за вызовом этого макроса следует использовать макрос `AC_MSG_RESULT`, который выдает результат проверки и символ перевода строки. Аргумент *feature-description* должен содержать что-нибудь типа '`понимает ли компилятор Fortran комментарии в стиле C++`' или '`for c89`'.

Этот макрос ничего не выводит, если `configure` запущен с ключами '`-quiet`' или '`-silent`'.

AC_MSG_RESULT (*result-description*) Macro

Уведомляет пользователя о результатах проверки. Аргумент *result-description* почти всегда содержит значение переменной кэша для данного теста, обычно равно '`yes`', '`no`' или имени файла. Этот макрос должен вызываться после вызова `AC_MSG_CHECKING` и аргумент *result-description* по смыслу должен дополнять сообщение выданное вызовом `AC_MSG_CHECKING`.

Этот макрос ничего не выводит, сели `configure` запущен с ключами '`-quiet`' или '`-silent`'.

AC_MSG_ERROR (*error-description*) Macro

Уведомляет пользователя об ошибке, которая препятствует работе `configure`. Этот макрос печатает сообщение об ошибке в стандартный поток вывода и за-

канчивает работу `configure` с ненулевым статусом. Аргумент *error-description* должен содержать что-то подобное '`неправильное значение $HOME для \$HOME`'.

AC_MSG_WARN (*problem-description*) Macro

Уведомляет пользователя `configure` о возможной проблеме. Этот макрос выдает сообщение в стандартный поток сообщений об ошибках; после этого `configure` продолжает свое выполнение, так что макросы, вызвавший `AC_MSG_WARN` должен предоставить действия по умолчанию для ситуаций, о которых он выдавал предупреждения. Аргумент *problem-description* должен содержать что-то подобное '`кажется ln -s создает жесткие ссылки`'.

Следующие два макроса являются устаревшими и являются альтернативой для макросов `AC_MSG_CHECKING` и `AC_MSG_RESULT`.

AC_CHECKING (*feature-description*) Macro

Этот макрос подобен `AC_MSG_CHECKING`, за исключением того, что он выдает символ перевода строки после вывода *feature-description*. Он в основном полезен для выдачи общего описания группы проверок, например:

```
AC_CHECKING(if stack overflow is detectable)
```

AC_VERBOSE (*result-description*) Macro

Этот макрос подобен `AC_MSG_RESULT`, за исключением того, что его вызов следует за вызовом `AC_CHECKING`, а не `AC_MSG_CHECKING`; выдаваемое им сообщение начинается с символа табуляции. Этот макрос считается устаревшим.

7 Создание макросов

Когда вы пишете тест свойства, который будет применяться в более чем одном пакете программного обеспечения, то лучше всего оформить его в виде нового макроса. В этом разделе приводятся некоторые инструкции и указания по написанию макросов Autoconf.

7.1 Определение макросов

Макросы Autoconf определяются с помощью макроса `AC_DEFUN`, который подобен встроенному макросу `define` программы `m4`. В добавление к определению макроса, `AC_DEFUN` добавляет к нему некоторый код, который используется для ограничения порядка вызовы макросов (см. Раздел 7.4.1 [Требуемые макросы], с. 64).

Определение макроса Autoconf выглядит примерно следующим образом:

`AC_DEFUN(`*macro-name*`, [`*macro-body*`])`

Квадратные скобки не показывают необязательный параметр: они должны присутствовать в определении макроса для избежания проблем расширения макроса (см. Раздел 7.3 [Заключение в кавычки], с. 63). Вы можете ссылаться на передаваемые макросу параметры с помощью переменных ‘`$1`’, ‘`$2`’ и т.п.

Для ввода комментариев в `m4`, используйте встроенный макрос `m4 dnl`; он заставляет `m4` игнорировать текст до начала новой строки. Он не нужен между определениями макросов в файлах ‘`acsite.m4`’ и ‘`aclocal.m4`’, поскольку весь вывод удаляется до вызова `AC_INIT`.

См. раздел “How to define new macros” в *GNU m4*, для более полной информации о написании макросов `m4`.

7.2 Имена макросов

Все макросы Autoconf названы именами, состоящими из букв заглавных букв и начинающихся с префикса ‘`AC_`’, для того, чтобы избежать конфликтов с другим текстом. Все переменные командного процессора, которые используются для внутренних целей в этих макросах, как правило называются именами из прописных букв и начинаются с ‘`ac_`’. Чтобы обеспечить, что ваши макросы не конфликтовали с существующими или будущими макросами Autoconf, вы должны использовать собственный префикс для ваших макросов и переменных командного процессора. В качестве возможных значений вы можете использовать ваши инициалы, или сокращенное название вашей организации или пакета программ.

Большинство имен макросов Autoconf следуют соглашению о структуре имени, которое показывает какой тип свойства проверяемого данным макросом. Имена макросов состоит из нескольких слов, который разделены символами подчеркивания, продвигаясь от общих слов к более специфическим. Имена соответствующих переменных кэша используют то же соглашение по именованию (см. Раздел 6.3.1 [Имена переменных кэша], с. 58, для получения дополнительной информации о них).

Первое слово имени после префикса ‘`AC_`’ обычно сообщает категорию тестируемого свойства. Вот какие категории используются Autoconf для специфических макросов,

один из типов которых вы вероятно захотите написать. Они также используются для именования переменных кэша, только используя прописные буквы. Используйте перечисленные категории при написании ваших макросов; если нужной категории нет, то вы можете вводит собственные.

`C`	Встроенные возможности языка C.
`DECL`	Объявления переменных C в заголовочных файлах.
`FUNC`	Функции в библиотеках.
`GROUP`	Группа UNIX владеющая файлами.
`HEADER`	Заголовочные файлы.
`LIB`	Библиотеки C.
`PATH`	Полные путевые имена файлов, включая программы.
`PROG`	Базовые имена программ.
`STRUCT`	Определения структур C в заголовочных файлах.
`SYS`	Свойства операционной системы.
`TYPE`	Встроенные или объявленные типы C.
`VAR`	Переменные C в библиотеках.

После категории следует имя тестируемого свойства. Любые дополнительные слова в имени макроса указывают на специфические аспекты тестируемого свойства. Например, `AC_FUNC_UTIME_NULL` проверяет поведение функции `utime` при вызове ее с указателем равным `NULL`.

Макрос, который является внутренней подпрограммой другого макроса должен иметь имя, которое начинается с имени этого макроса, за которым следует одно или несколько слов, описывающих что делает этот макрос. Например, макрос `AC_PATH_X` имеет внутренние макросы `AC_PATH_X_XMKMF` и `AC_PATH_X_DIRECT`.

7.3 Заключение в кавычки

Макросы, которые вызываются другими макросами оцениваются программой `m4` несколько раз; каждая оценка может потребовать другого уровня кавычек для предотвращения нежелательных расширений макросов или встроенных возможностей `m4`, таких как ‘`define`’ и ‘`$1`’. Кавычки также требуются вокруг аргументов макросов, которые содержат запятые, поскольку запятые разделяют аргументы макроса. Также хорошей привычкой является заключение в кавычки аргументов, которые содержат символы новой строки или вызовы других макросов.

Autoconf изменяет символ-кавычку программы `m4` со значений по умолчанию ‘‘’ и ‘’’ на ‘`[`’ и ‘`]`’, поскольку многие из макросов используют не сочетаемые ‘‘’ и ‘’’. Однако в нескольких местах макросам необходимо использовано символов-скобок (обычно в тексте программ на языке C или в регулярных выражениях). В этих местах макросы используют встроенную команду `m4 changequote` для временного изменения символа-кавычек на ‘`<<`’ и ‘`>>`’. (Иногда, если им нет нужды заключать в кавычки что-либо, то они запрещают заключение в кавычки установкой символов-кавычек равных пустым символам). Вот пример использования:

```
AC_TRY_LINK(
changequote(<<, >>)dnl
<<#include <time.h>
#ifndef tzname /* For SGI.  */
extern char *tzname[]; /* RS6000 and others reject char **tzname.  */
#endif>>,
changequote([, ])dnl
[atoi(*tzname);], ac_cv_var_tzname=yes, ac_cv_var_tzname=no)
```

Когда вы создает скрипт `configure`, используя свеже написанные макросы, то тщательно проверьте их на то, нужно ли добавить дополнительные символы-кавычки в эти макросы. Если одно или несколько слов исчезнут в выводе `m4`, то вам необходимо добавить дополнительные символы-кавычки. Если вы сомневаетесь, то просто добавьте кавычки.

Однако также возможно поместить слишком много уровней кавычек. Если это случается, то полученный скрипт `configure` будет содержать не-расширенный макрос. Программа `autoconf` выполняет проверку этой проблемы, выполняя команду '`grep AC_ configure`'.

7.4 Зависимости между макросами

Некоторые макросы Autoconf зависят от других макросов, которые должны быть вызваны первыми для того, чтобы работа производилась правильно. Autoconf предоставляет способ для того, чтобы проверить, что некоторые макросы были вызваны, и способ предоставления пользователю информацию о макросах, которые вызываются в порядке, которые может привести к неправильному выполнению.

7.4.1 Требуемые макросы

Макрос, которое вы можете написать, может нуждаться в значениях, которые перед этим были вычислены другими макросами. Например, `AC_DECL_YYTEXT` проверяет вывод `flex` или `lex`, так что он зависит от `AC_PROG_LEX`, который должен быть вызван перед этим для установки переменной командного процессора `LEX`.

Вместо того, чтобы заставлять пользователя макросов помнить все зависимости между макросами, вы можете использовать макрос `AC_REQUIRE` для того, чтобы автоматически отслеживать зависимости. `AC_REQUIRE` может помочь в обеспечении того, что макрос вызывается только когда это необходимо, и будет вызываться только раз.

AC_REQUIRE (*macro-name*) Macro

Если макрос `m4` с именем *macro-name* еще не был вызван, то необходимо вызвать его (без каких-либо аргументов). Убедитесь, что вы поместили имя *macro-name* в квадратные кавычки. *macro-name* должен быть определен с использованием макроса `AC_DEFUN` или должен содержать вызов макроса `AC_PROVIDE` для того, чтобы указать, что он был вызван.

Альтернативой этому использованию `AC_DEFUN` является использование `define` и вызов макроса `AC_PROVIDE`. Поскольку этот подход не предотвращает вложенных сообщений, то эта техника является устаревшей.

AC_PROVIDE (*this-macro-name*) Macro

Запоминает тот факт, что макрос *this-macro-name* был вызван. *this-macro-name* должен быть именем макроса, который вызывает `AC_PROVIDE`. Для простого получения этого имени используйте встроенную переменную `m4` с именем `$0`, примерно так:

```
AC_PROVIDE([$0])
```

7.4.2 Предлагаемый порядок

Некоторые макросы должны быть вызваны до других макросов, если оба макроса вызываются, но не *требует*, чтобы другие макросы были вызваны. Например, макрос, который изменяет поведение компилятора C должен быть вызван до любого из макросов, которые запускают компилятор C. Многие из этих зависимостей упоминаются в документации.

Autoconf предоставляет макрос `AC_BEFORE` для предупреждения пользователя о тех случаях, когда этот макросы вызываются в неправильном порядке в файле '`configure.in`'. Предупреждение выдается при создании скрипта `configure` из файла '`configure.in`', а не при запуске созданного `configure`. Например, `AC_PROG_CPP` проверяет может ли компилятор C запустить препроцессор C с ключом '`-E`'. Он должен быть вызван после любого из макросов, который изменяет поведение используемого компилятора C, такого как `AC_PROG_CC`. Так что макрос `AC_PROG_CC` должен содержать:

```
AC_BEFORE([$0], [AC_PROG_CPP])dnl
```

Это вызывает выдачу предупреждения пользователю, если вызов `AC_PROG_CPP` уже произошел до вызова макроса `AC_PROG_CC`.

AC_BEFORE (*this-macro-name, called-macro-name*) Macro

Заставляет `m4` выдать предупреждающее сообщение в стандартный поток сообщений об ошибках в том случае, если макроса *called-macro-name* уже был вызван. *this-macro-name* должен быть именем макроса, который вызывает `AC_BEFORE`. Макрос *called-macro-name* должен быть определен используя макрос `AC_DEFUN` или должен содержать вызов `AC_PROVIDE` для того, чтобы показать, что он был вызван.

7.4.3 Устаревшие макросы

Технология настройки и переносимости развивалась многие годы. Часто разрабатывались лучшие решения отдельных проблем, или систематизировались специальные подходы. Этот процесс происходил во многих частях Autoconf. Результатом этого является то, что некоторые макросы в настоящее время считаются *устаревшими*; они до сих пор работают, но не считаются лучшим способом решения. Autoconf предоставляет макрос `AC_OBSOLETE`, который предупреждает пользователей создающих скрипты `configure` о том, что они используют устаревшие макросы, чтобы поощрить их к использованию современных макросов. Простой вызов этого макроса выглядит так:

```
AC_OBSOLETE([$0], [; use AC_CHECK_HEADERS(unistd.h) instead])dnl
```

AC_OBSOLETE (*this-macro-name* [, *suggestion*]) Macro

Заставляет `m4` выдать сообщение в стандартный поток сообщений об ошибках, которое говорит о том, что макрос *this-macro-name* является устаревшим, и выдает имя файла и номер строки где был вызван этот макрос *this-macro-name* должен именем макроса, который производит вызов `AC_OBSOLETE`. Если задан аргумент *suggestion*, то он выдается в конце предупреждающего сообщения; например, он может быть советом о том, что нужно использовать вместо *this-macro-name*.

8 Ручная настройка

Некоторые типы свойств не могут быть определены автоматически путем запуска тестовых программ. Например, детали реализации формата объектных файлов или специальные ключи, которые необходимо передать компилятору или компоновщику. Вы можете проверить такие свойства используя специализированные возможности, такие как заставив `configure` проверить вывод программы `uname` или производя поиск библиотек, специфических для отдельных систем. Однако Autoconf предоставляет однообразный метод для обработки неопределяемых свойств.

8.1 Указание типа системы

Подобно другим скриптам GNU `configure`, созданные Autoconf скрипты `configure` могут делать заключение основываясь на каноническом имени типа системы, которое имеет форму:

cpu-company-system

`configure` обычно может определить каноническое имя типа системы на которой он запущен. Для этого он запускает скрипт с именем `config.guess`, который определяет имя, используя команду `uname` или символы определенные препроцессором C.

В качестве альтернативы, пользователь может указать тип системы как аргумент командной строки скрипта `configure`. Это необходимо сделать, если вы хотите использовать кросс-компиляцию. В большинстве сложных случаев кросс-компиляции будут вовлечены три типа систем. Для их указания используются следующие ключи:

`-build=`*build-type*
: тип системы на которой настраивается и компилируется пакет (используется редко);

`-host=`*host-type*
: тип системы на которой будет запускаться пакет;

`-target=`*target-type*
: тип системы для которой утилиты компилятора будут создавать код.

Если пользователь задает `configure` неключевой аргумент, то он используется как значение по умолчанию для всех типов систем, если только пользователь не указал типы явно для систем с помощью ключей командной строки. Если типы целевой и собирающей систем не заданы, а задан тип системы на которой будет запускаться пакет, то они равны заданному значению. Если вы используете кросс-компиляцию, то вам необходимо указать в командной строке скрипта `configure` имена используемых вами кросс-утилит, в частности компилятора C, например,

```
CC=m68k-coff-gcc configure -target=m68k-coff
```

`configure` распознает короткие алиасы для многих типов систем; например, в командной строке может быть задано имя ‘`decstation`’ вместо ‘`mips-dec-ultrix4.2`’. `configure` запускает скрипт с именем `config.sub` для канонизации алиасов типов систем.

8.2 Получение канонического типа системы

Следующие макросы делают тип системы доступным для скриптов `configure`. Они запускают скрипт командного процессора `config.guess` для определения значений для каждого из типов систем, в которых они нуждаются, и которые пользователь не указал в командной строке. Они запускают `config.sub` для канонизации заданных пользователем псевдонимов. Если вы используете эти макросы, то вы должны распространять эти два файла вместе с вашим исходным кодом. См. Раздел 3.2 [Вывод], с. 10, для получения информации о макросе `AC_CONFIG_AUX_DIR`, который вы можете использовать для контроля того, в каком именно каталоге `configure` будет искать эти файлы. Если вы не используете ни один из этих макросов, то `configure` игнорирует заданные ключи '`-host`', '`-target`' и '`-build`'.

AC_CANONICAL_SYSTEM Macro

Определяет тип системы и устанавливает выходные переменные равными именам канонических типов систем. См. Раздел 8.3 [Переменные типов систем], с. 68, где описано, какие именно переменные устанавливаются этим макросом.

AC_CANONICAL_HOST Macro

Выполняет часть операций `AC_CANONICAL_SYSTEM`, относящуюся к определению типа системы, на которой будет запускаться пакет. Это все, что необходимо для программ, которые не входят в набор утилит компилятора.

AC_VALIDATE_CACHED_SYSTEM_TUPLE (*cmd*) Macro

Если в кэш-файле записан тип системы, не совпадающий с текущим, то выполняется команда *cmd* или печатается стандартное сообщение об ошибке.

8.3 Переменные типов систем

После вызова `AC_CANONICAL_SYSTEM` информация о типе системы содержится в нижеперечисленных выходных переменных. После `AC_CANONICAL_HOST` устанавливаются только те из перечисленных переменных, чьи имена начинаются на `host`.

`build, host, target`
: канонические имена систем;

`build_alias, host_alias, target_alias`
: имена, указанные пользователем или канонические имена, если был использован файл `config.guess`;

`build_cpu, build_vendor, build_os`
`host_cpu, host_vendor, host_os`
`target_cpu, target_vendor, target_os`
: отдельные части канонического имени (для удобства).

8.4 Использование типов систем

Как использовать канонический тип системы? Обычно вы используете его в одном или нескольких операторах `case` в '`configure.in`' для выбора специфических для системы файлов С. Затем делает ссылки на файлы, чьи имена содержат информацию о системе, чтобы они назывались также своим обобщенным именем, например, '`host.h`' или '`target.c`'. Шаблоны в операторе `case` могут использовать специальные символы командного процессора для группировки нескольких вариантов вместе, например как в таком фрагменте:

```
case "$target" in
i386-*-mach* | i386-*-gnu*) obj_format=aout emulation=mach bfd_gas=yes ;;
i960-*-bout) obj_format=bout ;;
esac
```

AC_LINK_FILES (*source...*, *dest...*) Macro

Заставляет `AC_OUTPUT` сделать ссылку с каждого файла из списка *source* на соответствующий файл с именем *dest*. Если это возможно, то создается символьная ссылка, иначе создается жесткая ссылка. Имена *dest* и *source* должны быть заданы относительно каталога верхнего уровня с исходными текстами или каталога, в котором происходит сборка. Этот макрос может быть вызван неоднократно.

Например, такой вызов:

```
AC_LINK_FILES(config/${machine}.h config/${obj_format}.h, host.h object.h)
```

создает в текущем каталоге файл '`host.h`', который является ссылкой на '*srcdir*`/config/${machine}.h`', и '`object.h`', который является ссылкой на '*srcdir*`/config/${obj_format}.h`'.

Вы также можете использовать тип системы, на которой будет запускаться программа, для поиска утилит кросс-компиляции. См. Раздел 4.1.2 [Общие программы], с. 26, для информации о макросе `AC_CHECK_TOOL`, который выполняет это.

9 Локальная конфигурация

Скрипты `configure` поддерживают несколько видов решений локальной конфигурации. Пользователь может указать, где находятся внешние пакеты программного обеспечения, включить или отключить дополнительные возможности, установить программы, изменяя их имена, и установить значения по умолчанию для ключей `configure`.

9.1 Работа с внешним программным обеспечением

Некоторые пакеты требуют, или могут при случае использовать другие пакеты программного обеспечения, уже установленные в системе. Пользователь может указать скрипту `configure` с помощью ключей командной строки, какие внешние пакеты надо использовать. Ключи имеют одну из следующих форм:

`-with-`*package*[`=`*arg*]
`-without-`*package*

Например, '`-with-gnu-ld`' означает, что надо работать с компоновщиком GNU linker вместо других компоновщиков. '`-with-x`' означает работу с X Window System.

Пользователь может задать аргумент, поставив после имени пакета символ '`=`' и нужный аргумент. Вы можете задать аргумент, равный '`no`' для пакетов, которые используются по умолчанию; он сообщает о том, что этот пакет *не* надо использовать. Аргумент, который не равен ни '`yes`', ни '`no`', может включать имя или номер версии другого пакета, для более точного указания, с каким пакетом эта программа предполагает работать. Если аргумент не задан, то его значение по умолчанию равно '`yes`'. '`-without-`*package*' эквивалентно вызову '`-with-`*package*`=no`'.

Скрипты `configure` не выдают ошибок о ключах '`-with-`*package*', которые они не поддерживают. Такое поведение позволяет конфигурировать дерево исходных текстов, содержащее множество пакетов, с помощью скрипта `configure` верхнего уровня, когда пакеты поддерживают разные ключи, без выдачи фальшивых сообщений об ошибках в ключах, которые поддерживают лишь некоторые пакеты. К сожалению, побочным эффектом этого является то, что ошибка в задании ключей не диагностируется. До сих пор не было предложено лучшего подхода к решению этой проблемы.

Для каждого из внешних пакетов, который может быть использован в файле '`configure.in`', должен быть вызван макрос `AC_ARG_WITH` для определения того, заставил ли пользователь `configure` использовать этот пакет. Будет ли пакет использоваться по умолчанию или нет, а также то, какие аргументы будут правильны, зависит от вас.

AC_ARG_WITH (*package, help-string* [*, action-if-given* [*, action-if-not-given*]]) Macro

Если пользователь задал `configure` ключ '`-with-`*package*' или ключ '`-without-`*package*', то выполняются команды командного процессора *action-if-given*. Если ни один из ключей не задан, то выполняются команды *action-if-not-given*. Имя *package* задает другой пакет, с которым должна работать эта программа. Это имя должно содержать только буквы, цифры и знаки минус.

Аргумент ключа командной строки из кода командного процессора *action-if-given* в переменной командного процессора `withval`, который в действительности является значением переменной командного процессора `with_`*package*, с символами '`-`', замененными на символ '`_`'. Можете использовать эту переменную, если хотите.

Аргумент *help-string* является описанием ключа, который выглядит примерно так:

```
-with-readline          support fancy command line editing
```

help-string может занимать больше одной строки, если необходима подробная информация. Просто убедитесь, что строка разделена на колонки в выводе '`configure -help`'. Избегайте использовать символы табуляции в строке помощи. Для того, чтобы сохранить начальные пробелы, нужно поместить строку между символами '`[`' и '`]`'.

AC_WITH (*package, action-if-given* [, *action-if-not-given*]) Macro

Это устаревшая версия макроса `AC_ARG_WITH`, которая не поддерживает использование строки помощи.

9.2 Выбор ключей пакетов

Если пакет имеет необязательные возможности, которые задаются во время компиляции, то пользователь может задать `configure` ключи командной строки для указания— нужно ли их компилировать. Ключи имеют одну из следующих форм:

`-enable-`*feature*[`=`*arg*]
`-disable-`*feature*

Эти ключи позволяют пользователю выбрать, какие необязательные возможности нужно собрать и установить. Ключи '`-enable-`*feature*' никогда не должны приводить к тому, что какое-то свойство изменит свое поведение, или же заменять одну возможность другой. Эти ключи должны только включать или не включать части программы в процесс компиляции.

Пользователь может задать аргумент, который следует за именем свойства и знаком '`=`'. Если задать аргумент '`no`', то свойство будет недоступным. Свойство с аргументом может выглядеть примерно следующим образом: '`-enable-debug=stabs`'. Если аргумента не задано, то значением по умолчанию является '`yes`'. '`-disable-`*feature*' является эквивалентом '`-enable-`*feature*`=no`'.

Скрипты `configure` не выражают недовольства по поводу ключей '`-enable-`*feature*', которые они не поддерживают. Такое поведение позволяет конфигурировать дерево исходных текстов, содержащее множество пакетов, с помощью скрипта `configure` верхнего уровня, когда пакеты поддерживают разные ключи, без выдачи фальшивых сообщений об ошибках о ключах, которые поддерживают только некоторые пакеты. Побочным эффектом этого является то, что ошибка в задании ключей не диагностируется. До сих пор не было предложено лучшего подхода к решению этой проблемы.

Для каждой из необязательных возможностей '`configure.in`' должен вызывать `AC_ARG_ENABLE` для определения, запросил ли пользователь `configure` включить эту воз-

можность. Будет ли эта возможность включена по умолчанию или нет, и какие аргументы будут правильными, зависит от вас.

AC_ARG_ENABLE (*feature, help-string* [, *action-if-given* [, *action-if-not-given*]]) Macro

Если пользователь задал `configure` ключ '`-enable-`*feature*' или '`-disable-`*feature*', то запускаются команды *action-if-given*. Если не был задан ни один ключ, то запускаются команды *action-if-not-given*. Имя *feature* указывает необязательную возможность, которую пользователь может включить или выключить. Имя должно состоять только из букв, цифр и знаков "минус".

Аргумент ключа доступен из кода командного процессора *action-if-given* в переменной командного процессора `enableval`, которая в действительности является значением переменной `enable_`*feature*, причем символы '`-`' заменены на символ '`_`'. Если хотите, то можете использовать эту переменную. Аргумент *help-string* делает то же самое, что и соответствующий аргумент макроса `AC_ARG_WITH` (см. Раздел 9.1 [Внешнее ПО], с. 70).

AC_ENABLE (*feature, action-if-given* [, *action-if-not-given*]) Macro

Это устаревшая версия `AC_ARG_ENABLE`, которая не поддерживает использование строки помощи.

9.3 Детали локальной конфигурации

Некоторые пакеты программ требуют сложной специфической для машины информации. Например, это имена машин, предоставляющих какие-либо сервисы, имена компаний, а также электронные почтовые адреса, по которым можно связаться с какими-то людьми. Поскольку некоторые скрипты, созданные Metaconfig, запрашивают эту информацию интерактивно, то люди часто спрашивают, как можно получить эту информацию в Autoconf-скриптах, которые не являются интерактивными.

Такая информация по конфигурации машины должна быть помещена в файл, редактируемый *только людьми*, а не программами. Файл может располагаться либо в зависимости от значения используемой переменной `prefix`, либо находиться в стандартном месте, например, в домашнем каталоге пользователя. Он даже может быть указан в переменной среды. Программа должна использовать этот файл во время выполнения, а не во время компиляции. Настройка во время выполнения является более удобной для пользователей и делает процесс настройки более простым, чем получение информации во время процесса конфигурации. См. раздел "Variables for Installation Directories" в *GNU Coding Standards*, где описано, где именно необходимо размещать файлы данных.

9.4 Преобразование имен программ при установке

Autoconf поддерживает изменение имен программ при их установке. Для того, чтобы использовать это преобразование, в файле '`configure.in`' должен быть вызов макроса `AC_ARG_PROGRAM`.

AC_ARG_PROGRAM Macro

Помещает в выходную переменную `program_transform_name` последовательность команд `sed`, используемых для изменения имен устанавливаемых программ.

Если при запуске `configure` задан любой из нижеописанных ключей, то имена программ изменяются соответствующим образом. В противном случае, если был вызван макрос `AC_CANONICAL_SYSTEM` и значение, заданное с помощью ключа '`-target`' отличается от типа машины, (указанного с помощью ключа '`-host`' или типа по умолчанию, определенного с помощью `config.sub`), то в качестве префикса имени используется тип целевой машины и дефис. Если не задано ни того, ни другого, то преобразование имен не выполняется.

9.4.1 Ключи преобразования

Вы можете задать преобразование имен, запустив `configure` со следующими ключами командной строки:

`-program-prefix=`*prefix*
: добавляет префикс *prefix* к именам;

`-program-suffix=`*suffix*
: добавляет к именам суффикс *suffix*;

`-program-transform-name=`*expression*
: выполняет подстановки `sed` *expression* для имен программ.

9.4.2 Примеры преобразований

Эти преобразования полезны при работе с программами, которые являются частью кросс-компиляционной среды разработки. Например, кросс-ассемблер, запускаемый на Sun 4 и настроенный с ключом '`-target=i960-vxworks`' обычно устанавливается как '`i960-vxworks-as`', а не как '`as`', иначе его можно перепутать с родным ассемблером Sun 4.

Можно сделать так, чтобы имена программ начинались с символа '`g`', если не хотите, чтобы программы GNU, установленные в системе, заслоняли собой другие утилиты с тем же именем. Например, если вы настраиваете программу GNU `diff` с ключом '`-program-prefix=g`', то затем вы можете запустить '`make install`' и программа будет установлена как '`/usr/local/bin/gdiff`'.

В качестве более изощренного примера вы можете использовать

```
-program-transform-name='s/^/g/; s/^gg/g/; s/^gless/less/'
```

для добавления символа '`g`' к большинству имен программ в дереве исходных текстов, за исключением программ типа `gdb`, чьи имена уже начинаются с этого символа, и за исключением `less` и `lesskey`, которые не являются программами GNU. (Предполагается, что дерево исходных текстов, содержащее эти программы, уже сконфигурировано для использования этой возможности).

Одним из способов одновременной установки нескольких версий некоторых программ является добавление номера версии программы к имени. Например, если вы хотите сохранить для дальнейшего использования Autoconf версии 1, то вы можете настроить Autoconf версии 2 с помощью ключа '`-program-suffix=2`' для

того, чтобы программы были установлены под именами ‘/usr/local/bin/autoconf2’, ‘/usr/local/bin/autoheader2’ и т. п.

9.4.3 Правила преобразования

Вот как нужно использовать переменную program_transform_name в ‘Makefile.in’:

```
transform=@program_transform_name@
install: all
        $(INSTALL_PROGRAM) myprog $(bindir)/‘echo myprog|sed ’$(transform)’‘

uninstall:
        rm -f $(bindir)/‘echo myprog|sed ’$(transform)’‘
```

Если у вас устанавливается больше одной программы, то вы можете выполнять ту же операцию в цикле:

```
PROGRAMS=cp ls rm
install:
        for p in $(PROGRAMS); do \
          $(INSTALL_PROGRAM) $$p $(bindir)/‘echo $$p|sed ’$(transform)’‘; \
        done

uninstall:
        for p in $(PROGRAMS); do \
          rm -f $(bindir)/‘echo $$p|sed ’$(transform)’‘; \
        done
```

Преобразовывать ли имена файлов документации (Texinfo или man) – сложный вопрос. Кажется, на него нет единственного ответа, потому что для преобразования имен есть несколько причин. Часто документация не является специфической для конкретной архитектуры, а файлы Texinfo не конфликтуют с системной документацией. Но эти файлы иногда могут конфликтовать с ранними версиями тех же файлов, а страницы man иногда могут конфликтовать с системной документацией. В качестве компромисса, можно выполнять преобразования имен страниц man, но не руководств в формате Texinfo.

9.5 Установка значений по умолчанию для машины

Созданные Autoconf скрипты configure позволяют вам задать значения по умолчанию для некоторых параметров настройки. Вы можете сделать это, создавая файлы инициализации для машины и для целой системы.

Если установлена переменная среды CONFIG_SITE, то configure использует ее значение как имя скрипта командного процессора, который необходимо выполнить. В противном случае он считывает скрипт ‘*prefix*/share/config.site’, если тот существует, а затем скрипт ‘*prefix*/etc/config.site’, также если он существует. Таким образом, специфические для машины файлы перекрывают настройки в машинно-независимых файлах в случае конфликта.

Файлы настроек машины могут быть произвольными скриптами командного процессора, но реально использоваться в них могут только определенные строки ко-

да. Поскольку `configure` считывает кэш-файлы после того, как он считывает файлы настройки машины, то файл локальной конфигурации может определить кэш-файл по умолчанию, который будет общим для всех запускаемых в системе скриптов `configure`, которые созданы с помощью Autoconf. Если вы установите кэш-файл по умолчанию в файле локальной настройки, то хорошо было бы установить также выходную переменную `CC`, поскольку кэш-файл является правильным только для определенного компилятора, а многие системы имеют несколько компиляторов.

В файле локальных настроек вы можете проверять или изменять значения ключей командной строки, заданных скрипту `configure`; ключи устанавливают переменные командного процессора, которые называются так же, как и ключи командной строки, но с символами дефиса, замененными на символы подчеркивания. Исключением из этого правила являются ключи '`-without-`' и '`-disable-`', которые подобны заданию соответствующих ключей '`-with-`' или '`-enable-`' со значением '`no`'. Таким образом, '`-cache-file=localcache`' устанавливает переменную `cache_file` в значение '`localcache`'; '`-enable-warnings=no`' или '`-disable-warnings`' устанавливают переменную `enable_warnings` равной значению '`no`'; '`-prefix=/usr`' устанавливает переменную `prefix` равной '`/usr`'; и т. п.

В файлах локальных настроек также можно устанавливать нестандартные значения по умолчанию для других выходных переменных, таких как `CFLAGS`: иначе вам пришлось бы делать это снова и снова в командной строке. Если вы обычно используете нестандартные значения для переменных *prefix* или *exec_prefix* (которые обычно используются для указания файла локальной конфигурации), то все равно можно задать эти значения в этом файле, если указать его имя в переменной среды `CONFIG_SITE`.

Вы можете сами установить значения некоторых кэш-переменных в файле локальной конфигурации. Это полезно делать при кросс-компиляции, поскольку невозможно определить проверить возможности, которые требуют запуска тестовых программ. Вы можете "заполнить кэш" установкой этих значений для этих систем в файле '*prefix*`/etc/config.site`'. Для определения имен кэш-переменных, которые вам необходимо установить, поищите переменные с именами, содержащими '`_cv_`' в соответствующих скриптах `configure` или в исходном коде `m4` макросов Autoconf.

Кэш-файл не переопределяет ни одну переменную, установленную в файлах локальной конфигурации. Сходным образом вы не должны переопределять ключи командной строки в файлах локальной конфигурации. Ваш код должен проверять, имеют ли уже переменные типа `prefix` или `cache_file` значения по умолчанию (установленные ранее в процессе выполнения `configure`), и если да, то не изменять этих значений.

Вот пример файла '`/usr/share/local/gnu/share/config.site`'. Команда '`configure -prefix=/usr/share/local/gnu`' должна прочитать этот файл (если переменная `CONFIG_SITE` не установлена в другое значение).

```
# config.site для configure
#
# изменение некоторых значений по умолчанию.
test "$prefix" = NONE && prefix=/usr/share/local/gnu
test "$exec_prefix" = NONE && exec_prefix=/usr/local/gnu
test "$sharedstatedir" = '${prefix}/com' && sharedstatedir=/var
test "$localstatedir" = '${prefix}/var' && localstatedir=/var
#
```

```
# разрешить скриптам, созданным Autoconf 2.x, пользоваться общим кэш-файлом
# для получения результатов тестов, которые действительны для данной
# архитектуры.
if test "$cache_file" = ./config.cache; then
  cache_file="$prefix/var/config.cache"
  # Кэш-файл действителен только для одного компилятора C.
  CC=gcc
fi
```

10 Запуск скриптов configure

Ниже находятся рекомендации по настройке пакета, использующего скрипт configure. Эти рекомендации можно включить в файл 'INSTALL' вашего пакета. Текстовая версия файла 'INSTALL', которую вы можете использовать, поставляется с Autoconf.

10.1 Простая установка

Вот основные инструкции по установке.

Скрипт configure пытается определить правильные значения для различных, зависящих от системы переменных, которые используются в процессе установки. Он использует эти переменные для создания файлов 'Makefile' в каждом из каталогов пакета. он также может создавать один или несколько файлов '.h' содержащих зависящие от системы определения. В заключение, он создает скрипт командного процессора с именем 'config.status', который вы можете в дальнейшем запускать для воссоздания текущей настройки, также создается файл 'config.cache', который сохраняет результаты тестов, для ускорения перенастройки, и файл 'config.log', содержащий вывод компилятора (этот файл в основном полезен для отладки configure).

Если для компиляции пакета вам необходимо выполнить нетривиальные вещи, то то пожалуйста попытайтесь определить как configure мог бы проверить как выполнить их, и затем пошлите diff-файл или инструкции на адрес, данный в файле 'README', так что они могут быть рассмотрены для включения в следующий выпуск. Если в некоторых случаях 'config.cache' содержит результаты, которые вы не хотите хранить, то вы можете исправить или удалить его.

Файл 'configure.in' используется для создания скрипта 'configure' программой autoconf. Вам необходимо иметь 'configure.in' только, если вы хотите изменить его или заново создать скрипт 'configure' с помощью более новой версии autoconf.

Наиболее простым способом компиляции данного пакета являются следующие действия:

1. перейдите в каталог, содержащий исходный код пакета и наберите './configure' в командной строке, для того, чтобы настроить пакет для вашей системы. Если вы используете csh на старой версии System V, то вам может понадобиться набрать 'sh ./configure' вместо предыдущего примера, для того, чтобы не допустить выполнения данного скрипта с помощью csh.

   Работа configure займет некоторое время. В течении выполнения скрипт выдает некоторые сообщения, о том какие свойства он проверяет.
2. Наберите 'make' для компиляции пакета.
3. Вы можете набрать 'make check' для запуска любых собственных тестов, которые поставляются вместе с пакетом.
4. Наберите 'make install' для установки программ и файлов данных и документации.
5. вы можете удалить исполнимые файлы программ и объектные файлы из каталога с исходными текстами пакета набрав 'make clean'. Для удаления файлов

созданных `configure` (так что вы можете скомпилировать пакет с помощью разных компиляторов), наберите '`make distclean`'. Также существует цель '`make maintainer-clean`', но она в основном предназначена для разработчиков программного обеспечения. Если вы используете ее, то вы должны получить все другие программы, для того, чтобы обновлять файлы, которые поставляются с дистрибутивом.

10.2 Компиляторы и ключи

Некоторые системы требуют необычных ключей для компиляции или компоновки, о которых скрипт `configure` просто не знает. Вы можете задать начальные значения для переменных `configure` установив их в среде. Используя командный процессор совместимый с процессором Bourne вы можете задать эти переменные с помощью командной строки, подобной этой:

```
CC=c89 CFLAGS=-O2 LIBS=-lposix ./configure
```

или в системах, в которых имеется программа `env`, вы можете выполнить следующий код:

```
env CPPFLAGS=-I/usr/local/include LDFLAGS=-s ./configure
```

10.3 Компиляция для нескольких архитектур

Вы можете одновременно скомпилировать пакет для более одного типа компилятора, поместив объектные файлы для каждой из архитектур в отдельный каталог. Для того чтобы сделать это, вы должны использовать такую версию программы `make`, которая поддерживают переменную `VPATH`, например, такую как GNU `make`. Перейдите в каталог, в который вы хотите поместить объектные и исполняемые файлы и запустите оттуда скрипт `configure`. `configure` автоматически проверит исходные тексты в каталог, в котором находится `configure` в также в каталоге '`..`'.

Если вы используете программу `make`, которая не поддерживает переменную `VPATH`, то вы должны одновременно компилировать программу только для одной архитектуры. После того, как вы установили пакет для конкретной архитектуры, используйте правило '`make distclean`' до выполнения настройки для другой архитектуры.

10.4 Имена для установки

По умолчанию '`make install`' установит файлы пакета в '`/usr/local/bin`', '`/usr/local/man`', и т.д.. Вы можете задать префикс установки, который отличается от '`/usr/local`'. Это выполняется передачей `configure` ключа командной строки '`-prefix=`*path*'.

Вы можете указать разные префиксы установки для специфических архитектуры файлов, и для файлов не зависящих от архитектуры. Если вы зададите `configure` ключ '`-exec-prefix=`*path*', то пакет будет использовать *path* как префикс для установки программ и библиотек. Документация и другие файлы данных будут использовать обычный префикс.

В добавок, если вы используете необычное расположение каталогов, то вы можете задать ключи, подобные '`-bindir=`*path*', для того, чтобы указать различные значения

для отдельных типов файлов. Запустите '`configure -help`' для получения списка каталогов, которые вы можете задать в командной строке, и списка типов файлов устанавливаемых в каждый из каталогов.

Если пакет поддерживает это, то вы можете установить программу с дополнительными суффиксами или префиксами в имени программы.Это выполняется заданием `configure` ключа '`-program-prefix=`*PREFIX*' или '`-program-suffix=`*SUFFIX*'.

10.5 Дополнительные возможности

Некоторые пакеты обращают внимание на ключи '`-enable-`*feature*' переданные `configure`, где *feature* показывает дополнительную часть пакета. Они также могут обращать внимание на ключи '`-with-`*package*', где *package* является чем-то подобным '`gnu-as`' или '`x`' (для X Window System). В файле '`README`' должны быть описаны распознаваемые пакетом ключи '`-enable-`' и '`-with-`'.

Для пакетов, которые использую X Window System, `configure` обычно может автоматически найти заголовочные файлы и библиотеки X, однако если скрипт не смог определить их расположение, то вы можете запустить `configure` с ключами '`-x-includes=`*dir*' и '`-x-libraries=`*dir*' и указав правильные значения.

10.6 Указание типа системы

Может быть много возможностей, которые `configure` не сможет определить автоматически, но которые нужны для определения типа системы на которой будет запускать пакет. Обычно `configure` может выполнить определение типа системы, но если в случае неудачи скрипт выдаст сообщение, говорящее о том, что он не смог определить тип системы, то задайте тип с помощью ключа '`-host=`*type*'. *type* может являть либо коротким именем, определяющим тип системы, таким как '`sun4`', либо каноническим именем, содержащим 3 поля:

cpu-company-system

загляните в файл '`config.sub`' для того, чтобы узнать возможные значения для каждого из полей. Если файл '`config.sub`' не включен в состав пакета, то данному пакету не нужно знать тип системы.

Если вы собираете утилиты компилятора для кросс-компиляции, то вы также можете использовать ключ '`-target=`*type*' для выбора типа системы, для которой эти утилиты будут создавать код, а также ключ '`-build=`*type*' для выбора типа системы на которой вы компилируете пакет.

10.7 Совместное использование значений по умолчанию

Если вы хотите чтобы значения по умолчанию для скриптов `configure` использовались совместно, то вы можете создать локальный скрипт с именем '`config.site`', который задаст значения по умолчанию для таких переменных как `CC`, `cache_file` и `prefix`. `configure` ищет '*prefix*`/share/config.site`', если он существует, а затем '*prefix*`/etc/config.site`' если он существует. Или вы можете установить переменную среды `CONFIG_SITE` равную пути к этому скрипту. Предупреждение: не все скрипты `configure` производят поиск этого скрипта.

10.8 Контроль выполнения

`configure` распознает следующие ключи командной строки, которые контролируют как он выполняется.

`-cache-file=`*file*
: Использовать и сохранять результаты тестов в файле *file* вместо файла '`./config.cache`'. Для запрещения кэширования установите *file* равным '`/dev/null`', при отладке `configure`.

`-help`
: Выдает список ключей командной строки `configure` и прекращает работу.

`-quiet`
`-silent`
`-q`
: Не выдает сообщений о том, какие проверки выполняются. Для запрещения всего вывода, перенаправьте вывод в файл '`/dev/null`' (сообщения об ошибках все равно будут отображаться).

`-srcdir=`*dir*
: Ищет исходный текст пакета в каталоге *dir*. Обычно `configure` может автоматически определить этот каталог.

`-version`
: Выдает номер версии Autoconf использовавшейся для создания скрипта `configure` и прекращает работу.

`configure` также принимает некоторые другие, не так сильно полезные ключи.

11 Воссоздание конфигурации

Скрипт `configure` создает файл с именем ‘`config.status`’, который описывает, какие параметры конфигурации были указаны при последней конфигурации пакета. Это файл является скриптом командного процессора, который при запуске воссоздает ту же самую настройку.

Вы можете задать скрипту ‘`config.status`’ ключ ‘`-recheck`’, чтобы он обновил сам себя. Этот ключ полезен, если вы изменяете `configure`, так что результаты некоторых тестов могут измениться по сравнению с предыдущим запуском. Ключ ‘`-recheck`’ перезапускает `configure` с аргументами, использованными при предыдущих запусках, с добавлением ключа ‘`-no-create`’, который не дает `configure` запустить ‘`config.status`’ и создать ‘`Makefile`’ и другие файлы, а также с добавлением ключа ‘`-no-recursion`’, который предотвращает запуск скриптов `configure` в подкаталогах. (Это сделано для того, чтобы другие правила ‘`Makefile`’ могли бы запускать ‘`config.status`’ при его изменении; Например, см. Раздел 3.3.3 [Автоматическая пересборка], с. 16).

‘`config.status`’ также распознает ключ ‘`-help`’, который выдает список ключей ‘`config.status`’, и ключ ‘`-version`’, который выдает номер версии Autoconf, которая была использована при создании скрипта `configure`, создавшего файл ‘`config.status`’.

‘`config.status`’ проверяет несколько переменных среды, которые могут изменить его поведение:

CONFIG_SHELL Variable

Командный процессор, который запустит `configure` с ключом ‘`-recheck`’. Он должен быть совместимым с командным процессором Bourne. Значение по умолчанию – является ‘`/bin/sh`’.

CONFIG_STATUS Variable

Имя файла, которое будет использоваться для создания скрипта командного процессора, который сохранит текущую настройку. Значением по умолчанию является ‘`./config.status`’. Эта переменная полезна в том случае, когда один пакет использует части другого, а скрипты `configure` не должны быть слиты вместе, поскольку они сопровождаются по отдельности.

Следующие переменные обеспечивают возможность отдельным пакетам совместно использовать значения переменных, вычисленных скриптом `configure`. Это может быть полезно, если одному пакету нужно больше возможностей, чем другому. Эти переменные позволяют файлу ‘`config.status`’ создавать и другие файлы, не только те, что указаны в файле ‘`configure.in`’, чтобы их можно было бы использовать в другом пакете.

CONFIG_FILES Variable

Файлы, в которых будут выполняться подстановки ‘`@`*variable*`@`’. Обычно эти файлы задаются как аргументы макроса `AC_OUTPUT` в ‘`configure.in`’.

CONFIG_HEADERS Variable

Файлы, в которых будет выполняться подстановка операторов `#define` языка C. Обычно это файлы, заданные в аргументах макроса `AC_CONFIG_HEADER`; если этот макрос не был вызван, то ‘`config.status`’ игнорирует эту переменную.

Эти переменные также позволяют написать правила ‘`Makefile`’, которые будут пересоздавать только некоторые файлы. Например, в вышеприведенной зависимости (см. Раздел 3.3.3 [Автоматическая пересборка], с. 16), ‘`config.status`’ запускается дважды при изменении ‘`configure.in`’. Если это беспокоит вас, то вы можете сделать так, чтобы при каждом запуске обновлялись файлы только для этого правила:

```
config.h: stamp-h
stamp-h: config.h.in config.status
        CONFIG_FILES= CONFIG_HEADERS=config.h ./config.status
        echo > stamp-h

Makefile: Makefile.in config.status
        CONFIG_FILES=Makefile CONFIG_HEADERS= ./config.status
```

(Если ‘`configure.in`’ не вызывает макрос `AC_CONFIG_HEADER`, то нет необходимости устанавливать `CONFIG_HEADERS` в правилах `make`).

12 Вопросы об Autoconf

Раз за разом задаются одни и те же вопросы об Autoconf. Вот некоторые из них.

12.1 Распространение скриптов `configure`

> Какие ограничения существуют на распространение скриптов `configure`, созданных Autoconf? Как могут затронуть они мою программу, которая использует эти скрипты?

На использование или распространение скриптов настройки, созданных Autoconf, не накладывается никаких ограничений. В Autoconf версии 1, они подпадали под действие GNU General Public License. Мы все еще поощряем авторов к распространению их работ под действием лицензий, сходных с GPL, но это не обязательно при использовании Autoconf.

По поводу других файлов, которые могут быть использованы с `configure`: '`config.h.in`' находятся под теми авторскими правами, которые используются для вашего '`configure.in`', поскольку эти файлы являются производными этого файла и свободно доступного файла '`acconfig.h`'. '`config.sub`' и '`config.guess`' не подпадают под GPL в том случае, когда они используются со скриптами `configure`, созданными с помощью Autoconf: в этом случае вы можете распространять их под той же лицензией, что и остальной пакет. '`install-sh`' получен от X Consortium и не защищен авторскими правами.

12.2 Почему требуется GNU `m4`?

> Почему Autoconf требует наличия GNU `m4`?

Многие из реализаций `m4` имеют зашитые в программу ограничения на размер и количество макросов, которые Autoconf не может преодолеть. В них также отсутствуют некоторые встроенные макросы, без которых было бы трудно создать такое сложное приложение, как Autoconf, включая:

```
builtin
indir
patsubst
__file__
__line__
```

Поскольку Autoconf нужен только людям, сопровождающим программное обеспечение, и поскольку GNU `m4` прост в настройке и установке, то кажется вполне естественным, что также требуется установить GNU `m4`. Многие люди, сопровождающие программное обеспечение GNU и другое свободное ПО, уже установили у себя большинство утилит GNU, потому что предпочитают именно их.

12.3 Как я могу начать работу?

> Если Autoconf требует GNU `m4`, а GNU `m4` имеет скрипт Autoconf `configure`, то как я могу начать работу? Это похоже на проблему яйца и курицы!

Вы неправильно поняли. Хотя GNU `m4`и поставляется со скриптом `configure`, созданным Autoconf, сам Autoconf не требуется для запуска скрипта и установки GNU `m4`. Autoconf требуется только если вы хотите изменить скрипт `configure` для `m4`, а это требуется немногим (в основном тем, кто сопровождает данный пакет).

12.4 Почему не используется Imake?

Почему не используется Imake вместо скриптов `configure`?

Разные люди писали ответы на этот вопрос, так что я приведу здесь переделанный вариант их объяснений.

Следующий ответ основан на материале, написанном Richard Pixley:

Скрипты, созданные Autoconf, часто работают на машинах, для которых они не были предназначены. То есть они хорошо работают, когда нужно создать конфигурацию для совершенно новой системы. Imake такого не умеет.

Imake использует общую базу данных, специфических для конкретных систем. Для X11 этого достаточно, потому что этот пакет делается одной организацией, которая имеет централизованный контроль над этой базой данных.

Утилиты GNU выпускаются не так. Каждую утилиту GNU сопровождает отдельный человек, и эти люди разбросаны по всему миру. Использование общей базы данных будет просто кошмаром при сопровождении пакета. Autoconf может показаться неким подобием базы данных, но на самом деле это не так. Вместо перечисления зависимостей от машины, он перечисляет требования программ.

Если вы рассматриваете набор GNU как набор отдельных утилит, то возникают сходные проблемы. Но утилиты разработки GNU могут быть настроены как кросс-утилиты в почти любом сочетании машина+цель. Все эти конфигурации могут быть установлены одновременно. Она даже могут делить между собой не зависящие от машины файлы. Imake не позволяет сделать этого.

Шаблоны Imake являются формой стандартизации. Стандарты кодирования GNU позволяют сделать это без ненужного наложения тех же самых ограничений.

Вот дополнительные объяснения, написанные Per Bothner:

Одним из преимуществ Imake является то, что он легко создает большие файлы Makefile, используя директивы препроцессора `cpp` '`#include`' и механизм макросов. Однако на `cpp` невозможно программировать: у него ограниченные возможности условных операторов и нет операторов циклов. Помимо того, `cpp` не имеет доступа к переменным среды.

Все эти проблемы решаются использованием командного процессора `sh` вместо `cpp`. Командный процессор предоставляет полные возможности программирования, имеет подстановку макросов, может выполнять или создавать другие скрипты командного процессора и может пользоваться переменными среды.

Paul Eggert развил эту тему дальше:

При использовании Autoconf установщикам пакета не нужно предполагать, что Imake уже установлен и работает правильно. Это не кажется таким уж большим достоинством для людей привыкших к Imake, но на многих машинах Imake не установлен или установка по умолчанию работает не совсем правильно и, таким образом,

требование наличия Imake для установки пакета будет препятствовать его распространению на таких машинах. Например, файлы шаблонов и настроек Imake могут быть установлены неправильно на машине, или процедура построения, используемая Imake, может неправильно предполагать, что все исходные тексты находятся в одном большом дереве каталогов, или настройка Imake может предполагать, что имеется один компилятор, в то время как вам необходимо использовать другой компилятор, или могут быть несоответствия между версиями Imake, используемой пакетом и версией Imake, поддерживаемой данной машиной. Эти проблемы встречаются намного реже при использовании Autoconf, поскольку каждый пакет поставляется со своим, независимым препроцессором конфигурации.

Также Imake часто страдает от неожиданного взаимодействия между `make` и препроцессором C. Фундаментальной проблемой является то, что препроцессор C был создан для обработки программ на языке C, а не файлов '`Makefile`'. Это является менее значимой проблемой при использовании Autoconf, который использует препроцессор общего назначения `m4`, а автор пакета (а не установщик пакета) выполняет обработку стандартным способом.

В заключение, Mark Eichin замечает:

Imake, в конце концов, не такой уж и расширяемый. Для того, чтобы добавить к Imake новую возможность, вам необходимо создать собственный шаблон для проекта и сдублировать большинство возможностей существующего шаблона. Это означает, что для сложного проекта использование поставляемых производителем шаблонов Imake приведет к сбою — поскольку они не предоставляют возможностей, в которых нуждается ваш проект (если только он не является программой для X11).

Однако с другой стороны:

Одно из преимуществ Imake по сравнению с `configure`: файлы '`Imakefile`' оказываются значительно меньше (более того, они менее многословны), чем файлы '`Makefile.in`'. Однако существуют решения этой проблемы— по крайней мере для исходных текстов Kerberos V5, мы изменили файлы для вызова общих частей: фрагментов '`Makefile`' для всего дерева каталогов, которые находятся в файлах '`post.in`' и '`pre.in`'. Это означает, что часть общей функциональности не дублируется, даже хотя они будут находиться в скриптах `configure`.

13 Обновление с версии 1

По большей части Autoconf версии 2 обратно совместим с версией 1. Однако же, в нем появились более удобные способы решения некоторых вещей, а некоторые особенно уродливые способы из версии 1 не поддерживаются. Так что, в зависимости от сложности ваших файлов ‘`configure.in`’, вам, может быть, придется вручную скорректировать ваши файлы, чтобы использовать их с Autoconf версии 2. Этот раздел описывает некоторые проблемы, которых можно ожидать при переходе к новой версии. Также, возможно, ваши скрипты `configure` получат выгоду от использования новых возможностей версии 2; список изменений приведен в файле ‘`NEWS`’ дистрибутива Autoconf.

В первую очередь убедитесь, что у вас установлен GNU `m4` версии 1.1 или более свежей, предпочтительней использовать версию 1.3 или следующие. Версии до 1.1 имели ошибку, которая препятствовала их работе с Autoconf версии 2. Версии 1.3 и более поздние работают быстрее, чем более ранние версии, поскольку с версии 1.3 GNU `m4` имеет более эффективную реализацию diversions и может сохранить свое состояние в файле, который потом может быстро считан обратно.

13.1 Измененные имена файлов

Если у вас есть файл ‘`aclocal.m4`’, установленный вместе с Autoconf (а не в каталоге с исходными текстами), то вы должны переименовать его в ‘`acsite.m4`’. См. Раздел 2.4 [Запуск autoconf], с. 8.

Если вы распространяете с вашим пакетом файл ‘`install.sh`’, то переименуйте его в ‘`install-sh`’, так что встроенные правила `make` не будут создавать на его основе файл ‘`install`’. `AC_PROG_INSTALL` ищет файл, пользуясь обоими именами, но лучше использовать новое имя.

Если вы используете ‘`config.h.top`’ или ‘`config.h.bot`’, то вы все равно сможете их использовать, но будет лучше, если вы объедините их в файл ‘`acconfig.h`’. См. Раздел 3.4.2 [Запуск autoheader], с. 18.

13.2 Измененные файлы Makefile

Добавьте ‘`@CFLAGS@`’, ‘`@CPPFLAGS@`’ и ‘`@LDFLAGS@`’ в ваши файлы ‘`Makefile.in`’, чтобы туда попадали значения соответствующих переменных среды, установленные при запуске `configure`. Это необязательно, но удобно для пользователей.

Также добавьте ‘`@configure_input@`’ в комментарий каждого из входных файлов макроса `AC_OUTPUT`, кроме файлов ‘`Makefile`’, чтобы выходные файлы содержали сообщение о том, что они созданы скриптом `configure`. Автоматический выбор синтаксиса для комментариев в файлах, для которых вызывается `AC_OUTPUT`, было бы слишком сложно.

Добавьте ‘`config.log`’ и ‘`config.cache`’ в список файлов, которые вы удаляете с помощью цели `distclean`.

Если у вас в ‘`Makefile.in`’ имеется следующее:

```
prefix = /usr/local
exec_prefix = ${prefix}
```

то вы должны изменить эту запись на следующую:

```
prefix = @prefix@
exec_prefix = @exec_prefix@
```

Старое поведение замены переменных без знаков '@' вокруг них было удалено.

13.3 Измененные макросы

В Autoconf версии 2 многие макросы были переименованы. Вы все равно можете использовать старые имена, но новые имена макросов более понятны и для них легче найти документацию. См. Глава 15 [Старые имена макросов], с. 94, где приведена таблица соответствия новых имен старым именам. Используйте программу `autoupdate` для преобразования ваших файлов '`configure.in`' для использования новых имен макросов. См. Раздел 13.4 [Запуск autoupdate], с. 87.

Некоторые макросы были заменены аналогичными, которые лучше выполняют нужную задачу, но несовместимы по параметрам вызова. Если вы получаете предупреждения о вызове устаревших макросов во время запуска `autoconf`, то можете спокойно игнорировать эти предупреждения, но ваш скрипт `configure`, вообще, будет работать лучше, если вы последуете советам заменить устаревший макрос на новый. Аналогичным образом был изменен механизм выдачи результатов тестов. Если вы использовали команды `echo` или `AC_VERBOSE` (может быть, посредством `AC_COMPILE_CHECK`), то вывод вашего скрипта `configure` будет выглядеть лучше, если вы станете использовать макросы `AC_MSG_CHECKING` и `AC_MSG_RESULT`. См. Раздел 6.4 [Выдача сообщений], с. 60. Эти макросы лучше всего работают при использовании кэшированных переменных. См. Раздел 6.3 [Кэширование результатов], с. 57.

13.4 Использование `autoupdate` для обновления `configure`

Программа `autoupdate` обновляет файл '`configure.in`' заменяя вызовы старых макросов Autoconf на вызовы макросов с новыми именами. В Autoconf версии 2 большинство макросов были переименованы для использования более общей и понятной схемы наименования. Для описания новой схемы именования См. Раздел 7.2 [Имена макросов], с. 62. Хотя макросы со старыми именами все равно работают (см. Глава 15 [Старые имена макросов], с. 94, где приведен список старых имен макросов и соответствующих им новых имен), но если вы обновите свои файлы для соответствия новым именам макросов, то файлы '`configure.in`' станут читабельнее, а использовать свежую документацию по Autoconf станет проще.

Если `autoupdate` запущена без аргументов, то она обновляет '`configure.in`', делая резервную копию оригинальной версии файла с использованием суффикса '`~`' (или значения переменной среды `SIMPLE_BACKUP_SUFFIX`, если она установлена). Если вы зададите аргумент программе `autoupdate`, то она будет считывать данные из этого файла вместо '`configure.in`' и выводить данные в поток стандартного вывода.

`autoupdate` распознает следующие ключи командной строки:

`-help`
`-h` Выдает список ключей командной строки и прекращает работу.

`-macrodir=`*dir*

`-m` *dir* Ищет файлы с макросами Autoconf в каталоге *dir*, а не в каталоге, в который производилась установка. Вы также задать путь к этому каталогу в переменной среды `AC_MACRODIR`; использование этого ключа перекрывает переменную среды.

`-version` Выдает номер версии `autoupdate` и прекращает работу.

13.5 Измененные результаты

Если вы проверяли результаты предыдущих тестов путем проверки переменной командного процессора `DEFS`, то теперь вам необходимо переключиться на проверку значений переменных кэша для данных тестов. Переменная `DEFS` больше не существует во время запуска `configure`; она создается только при генерации выходных файлов. Это изменение поведения было сделано потому, что правильное экранирование содержимого этой переменной оказалось слишком громоздким и неэффективным при каждом вызове макроса `AC_DEFINE`. См. Раздел 6.3.1 [Имена переменных кэша], с. 58.

Например, вот фрагмент ‘`configure.in`’, написанного для Autoconf версии 1:

```
AC_HAVE_FUNCS(syslog)
case "$DEFS" in
*-DHAVE_SYSLOG*) ;;
*) # syslog не находится в библиотеках по умолчанию. Смотрим, есть ли он в
   # других библиотеках.
  saved_LIBS="$LIBS"
  for lib in bsd socket inet; do
    AC_CHECKING(for syslog in -l$lib)
    LIBS="$saved_LIBS -l$lib"
    AC_HAVE_FUNCS(syslog)
    case "$DEFS" in
    *-DHAVE_SYSLOG*) break ;;
    *) ;;
    esac
    LIBS="$saved_LIBS"
  done ;;
esac
```

Вот как это записывается для версии 2:

```
AC_CHECK_FUNCS(syslog)
if test $ac_cv_func_syslog = no; then
  # syslog не находится в библиотеках по умолчанию. Смотрим, есть ли он в
  # других библиотеках.
  for lib in bsd socket inet; do
    AC_CHECK_LIB($lib, syslog, [AC_DEFINE(HAVE_SYSLOG)
      LIBS="$LIBS $lib"; break])
  done
fi
```

Если вы обходили ошибку в макросе `AC_DEFINE_UNQUOTED`, добавляя символы обратной косой черты перед кавычками, то теперь вам придется удалить их. Этот макрос

сейчас работает предсказуемо и не рассматривает особым образом кавычки (кроме обратных). См. Раздел 6.2 [Установка выходных переменных], с. 56.

Все логические переменные командного процессора, устанавливаемые макросами Autoconf, используют ‘`yes`’ для истинных переменных. Большинство из них использует ‘`no`’ для ложных значений, хотя для обратной совместимости некоторые из них используют пустую строку. Если вы полагались, что переменная командного процессора будет установлена в что-нибудь типа ‘`1`’ или ‘`t`’ для истинного значения, то вам необходимо изменить ваши тесты.

13.6 Измененное написание макросов

При определении ваших собственных макросов вы должны использовать макрос `AC_DEFUN` вместо `define`. `AC_DEFUN` автоматически вызывает макрос `AC_PROVIDE` и проверяет, что макросы, вызываемые через `AC_REQUIRE`, не прерывают другие макросы, для предотвращения вложенных сообщений ‘`checking...`’ на экране. На самом деле, старый способ не причинит вреда, но он менее удобен и менее привлекателен. См. Раздел 7.1 [Определение макросов], с. 62.

Вы, вероятно, рассматривали макросы, поставляемые с Autoconf, как руководство по написанию макросов. Посмотрите на новую их версию, потому что стиль некоторых макросов сильно улучшен, а новые возможности активно используются.

Если вы делали хитрые вещи, используя недокументированные свойства Autoconf (макросы, переменные, diversions), то проверьте, не нужно ли изменить что-нибудь, чтобы учесть сделанные в Autoconf изменения. Может быть, теперь вы можете пользоваться стандартной возможностью версии 2, вместо того, чтобы упражняться в изобретательности. Или нет.

Для ускорения работы написанных вами макросов добавьте в них поддержку кэширования. Просмотрите все ваши тесты, может быть их нужно оформить в виде макросов, которые вы сможете использовать в разных пакетах.

14 История Autoconf

Вы спросите, зачем вообще Autoconf был написан? Как он оказался там, где он сейчас есть? Почему он выглядит подобно плевку гориллы? Если вы не интересуетесь такими вопросами, то в этой главе ничего полезного для вас нет, и вы можете спокойно пропустить ее. Но если вам *действительно* интересно, то я пролью немного света....

14.1 Бытие

В июне 1991 года я сопровождал много утилит GNU для Free Software Foundation. По мере того, как они переносились на все большее количество платформ, количество ключей '`-D`', которое пользователю надо было выбрать в '`Makefile`' (около 20), становилось обременительным. Особенно для меня— я тестировал каждую новую версию на различных платформах. Так что я написал для пакета fileutils небольшой скрипт на языке командного процессора для определения некоторых правильных настроек, и я выпустил его как часть пакета fileutils 2.0. Этот скрипт `configure` работал достаточно хорошо, так что в следующем месяце я вручную адаптировал его для создания подобных скриптов `configure` для нескольких других пакетов утилит GNU. Brian Berliner также адаптировал один из моих скриптов к своей системе контроля версий CVS.

Позже, тем же летом, я узнал, что Richard Stallman и Richard Pixley разработали аналогичные скрипты для использования в наборе утилит компиляции GNU; так что я адаптировал мои скрипты `configure` для поддержки развивающегося интерфейса их скриптов: использование файлов '`Makefile.in`' как шаблонов; добавление '`+srcdir`', первого ключа (из многих); и создание файлов '`config.status`'.

14.2 Исход

По мере получения ответов от пользователей я добавил много улучшений, используя Emacs для поиска и замены, вырезания и вставки, одних и тех же изменений в каждом из скриптов. По мере того, как все большее количество утилит GNU были адаптированы для использования скриптов `configure`, ручное обновление становилось все более неудобно. Rich Murphey, сопровождавший графические утилиты GNU, послал мне письмо, в котором писал, что скрипты `configure` работают очень хорошо, и спрашивал, нету ли у меня утилиты для их генерации, и могу ли я послать ее ему. Нет, я думал, но я должен был! Так, что я начал работать над тем, как создавать эти файлы. Так началось путешествие от рабства написанных вручную скриптов `configure` к изобилию и легкости Autoconf.

Пакет Cygnus `configure`, который был разработан примерно в то же время, управлялся таблицей; он предназначался в основном для работы с небольшим количеством типов систем и небольшим количеством возможностей, которые по большей части нельзя было автоматически определить (например, детали формата объектных файлов). Автоматическая система настройки, которую Brian Fox разработал для Bash, использовала аналогичный подход. Для общего пользования, мне кажется безнадежной попытка сопровождать постоянно обновляемую базу данных возможностей каждого из вариантов каждой операционной системы. Легче и надежнее будет проверять большинство свойств на лету— особенно на гибридных системах, которые люди изменяли локально, или на которых были установлены заплатки от производителя.

Я рассматривал архитектуру, сходную с используемой в Cygnus `configure`, где имеется один скрипт `configure`, который при запуске считывает части '`configure.in`'. Но я не хотел распространять с каждым пакетом тесты для всех возможностей, так что я пришел к решению иметь разные скрипты `configure`, созданные из '`configure.in`' с помощью препроцессора. Этот подход также представлял больший контроль и большую гибкость.

Я также ознакомился с использованием пакета Metaconfig, созданного Larry Wall, Harlan Stenn и Raphael Manfredi, но я решил не использовать его по нескольким причинам. Создаваемые с его помощью скрипты `Configure` являются интерактивными, что я нашел достаточно неудобным; мне не понравился способ, каким он проверял некоторые возможности (такие как наличие библиотечных функций); я не знал, сопровождался ли он тогда все еще, а скрипты `Configure`, которые я рассматривал, не работали на многих современных системах (таких как System V R4 и NeXT); у него не было достаточной гибкости в реакции на наличие или отсутствие какой-либо возможности; я нашел его трудным в освоении; он был слишком большим и сложным для моих нужд (я не осознавал тогда, как сильно придется развить Autoconf).

Я рассматривал использование языка Perl для создания моих скриптов `configure`, но решил, что `m4` лучше для выполнения простых текстовых подстановок: это получается проще, поскольку операция вывода подразумевается по умолчанию. Плюс к тому, каждый пользователь уже имеет его в своей системе. (В начале я не полагался на расширения GNU для `m4`). Несколько моих друзей в университете штата Maryland создавали надстройки на `m4` для разных программ, включая `tvtwm`, и я был заинтересован в изучении нового языка.

14.3 Левит

Поскольку мои скрипты `configure` определяли возможности системы автоматически, без интерактивного взаимодействия с пользователем, я решил назвать программу, которая создавала эти скрипты именем Autoconfig. Но с добавлением номера версии, это имя было слишком длинным для старых систем UNIX, так что я укоротил имя до Autoconf.

Осенью 1991 я собрал группу товарищей, чтобы начать поиски Чаши Грааля Переносимости (эээ, ну, то есть, чтобы они тестировали альфа-версии). Они предоставляли мне обратную связь, а я занимался инкапсуляцией кусочков моих вручную написанных скриптов в макросы `m4`, добавлял возможности и улучшал технологию проверок. Среди тестеров особенно выделялись: François Pinard, кто выдвинул идею сделать '`autoconf`' скриптом командного процессора, который запускал бы `m4` и проверял бы, чтобы все макросы были обработаны; Richard Pixley, кто предложил для получения более точных результатов запускать компилятор вместо поиска заголовочных файлов и символов по файловой системе; Karl Berry, который использовал Autoconf для настройки TEX и добавил индекс макросов в документацию; а также Ian Taylor, который добавил поддержку создания заголовочного файла C как альтернативу помещения ключей '`-D`' в '`Makefile`', так что он смог использовать Autoconf для пакета UUCP. Люди, тестировавшие альфа-версии, бодро изменяли свои файлы снова и снова, поскольку имена и соглашения о вызовах изменялись от версии к версии Autoconf. Они

также предоставили мне множество специфических проверок, отличных идей и исправленных ошибок.

14.4 Числа

В июле 1992, после месяцев тестирований альфа-версий, я выпустил Autoconf 1.0, и преобразовал много утилит GNU для его использования. Я был очень удивлен положительной реакцией на выпуск пакета. Так много людей стало использовать его, так что я не смог отслеживать их, включая людей, работающих над программным обеспечением, которое не является частью проекта GNU (например, TCL, FSP и Kerberos V5). Autoconf продолжал быстро развиваться, поскольку множество людей, использующих `configure`, писали мне о проблемах, с которыми встретились.

Autoconf превратился в хороший тест реализаций `m4`. UNIX `m4` начали выдавать дампы памяти, из-за длины макросов, определяемых Autoconf; несколько ошибок было найдено в GNU `m4`. В конечном счете мы осознали, что нам необходимо использовать некоторые возможности, которые имеются только в GNU `m4`. В частности, версия 4.3BSD `m4` имела слишком маленький набор встроенных макросов; версия для System V немного лучше, но все равно не предоставляла всех нужных нам возможностей.

Происходила дальнейшая разработка по мере того, как люди Autoconf в более жесткие условия (и использовали так, как я не ожидал). Karl Berry добавил проверки для X11. David Zuhn сделал поддержку C++. François Pinard сделал диагностику неправильных аргументов. Jim Blandy использовал пакет для настройки GNU Emacs, закладывая фундамент для некоторых последующих улучшений. Roland McGrath, использовавший пакет для библиотеки GNU C, написал скрипт `autoheader` для автоматизации создания шаблонов заголовочных файлов C, а также добавил ключ '`-verbose`' к `configure`. Noah Friedman добавил поддержку ключа '`-macrodir`' и переменной среды `AC_MACRODIR`. (Он также ввел в употребление термин *autoconfiscate*, который означает "адаптировать программное обеспечение для использования Autoconf".) Roland и Noah улучшили экранирование специальных символов в макросе `AC_DEFINE` и исправили множество ошибок, особенно когда я пресытился проблемами с переносимостью с февраля по июнь 1993.

14.5 Второзаконие

Постепенно накапливался длинный список важных возможностей, которыми хотелось бы пользоваться, а несколько лет, в течение которых множество людей накладывали на Autoconf заплатки, привели к накоплению всякого хлама, который невозможно было вычистить. В апреле 1994, в процессе работы на Cygnus Support, я начал полный пересмотр Autoconf. Я добавил большинство возможностей, которые отсутствовали в Autoconf, но присутствовали в Cygnus `configure` — в основном адаптируя некоторые части Cygnus `configure` с помощью David Zuhn и Ken Raeburn. Эти возможности включают поддержку использования '`config.sub`', '`config.guess`', '`-host`' и '`-target`'; создание ссылок на файлы; и запуск скриптов `configure` в подкаталогах. Добавление этих возможностей позволило Ken'у преобразовать для использования Autoconf GNU `as`, а Rob'y Savoye – DejaGNU.

Я добавил множество возможностей, о которых просили разные люди. Многие просили, чтобы скрипты `configure` могли использовать результаты проверок при последующих запусках, потому что (особенно при настройке большого дерева исходных текстов, как, например, делает Cygnus) они были ужасающе медленны. Mike Haertel предложил добавить специфические для машин скрипты инициализации. Люди, распространяющие программное обеспечение, которое будет работать на MS-DOS, просили предоставить возможность переопределения расширений '`.in`' в именах файлов, из-за чего появлялись имена типа '`config.h.in`', содержащие две точки. Jim Avera сделал обширное исследование проблем с экранированием кавычек в макросах `AC_DEFINE` и `AC_SUBST`; его проницательность привела к значительным улучшениям. Richard Stallman просил, чтобы вывод компилятора посылался в файл '`config.log`' вместо '`/dev/null`', чтобы помочь людям отлаживать скрипты `configure` для Emacs.

Я сделал некоторые изменения, потому что был недоволен качеством программы. Я сделал сообщения о результатах проверок менее двусмысленными, и сделал так, чтобы эти сообщения всегда выдавались. Я подправил имена макросов и подправил несовместимости со стандартами кодирования. Я добавил некоторые вспомогательные утилиты, которые были разработаны, чтобы помочь в адаптации пакетов для использования Autoconf. С помощью François Pinard, я сделал так, чтобы макросы не прерывали другие сообщения других макросов. (Эта возможность вывела на чистую воду некоторые узкие места в производительности GNU `m4`, которые были поспешно исправлены!) Я реорганизовал документацию, чтобы она вращалась вокруг тех самых проблем, которые люди и хотели решить. И я начал создавать набор тестов, поскольку опыт показал, что Autoconf имеет ярко выраженную тенденцию к регрессу при изменениях.

Опять же, несколько альфа-тестеров дали бесценную информацию, особенно François Pinard, Jim Meyering, Karl Berry, Rob Savoye, Ken Raeburn и Mark Eichin.

В конце концов, версия 2.0 была готова. И было много радости по этому поводу. (И у меня опять появилось свободное время. Кажется. Нет, я уверен!)

15 Старые имена макросов

В Autoconf версии 2, большинство макросов было переименовано для использования более обобщенной и информативной схемы наименования. Вот старые имена макросов, которые были переименованы, и новые имена, которым они соответствуют. Хотя старые имена все равно можно использовать с программой `autoconf` для обратной совместимости, старые имена макросов считаются устаревшими. Для описания новой схемы наименования См. Раздел 7.2 [Имена макросов], с. 62.

`AC_ALLOCA`
: `AC_FUNC_ALLOCA`

`AC_ARG_ARRAY`
: удален из-за ограниченной полезности

`AC_CHAR_UNSIGNED`
: `AC_C_CHAR_UNSIGNED`

`AC_CONST` `AC_C_CONST`

`AC_CROSS_CHECK`
: `AC_C_CROSS`

`AC_ERROR` `AC_MSG_ERROR`

`AC_FIND_X`
: `AC_PATH_X`

`AC_FIND_XTRA`
: `AC_PATH_XTRA`

`AC_FUNC_CHECK`
: `AC_CHECK_FUNC`

`AC_GCC_TRADITIONAL`
: `AC_PROG_GCC_TRADITIONAL`

`AC_GETGROUPS_T`
: `AC_TYPE_GETGROUPS`

`AC_GETLOADAVG`
: `AC_FUNC_GETLOADAVG`

`AC_HAVE_FUNCS`
: `AC_CHECK_FUNCS`

`AC_HAVE_HEADERS`
: `AC_CHECK_HEADERS`

`AC_HAVE_POUNDBANG`
: `AC_SYS_INTERPRETER` (отличающееся соглашение по вызову)

`AC_HEADER_CHECK`
: `AC_CHECK_HEADER`

`AC_HEADER_EGREP`
: `AC_EGREP_HEADER`

AC_INLINE
: AC_C_INLINE

AC_LN_S
: AC_PROG_LN_S

AC_LONG_DOUBLE
: AC_C_LONG_DOUBLE

AC_LONG_FILE_NAMES
: AC_SYS_LONG_FILE_NAMES

AC_MAJOR_HEADER
: AC_HEADER_MAJOR

AC_MINUS_C_MINUS_O
: AC_PROG_CC_C_O

AC_MMAP
: AC_FUNC_MMAP

AC_MODE_T
: AC_TYPE_MODE_T

AC_OFF_T
: AC_TYPE_OFF_T

AC_PID_T
: AC_TYPE_PID_T

AC_PREFIX
: AC_PREFIX_PROGRAM

AC_PROGRAMS_CHECK
: AC_CHECK_PROGS

AC_PROGRAMS_PATH
: AC_PATH_PROGS

AC_PROGRAM_CHECK
: AC_CHECK_PROG

AC_PROGRAM_EGREP
: AC_EGREP_CPP

AC_PROGRAM_PATH
: AC_PATH_PROG

AC_REMOTE_TAPE
: удален из-за ограниченной полезности

AC_RESTARTABLE_SYSCALLS
: AC_SYS_RESTARTABLE_SYSCALLS

AC_RETSIGTYPE
: AC_TYPE_SIGNAL

AC_RSH
: удален из-за ограниченной полезности

AC_SETVBUF_REVERSED
: AC_FUNC_SETVBUF_REVERSED

```
AC_SET_MAKE
          AC_PROG_MAKE_SET
AC_SIZEOF_TYPE
          AC_CHECK_SIZEOF
AC_SIZE_T
          AC_TYPE_SIZE_T
AC_STAT_MACROS_BROKEN
          AC_HEADER_STAT
AC_STDC_HEADERS
          AC_HEADER_STDC
AC_STRCOLL
          AC_FUNC_STRCOLL
AC_ST_BLKSIZE
          AC_STRUCT_ST_BLKSIZE
AC_ST_BLOCKS
          AC_STRUCT_ST_BLOCKS
AC_ST_RDEV
          AC_STRUCT_ST_RDEV
AC_SYS_SIGLIST_DECLARED
          AC_DECL_SYS_SIGLIST
AC_TEST_CPP
          AC_TRY_CPP
AC_TEST_PROGRAM
          AC_TRY_RUN
AC_TIMEZONE
          AC_STRUCT_TIMEZONE
AC_TIME_WITH_SYS_TIME
          AC_HEADER_TIME
AC_UID_T  AC_TYPE_UID_T
AC_UTIME_NULL
          AC_FUNC_UTIME_NULL
AC_VFORK  AC_FUNC_VFORK
AC_VPRINTF
          AC_FUNC_VPRINTF
AC_WAIT3  AC_FUNC_WAIT3
AC_WARN   AC_MSG_WARN
AC_WORDS_BIGENDIAN
          AC_C_BIGENDIAN
AC_YYTEXT_POINTER
          AC_DECL_YYTEXT
```

Индекс переменных среды

Вот алфавитный список переменных среды, которые проверяет Autoconf.

A

C

S

Индекс выходных переменных

Это алфавитный список переменных, которые Autoconf может подставлять в файлы, которые он создает, обычно это один или несколько файлов ‘Makefile’. Для получения информации о том, как это делается См. Раздел 6.2 [Установка выходных переменных], с. 56.

R

S

T

X

Y

Индекс символов препроцессора

Вот алфавитный список символов препроцессора C, которые определяют макросы Autoconf. Для работы с Autoconf, исходному коду на языке C необходимо использовать эти имена в директивах `#if`.

default

C

D

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W

Y

Индекс макросов

Вот алфавитный список макросов Autoconf. Для того, чтобы сделать список читабельнее, макросы перечислены без префикса ‘AC_’.

A

B

C

D

E

F

G

H

I

L

M

O

P

R

S

T

U

V

W

X

Y

Оглавление