

# make(1)

## НАЗВАНИЕ

make - сопровождение, обновление и пересоздание групп программ

## СИНТАКСИС

```
make [-f имя_файла] [-Beiknpqrstuw] [имена]
```

## ОПИСАНИЕ

Утилита **make** позволяет программисту сопровождать, обновлять и пересоздавать группы компьютерных программ. **make** выполняет команды из *файла управления проектом* (makefile) для обновления одного или нескольких целевых **имен** (**имена** обычно представляют собой программы). Если опция **-f** не указана, последовательно делаются попытки использовать **makefile**, **Makefile** и файлы *системы управления исходным кодом* (Source Code Control System - SCCS) **s.makefile** и **s.Makefile**. Если в качестве **makefile** указан -, берется стандартный входной поток. Можно указывать несколько пар аргументов **-f makefile**.

**make** обновляет цель, только если цели, от которых она зависит, более новые, чем сама цель. Все необходимые для построения цели файлы рекурсивно добавляются к списку целей. Отсутствующие файлы предполагаются устаревшими.

В файлы управления проектом можно включать следующий список директив, изменяющих поведение **make**. Они используются в файлах управления проектами как цели:

- |                   |   |
|-------------------|---|
| <b>.DEFAULT:</b>  | Если файл должен быть сделан, но для этого нет явных команд или соответствующих правил построения, используются команды, ассоциированные с целью <b>.DEFAULT</b> , если она существует. |
| <b>.IGNORE:</b>   | Действует так же, как опция <b>-i</b> .   |
| <b>.MUTEX:</b>    | Устанавливает последовательность обновления указанных целей (см. подраздел " <a href="#">Параллельный make</a> " ниже).   |
| <b>.PRECIOUS:</b> | Цели, от которых зависит запись <b>.PRECIOUS</b> , не будут удалены при выходе или нажатии клавиши выхода или прерывания.   |
| <b>.SILENT:</b>   | Действует так же, как опция <b>-s</b> .   |

Опции **make** перечислены ниже:

- |                     |  |
|---------------------|--|
| <b>-B</b>           | Разбивать результаты выполнения параллельного <b>make</b> на соответствующие блоки для удобства чтения.                                    |
| <b>-e</b>           | Переменные среды переопределяют присваивания в файлах управления проектами.  |
| <b>-f имя_файла</b> | Указывает имя файла управления проектом.   |
| <b>-i</b>           | Игнорировать коды ошибок, возвращаемые вызываемыми программами.  |
| <b>-k</b>           | Отказаться от обработки текущей записи, если она закончилась неудачно, но продолжить обработку других ветвей, не зависящих от этой записи. |
| <b>-n</b>           | Режим без выполнения. Команды выдаются, но не выполняются. Печатаются даже командные строки, начинающиеся с <b>@</b> .                     |
| <b>-p</b>           | Выдать полный набор макроопределений и описаний целей.   |

<b>-P</b>	Обновлять одновременно несколько целей. Количество одновременно обновляемых целей определяется переменной среды <b>PARALLEL</b> и наличием директив <b>.MUTEX</b> в файлах управления проектами.
<b>-q</b>	Запрос. <b>make</b> возвращает нулевой или ненулевой код возврата в зависимости от того, обновлен или нет целевой файл.
<b>-r</b>	Не использовать встроенные правила.
<b>-s</b>	"Молчаливый" режим. Не выдавать строки команд перед выполнением.
<b>-t</b>	Обновить дату последнего изменения ("затронуть" - <b>touch</b> ) у целевых файлов (тем самым они будут считаться обновленными) вместо выполнения обычных команд.
<b>-u</b>	Безусловно пересоздать цели, игнорируя все временные отметки.
<b>-w</b>	Подавить выдачу предупреждений. Фатальные сообщения по-прежнему будут выдаваться.

## Создание файла управления проектом (makefile)

*Файл управления проектом*, указываемый в опции **-f** (или используемый по умолчанию), - это файл специальной структуры, содержащий явные инструкции по обновлению и пересозданию программ, а также последовательность записей, определяющих зависимости. Первая строка записи представляет собой непустой список целей через пробел, за которым идет двоеточие (:), а затем (возможно пустой) список требуемых файлов или файлов, от которых зависит цель. Текст после ; и все последующие строки, начинающиеся с символа табуляции, являются командами интерпретатора **shell**, которые необходимо выполнить для обновления цели. Первая непустая строка, не начинающаяся с символа табуляции или #, задает новую зависимость или определение макроса. Команды интерпретатора **shell** можно переносить на следующую строку, указывая *обратную косую* (\) перед переводом строки. Все введенное в команде (за исключением начального символа табуляции) передается непосредственно интерпретатору без изменений. Так,

```
echo a\  
b
```

выдаст

```
ab
```

точно так же, как и в командном интерпретаторе.

Комментарии начинаются со знака # и продолжаются до перевода строки; при этом в них включаются последовательности обратная косая-перевод строки.

Следующий файл управления проектом утверждает, что **pgm** зависит от двух файлов, **a.o** и **b.o**, которые, в свою очередь, зависят от соответствующих файлов с исходным кодом (**a.c** и **b.c**) и общего заголовочного файла **incl.h**:

```
pgm: a.o b.o  
cc a.o b.o -o pgm  
a.o: incl.h a.c  
cc -c a.c  
b.o: incl.h b.c  
cc -c b.c
```

Строки команд выполняются по очереди, каждая в отдельном командном интерпретаторе. Чтобы указать, какой командный интерпретатор должна использовать утилита **make** для выполнения команд, можно использовать переменную среды **SHELL**. По умолчанию используется **/usr/bin/sh**. Первым одним или двумя символами команды могут быть: @, -, @- или -@. Если указан символ @, выдача команды подавляется. Если указан символ -, **make** игнорирует ошибку. При выполнении каждая строка печатается, если только не указана опция **-s**, в файле управления проектом нет директивы **.SILENT**: и если начальная последовательность символов команды не содержит символа @. Опция **-n** вызывает печать без выполнения; однако, если командная строка содержит подстроку **\$(MAKE)**, она всегда выполняется (см. обсуждение

макроса **MAKEFLAGS** в разделе "Среда" ниже). Опция **-t** (затронуть) обновляет дату модификации файла, не выполняя никаких команд.

Команды, возвращающие ненулевой код возврата, обычно прерывают работу **make**. Если указана опция **-i**, в файле управления проектом имеется запись **.IGNORE:** или если начальная последовательность символов содержит символ **-**, ошибка игнорируется. Если указана опция **-k**, обработка текущей записи прекращается, но продолжается для других ветвей, не зависящих от соответствующей цели.

Прерывание или выход ведет к удалению цели, если только цель не указана в списке зависимостей директивы **.PRECIOUS**.

## Параллельный make

Если утилита **make** вызывается с опцией **-P**, она путается строить одновременно, параллельно несколько целей. (Это делается с помощью стандартного механизма многозадачности системы UNIX, который позволяет одновременно выполнять несколько процессов.) Для файла управления проектом, показанного в примере в предыдущем разделе, будут созданы процессы для параллельного построения **a.o** и **b.o**. После завершения процессов будет построен файл **pgm**.

Количество целей, которое **make** попытается построить параллельно, определяется значением переменной среды **PARALLEL**. Если указана опция **-P**, но переменная **PARALLEL** не установлена, **make** будет пытаться построить не более двух целей одновременно.

Для указания определенной последовательности обновления целей можно использовать директиву **.MUTEX**. Эта возможность пригодится, когда две или более цели изменяют общий файл вывода, например, при добавлении модулей в библиотеку или при создании промежуточного файла с одним и тем же именем, как делают утилиты **lex** и **yacc**. Если бы файл управления проектом в примере из предыдущего раздела содержал директиву **.MUTEX** вида

```
.MUTEX: a.o b.o
```

это не позволило бы **make** строить **a.o** и **b.o** параллельно.

## Среда

Утилита **make** читает переменные среды. Предполагается, что все переменные являются определениями макросов, и они обрабатываются соответствующим образом. Переменные среды обрабатываются до любого файла управления проектом и после применения внутренних правил; поэтому определения макросов в файле управления проектом имеют приоритет над переменными среды. При указании опции **-e** переменные среды имеют приоритет над определениями макросов в файле управления проектом. Суффиксы и связанные с ними правила в файле управления проектом имеют приоритет над соответствующими суффиксами во встроенных правилах.

Переменная среды **MAKEFLAGS** обрабатывается **make** как содержащая любые допустимые опции (кроме **-f**, **-p** и **-r**), определяемые в командной строке. Более того, при вызове **make** проверяет наличие переменной в среде, помещает в нее текущие опции и передает ее в вызовы команд. Таким образом, **MAKEFLAGS** всегда содержит текущие введенные опции. Эта возможность оказалась очень полезной для "супер-**make**ов". Фактически, как упоминалось выше, при использовании опции **-n** команда **\$(MAKE)** все равно выполняется; таким образом, можно выполнить **make -n** рекурсивно для всей программной системы, чтобы увидеть, что будет выполняться. Такого результата можно достичь потому, что опция **-n** помещается в **MAKEFLAGS** и

передается последующим вызовам **\$(MAKE)**. Такое использование **make** представляет собой один из способов отладки всех файлов управления для программного проекта без фактического выполнения требуемых действий.

## Включаемые файлы

Если в качестве первых семи символов строки в файле управления проектом указана строка **include** и после нее идет пробел или символ табуляции, предполагается, что остаток строки представляет собой имя файла, который и будет прочитан текущим вызовом после подстановки макросов.

## Макросы

Записи вида **строка1 = строка2** являются определениями макросов. **строка2** определяется как все символы до символа комментария или незамаскированного перевода строки. Последующие вхождения **\$(строка1[:подст1=[подст2]])** заменяются **строкой2**. Скобки необязательны, если используется односимвольное имя макроса и нет последовательности подстановки. Необязательная конструкция **:подст1=подст2** является *последовательностью подстановки* (substitute sequence). Если она указана, все не перекрывающиеся вхождения **подст1** в указанном макросе заменяются на **подст2**. Строки (для подстановки такого рода) ограничиваются пробелами, символами табуляции и перевода строки и началом строк. Пример использования последовательности подстановки показан в разделе " ниже.

## Внутренние макросы

Имеется пять поддерживаемых внутренне макросов, полезных при написании правил построения целей.

- \$\*** Макрос **\$\*** заменяется именем текущего файла из списка зависимостей, от которого отброшен суффикс. Он заменяется только в правилах вывода.
- \$@** Макрос **\$@** заменяется на полное имя текущей цели. Он заменяется только для явно указанных зависимостей.
- \$<** Макрос **\$<** заменяется только для правил вывода или правила **.DEFAULT**. Это модуль, устаревший по отношению к цели (построенное имя файла, от которого зависит цель). Так, в правиле **.c.o** макрос будет заменен на файл **.c**. Вот пример построения оптимизированных файлов **.o** из файлов **.c**:

```
.c.o: cc -c -O $*.c
```

или:

```
.c.o: cc -c -O $<
```

- \$?** Макрос **\$?** заменяется при обработке явных правил в файле управления проектом. Это список необходимых файлов, устаревших по отношению к цели, или, по сути, модулей, которые необходимо перестроить.
- \$%** Макрос **\$%** заменяется, только когда цель является элементом архивной библиотеки вида **lib(file.o)**. В этом случае, **\$@** заменяется на библиотеку (**lib**), а **\$%** заменяется на элемент библиотеки, **file.o**.

Четыре из пяти макросов имеют альтернативные формы. Если к любому из этих четырех макросов добавляется прописная буква **D** или **F**, его значение заменяется на полное имя каталога для **D** и только имя файла для **F**. Таким образом, **\$(@D)** ссылается на определяющую каталог часть строки **\$@**. Если определяющей каталог части нет, генерируется **./**. Альтернативной формы не имеет только макрос **\$?**.

## Суффиксы

Для определенных имен (например, заканчивающихся на **.o**) известны типичные файлы, от которых они зависят (*исходные файлы*), такие как **.c**, **.s** и так далее. Если в файле управления проектом нет команд обновления для такого файла и если существует соответствующий исходный файл, он компилируется для создания цели. В этом случае **make** использует правила вывода, позволяющие строить файлы из других файлов путем проверки суффиксов и определения подходящего внутреннего правила построения. В настоящее время поддерживаются следующие стандартные правила вывода:

<b>.c</b>	<b>.c~</b>	<b>.f</b>	<b>.f~</b>	<b>.s</b>	<b>.s~</b>	<b>.sh</b>	<b>.sh~</b>	<b>.C</b>	<b>.C~</b>
<b>.c.a</b>	<b>.c.o</b>	<b>.c~.a</b>	<b>.c~.c</b>	<b>.c~.o</b>	<b>.f.a</b>	<b>.f.o</b>	<b>.f~.a</b>	<b>.f~.f</b>	<b>.f~.o</b>
<b>.h~.h</b>	<b>.l.c</b>	<b>.l.o</b>	<b>.l~.c</b>	<b>.l~.l</b>	<b>.l~.o</b>	<b>.s.a</b>	<b>.s.o</b>	<b>.s~.a</b>	<b>.s~.o</b>
<b>.s~.s</b>	<b>.sh~.sh</b>	<b>.y.c</b>	<b>.y.o</b>	<b>.y~.c</b>	<b>.y~.o</b>	<b>.y~.y</b>	<b>.C.a</b>	<b>.C.o</b>	<b>.C~.a</b>
<b>.C~.C</b>	<b>.C~.o</b>	<b>.L.C</b>	<b>.L.o</b>	<b>.L~.C</b>	<b>.L~.L</b>	<b>.L~.o</b>	<b>.Y.C</b>	<b>.Y.o</b>	<b>.Y~.C</b>
<b>.Y~.o</b>	<b>.Y~.Y</b>								

Внутренние правила **make** содержатся в исходном файле **rules.c** программы **make**. Эти правила можно изменить. Для получения скомпилированных в **make** правил на любой машине в форме, подходящей для перекompиляции, используется следующая команда:

```
make -pf - 2>/dev/null
```

Тильда в перечисленных выше правилах относится к файлу SCCS (см. **sccsfile(4)**). Так, правило **.c~.o** преобразует исходный файл C из SCCS в объектный файл (**.o**). Поскольку система управления исходным кодом SCCS использует префикс **s**, она несовместима с подходом **make**, основанном на суффиксах. Тем не менее, с помощью тильды можно заменить ссылку на файл ссылкой на соответствующий файл SCCS.

Правило с единственным суффиксом (например, **.c:**) определяет, как построить **x** из **x.c**. Фактически, второй суффикс пустой. Эта возможность пригодится для построения целей из одного исходного файла, например, процедур командного интерпретатора или простых программ на C.

Дополнительные суффиксы указываются в виде списка зависимостей для **.SUFFIXES**. Порядок здесь имеет значение: в качестве исходного файла используется первое возможное имя, для которого существует как файл, так и правило построения. По умолчанию используется список:

```
.SUFFIXES: .o .c .c~ .y .y~ .l .l~ .s .s~ .sh .sh~ .h .h~
.f .f~ .C .C~ .Y .Y~ .L .L~
```

И в этом случае приведенная выше команда печати внутренних правил покажет реализованный на данной машине список суффиксов. Несколько списков суффиксов объединяются в один; **.SUFFIXES:** без списка зависимостей очищает список суффиксов.

## Правила вывода

Первый пример можно реализовать короче.

```
pgm: a.o b.o
cc a.o b.o -o pgm
a.o b.o: incl.h
```

Такое сокращение возможно, потому что **make** имеет ряд внутренних правил построения файлов. Пользователь может добавлять правила к этому списку, просто задавая их в файле управления проектом.

Некоторые макросы используются стандартными правилами вывода, чтобы можно было включать дополнительные опции в результирующие команды. Например, **CFLAGS**, **LFLAGS** и **YFLAGS** используются в качестве опций компилятора для **cc(1)**, **lex(1)** и **yacc(1)**, соответственно. Можно опять порекомендовать описанный выше метод просмотра текущих правил.

Поиском исходных файлов можно управлять. Правило создания файла с суффиксом **.o** из файла с суффиксом **.c** задается как запись с **.c.o**: в качестве цели и с пустым списком зависимостей. Команды интерпретатора shell, связанные с целью, определяют правило создания файла **.o** из файла **.c**. Любая цель, не содержащая наклонных и начинающаяся с точки рассматривается как правило, а не как обычная цель.

## Библиотеки

Если имя цели или одного из элементов списка зависимостей содержит круглые скобки, считается, что это архивная библиотека, а строка в скобках ссылается на элемент библиотеки. Так, **lib(file.o)** и **\$(LIB)(file.o)** оба ссылаются на архивную библиотеку, содержащую **file.o**. (В этом примере предполагается, что макрос **LIB** предварительно определен.) Выражение **\$(LIB)(file1.o file2.o)** недопустимо. Правила, относящиеся к архивным библиотекам, имеют вид **.XX.a**, где **XX** - это суффикс, из файла с которым должен строиться элемент архива. Неприятный побочный эффект текущей реализации требует, чтобы **XX** отличался от суффикса элемента архива. Поэтому нельзя, чтобы **lib(file.o)** зависел от **file.o** явно. Чаще всего архивы создаются так, как показано ниже. Предполагается, что все исходные файлы написаны на языке C:

```
lib:      lib(file1.o) lib(file2.o) lib(file3.o)
@echo lib is now up-to-date
.c.a:
$(CC) -c $(CFLAGS) $<
$(AR) $(ARFLAGS) $@ $(
```

Фактически, правило **.c.a**, указанное выше, встроено в **make** и в этом примере не обязательно. Ниже приведен более интересный, но и более ограниченный пример конструкции создания архивной библиотеки:

```
lib:      lib(file1.o) lib(file2.o) lib(file3.o)
$(CC) -c $(CFLAGS) $(?:.o=.c)
$(AR) $(ARFLAGS) lib $?
rm $?
@echo lib is now up-to-date
.c.a:;
```

Здесь использован режим подстановки при расширении макроса. По определению **\$?** является списком имен объектных файлов (в **lib**), исходные тексты на C для которых устарели. Подстановка заменяет **.o** на **.c**. (К сожалению, нельзя заменить на **.c~**; однако, такое преобразование может стать возможным в дальнейшем.) Также обратите внимание на отключение правила **.c.a**., которое создавало бы объектные файлы один за другим. Эта конструкция существенно ускоряет работу с архивными библиотеками. Такого типа конструкции становятся очень громоздкими, если архивная библиотека содержит смесь программ на ассемблере и на C.

## ФАЙЛЫ

**[Mm]akefile**  
**s.[Mm]akefile**

**/usr/bin/sh**

**/usr/lib/locale/локаль/LC\_MESSAGES/uxcpu**

файл сообщений для используемого языка (См. **LANG** в **environ(5)**.)

## ССЫЛКИ

[cc\(1\)](#), [cd\(1\)](#), [fprintf\(3S\)](#), [lex\(1\)](#), [scsfile\(4\)](#), [sh\(1\)](#), [yacc\(1\)](#)

## ПРИМЕЧАНИЯ

Некоторые команды возвращают ненулевой код не тогда, когда нужно; чтобы обойти это, используйте **\_i** или префикс **-** в строке команды.

Файлы с именами, содержащими символы `=`, `:` и `@`, не будут правильно обрабатываться. Команды, непосредственно выполняемые командным интерпретатором, в частности **`cd(1)`**, действуют в **`make`** только в пределах строки. Синтаксис **`lib(file1.o file2.o file3.o)`** недопустим. Нельзя строить **`lib(file.o)`** из **`file.o`**.

Copyright 1994 Novell, Inc.  
Copyright 2000 В. Кравчук, [OpenXS Initiative](#), перевод на русский язык