

Встроенная команда mapfile

Краткое описание

```
mapfile [-n COUNT] [-O ORIGIN] [-s COUNT] [-t] [-u FD] [-C ОБРАТНЫЙ ВЫЗОВ] [-c КВАНТОВЫЙ] [МАССИВ]
```

```
readarray [-n COUNT] [-O ORIGIN] [-s COUNT] [-t] [-u FD] [-C ОБРАТНЫЙ ВЫЗОВ] [-c КВАНТОВЫЙ] [МАССИВ]
```

Описание

Эта встроенная команда также доступна с помощью имени команды `readarray`.

`mapfile` является одной из двух встроенных команд, в первую очередь предназначенных для обработки стандартного ввода (другой является `read`). `mapfile` считывает строки стандартного ввода и присваивает каждую элементам индексированного массива. Если имя массива не задано, по умолчанию используется имя массива `MAPFILE`. Целевой массив должен быть "обычным" целочисленным индексированным массивом.

`mapfile` возвращает успех (0), если не задана недопустимая опция или данный массив `ARRAY` не задан только для чтения.

Опция	Описание
-c QUANTUM	Определяет количество строк, которые должны быть прочитаны между каждым вызовом обратного вызова, указанным с -C помощью. КВАНТ по умолчанию равен 5000
-C CALLBACK	Задаёт обратный вызов. Строка CALLBACK может быть любым кодом оболочки, индексом массива, который будет назначен, и строка добавляется во время оценки.
-n COUNT	Считывает не более COUNT строк, затем завершается. Если COUNT равно 0, то считываются все строки (по умолчанию).
-O ORIGIN	Начинает заполнение данного массива ARRAY по индексу ORIGIN, а не очищает его и начинается с индекса 0.
-s COUNT	Отбрасывает первые COUNT прочитанные строки.

Опция	Описание
-t	Удалите любую завершающую новую строку из прочитанной строки, прежде чем она будет назначена элементу массива.
-u FD	Чтение из filedescriptor FD вместо стандартного ввода.

Хотя `mapfile` это не обычная или переносимая функция оболочки, ее функциональность будет знакома многим программистам. Почти все языки программирования (кроме оболочек) с поддержкой составных типов данных, таких как массивы, и которые обрабатывают открытые файловые объекты традиционным способом, имеют некоторый аналогичный ярлык для легкого чтения всех строк некоторого ввода в качестве стандартной функции. В Bash `mapfile` сам по себе не может делать ничего, что уже нельзя было бы сделать с помощью чтения и цикла, и если переносимость вызывает даже небольшую озабоченность, ее никогда не следует использовать. Тем не менее, она *значительно* превосходит цикл чтения и может сделать код короче и чище, что особенно удобно для интерактивного использования.

Примеры

Вот реальный пример интерактивного использования, заимствованный из Gentoo workflow. Обновления Xorg требуют перестройки драйверов, а предложенная Gentoo команда далека от идеала, поэтому давайте ее доработаем. Первая команда создает список пакетов, по одному на строку. Мы можем прочитать их в массиве с именем "args", используя `mapfile`, удаляя завершающие строки с помощью опции '-t'. Полученный массив затем расширяется в аргументы команды "emerge" - интерфейс к менеджеру пакетов Gentoo. Этот тип использования может обеспечить безопасную и эффективную замену `xargs(1)` в определенных ситуациях. В отличие от `xargs`, все аргументы гарантированно передаются при одном вызове команды без разбиения на слова, расширения имени пути или других глупостей.

```
# eix --only-names -IC x11-drivers | { mapfile -t args; emerge -av1  
"${args[@]}" <&1; }
```

Обратите внимание на использование группировки команд, чтобы сохранить команду `emerge` внутри подоболочки канала и в пределах "аргументов". Также обратите внимание на необычное перенаправление. Это потому, что флаг -a делает `emerge` интерактивным, запрашивая у пользователя подтверждение перед продолжением и проверяя с помощью `isatty(3)` прерывание, если `stdin` не указан на терминале. Поскольку `stdin` всей группы команд по-прежнему поступает из канала, даже если `mapfile` прочитал все доступные входные данные, мы просто заимствуем FD 1, поскольку так получилось, что он указывает туда, куда мы хотим. Подробнее об этом в вики `greycat`: <http://mywiki.woolledge.org/BashFAQ/024> (<http://mywiki.woolledge.org/BashFAQ/024>)

Обратный вызов

Это одна из наиболее необычных функций встроенной команды Bash. Насколько я могу судить, точное поведение выглядит следующим образом: если определено, при чтении каждой строки код, содержащийся в строковом аргументе флага -C, вычисляется и выполняется *перед* назначением каждого элемента массива. Для этой строки нет ограничений, которая может быть любым произвольным кодом, однако два дополнительных "слова" автоматически добавляются в конец перед вычислением: индекс и соответствующая строка данных, которые будут назначены следующему элементу массива. Поскольку все это происходит до назначения, функция обратного вызова не может быть использована для изменения назначаемого элемента, хотя она может считывать и изменять любые уже назначенные элементы массива.

Очень простым примером может быть использование ее в качестве индикатора выполнения. Это выведет точку для каждой прочитанной строки. Обратите внимание на экранированный комментарий, чтобы скрыть добавленные слова из printf.

```
$ printf '%s\n' {1..5} | mapfile -c 1 -C 'printf . \#' )
.....
```

Действительно, предполагаемое использование заключается в том, чтобы обратный вызов просто содержал имя функции с дополнительными словами, передаваемыми ей в качестве аргументов. Если вы вообще собираетесь использовать обратные вызовы, это, вероятно, лучший способ, потому что он обеспечивает легкий доступ к аргументам без уродливого "кода в строке".

```
$ foo() { echo "|$1|"; }; mapfile -n 11 -c 2 -C 'foo'
|2|
|4|
и т.д..
```

Для полноты картины, вот несколько более сложных примеров, вдохновленных вопросом, заданным в #bash - как добавлять что-то к каждой строке некоторого ввода, а затем выводить четные и нечетные строки в отдельные файлы. Это далеко не лучший возможный ответ, но, надеюсь, иллюстрирует поведение обратного вызова:

```
$ { printf 'ввод%s \n' {1..10} | mapfile -c 1 -C '>&$(( ${#} x[@] } % 2) + 3 )) printf -- "%.sprefix %s" x; } 3>outfile04>outfile1
$ cat outfile{0,1}
префиксный ввод1
префиксный ввод3
префиксный ввод5
префиксный ввод7
префиксный ввод9
префиксный ввод2
префиксный ввод4
префиксный ввод6
префиксный ввод8
префиксный ввод10
```

Поскольку синтаксически перенаправления разрешены в любом месте команды, мы помещаем его перед printf, чтобы избежать дополнительных аргументов. Вместо того, чтобы открывать "outfile<n>" для добавления при каждом вызове путем вычисления имени файла, сначала откройте FD для каждого и вычислите, в какой FD отправлять выходные данные, измеряя размер $x \bmod 2$. Спецификация формата нулевой ширины используется для поглощения аргумента index number .

Другим вариантом может быть добавление каждой из этих строк к элементам отдельных массивов. Я оставлю разбор этого в качестве упражнения для читателя. Это довольно хак, но иллюстрирует некоторые интересные свойства printf -v и mapfile -C (которые вам, вероятно, никогда не следует использовать в реальном коде).

```
$ y=( 'нечетный [j]' 'четный[j++]' ); printf 'ввод%s\n' {1..10} | { m
apfile -tc 1 -C 'printf -v "${y[${#x[@}]} % 2}]" -- "%.sprefix %s" ' x;
printf '%s \n' "${нечетный[@]}" " "${четный[@]}"; }
префиксный ввод1
префиксный ввод3
префиксный ввод5
префиксный ввод7
префиксный ввод9

префиксный ввод2
префиксный ввод4
префиксный ввод6
префиксный ввод8
префиксный ввод10
```

Этот пример, основанный на еще одном вопросе #bash, иллюстрирует mapfile в сочетании с read . Пример ввода - это heredoc to main . Цель состоит в том, чтобы создать "структуру" на основе записей во входном файле, состоящую из чисел, следующих за двоеточием в каждой строке. Каждая 3-я строка - это ключ, за которым следуют 2 соответствующих поля. Функция showRecord принимает ключ и возвращает запись.

```
#!/usr/bin/env bash

showRecord() {
printf 'ключ[%d] = %d, %d \n' "$1" "${vals[@]:ключи[$1]*2:2}"
}

parseRecords() {
trap 'unset -f _f' ВОЗВРАЩАЕТ
_f() {
локальный x
IFS=: read -r _ x
((ключи[x]=n++))
}
локальный n

_f
mapfile -tc2 -c _f "$1"
вычисление "$1"=( "${'"$1"'[@]##*:}" )' # Возвращает массив с некотор
ыми изменениями
}

main() {
local -a ключи vals
parseRecords vals
showRecord "$1"
}

главная "$1" <<- "EOF"
fabric.domain:123
routex:1
routey:2
fabric.domain:321
routex:6
routey:4
EOF
```

Например, запуск `scriptname 321` приведет к результату `key[321] = 6, 4`.
Каждые 2 прочитанные строки вызывается `mapfile` функция `_f`, которая считывает одну дополнительную строку. Поскольку первая строка в файле является ключом и `_f` отвечает за ключи, она вызывается первой, так что `mapfile` начинается с чтения второй строки ввода, вызывая `_f` с каждой последующей 2 итерациями. Ловушка `ВОЗВРАТА` не имеет значения.

Ошибки

- Ранние реализации были с ошибками. Например, `mapfile` заполнение буфера истории строк чтения вызовами `CALLBACK`. Это было исправлено в 4.1 бета-версии.
- `mapfile -n` считывает дополнительную строку за последней строкой, назначенной массиву, через Bash. Исправлено в 4.2.35 ([ftp://ftp.gnu.org/gnu/bash/bash-4.2-patches/bash42-035](http://ftp.gnu.org/gnu/bash/bash-4.2-patches/bash42-035)).

- mapfile обратные вызовы могут вызвать сбой, если присваиваемая переменная обрабатывается определенным образом.
<https://lists.gnu.org/archive/html/bug-bash/2013-01/msg00039.html>
(<https://lists.gnu.org/archive/html/bug-bash/2013-01/msg00039.html>) . Исправлено в 4.3.

Чтобы сделать



- Создайте реализацию в виде функции оболочки, переносимой между Ksh, Zsh и Bash (и, возможно, другими оболочками, подобными bourne, с поддержкой массива).

Смотрите также

- Массивы
- Встроенная команда read - Если вы еще не знаете об этом, почему вы читаете эту страницу?
- <http://mywiki.woledge.org/BashFAQ/001>
(<http://mywiki.woledge.org/BashFAQ/001>) - Это [FAQ\(\)](#) 1 не просто так.



Обсуждение

 [commands/builtin/mapfile.txt](#)  Last modified: 2013/08/19 08:09 by ormaaj

This site is supported by Performing Databases - your experts for database administration

Bash Hackers Wiki



Except where otherwise noted, content on this wiki is licensed under the following license:
GNU Free Documentation License 1.3