

Устаревший и устаревший синтаксис

Эта (неполная) страница описывает некоторые синтаксисы и команды, которые в какой-то мере считаются устаревшими. Подробное обсуждение обоснования выходит за рамки этой страницы. См. Страницу переносимости для обсуждения проблем переносимости.

В этой первой таблице приведен синтаксис, который поддерживается Bash, но имеет мало законных применений, если таковые имеются. Эти функции существуют в основном для Bourne, csh или какой-либо другой обратной совместимости с устаревшими оболочками, или были специфичными для Bash функциями, которые считались неудачными экспериментами и устарели или были заменены лучшей альтернативой. Они должны быть неактуальны для большинства людей, за исключением, может быть, игроков в гольф. Новые скрипты никогда не должны их использовать. Ни один из элементов в этом списке не указан самой последней версией POSIX, а некоторые могут быть несовместимы с POSIX.

Синтаксис	Замена	Описание
<code>&>FILE</code> и <code>>&FILE</code>	<code>>FILE 2>&1</code>	Этот синтаксис переназначения является сокращением <code>2>&1</code> и происходит из <code>o</code> . Последняя форма <code>osof</code> необычна и никогда не используется, а явна использующая отдельный перенаправления, предпочтительнее обе ярлыки вносят путаницу дескриптор копирования поскольку синтаксис не

Синтаксис	Замена	Описание
		<p>также вносят дублирование синтаксического анализа, конфликтуют с POSIX. (без этой функции обрабатывается cmd1 &>file cmd2 как "фоновый cmd1 , а затем выполняются cmd2 с перенаправлением на стандартный file вывод является правильной интерпретацией этого в См.: перенаправление</p> <pre>\$ { bash; dash </; } <<<'echo foo>&>/dev/fd/2 echo bar foo echo bar bar</pre>
<code>\$(EXPRESSION)</code>	<code>\$((EXPRESSION))</code>	<p>Этот недокументированный синтаксис полностью соответствует POSIX арифметическим расширениям <code>\$((EXPRESSION))</code> . Он реализован почти везде Bash и Zsh. См. Арифметическое расширение. Некоторое обсуждение (http://lists.gnu.org/archive/bash/2012-04/msg00034</p>
<code>COMMAND & COMMAND</code>	<code>COMMAND 2>&1 COMMAND</code>	<p>Это альтернативный оператор конвейера, производный от устаревшего. Официально Bash не считается устаревшим, но я настоятельно рекомендую это делать, так как это конфликтует с оператором, используемым для создания совместного процесса в большинстве оболочек, также имеет запутанное поведение. Стандартный перенаправляется перенаправляется перенаправляется</p>

Синтаксис	Замена	Описание
		обычный канал, в то время как стандартный вывод перенаправляется после других перенапр предшествующих опер канала. В целом, это бессмысленное раздув синтаксиса. Вместо это используйте явное перенаправление.
<code>function NAME() COMPOUND-CMD</code>	<code>NAME() COMPOUND-CMD</code> или <code>function NAME { CMDS; }</code>	Это объединение опре функций в стиле Korn и использованием как <code>function</code> ключевого сл круглых скобок. У него полезной цели и нет основы или причины дл существования. Он не у POSIX. Он принимается mksh, zsh и, возможно, некоторыми другими об Korn, где он рассматри идентичный функции в POSIX. Он не принят А Его никогда не следует использовать. Ключевой приведен в следующе таблице <code>function</code> . У В этой функции, задокументированной к устаревшая.

<code>for x; { ...; }</code>	<code>do , done , in , esac , и т.д.</code>	Этот недокументирован синтаксис заменяет <code>do done</code> зарезервированн фигурными скобками. М оболочки Korn поддерж различные перестанов синтаксисе для опреде составных команд, таки <code>case</code> , и <code>while</code> . Какие некоторые детали, нап требуется ли перевод с точка с запятой. различ
------------------------------	---	---

Синтаксис	Замена	Описание
		for Работает, но не рекомендуется. Излишне говорить, что используйте его.
<div><div></div><div>В этой таблице перечислены синтаксисы, которые указаны в POSIX (если не указано иное ниже), но были заменены более совершенными альтернативами (либо в POSIX, Bash, либо в обоих), либо настоятельно не рекомендуется по другим причинам, таким как поощрение плохой практики или опасного кода. Те, которые указаны в POSIX, могут быть плохо спроектированы и неизменяемы по историческим причинам.</div></div>		
Синтаксис	Замена	Описание

Синтаксис	Замена	Описани
Расширения без кавычек, разделение слов и расширение пути (глобализация)	Правильное цитирование (http://mywiki.woolledge.org/Quotes), массивы в стиле ksh / Bash, расширение "\$ @", встроенная команда чтения	<i>Ошибки цитирования</i> - это распространённые ошибки, несколько неинтуитивных перенесённых из оболочки на сломанные скрипты и из документированном поведении важных расширений выполняются слева направо. Тем не менее, расширения, в первую очередь слова и глобирование, а в Bash, расширение в фигурных скобках по умолчанию выполняются с результатами предыдущих расширений, если они заключены в кавычки . Это процесс расширения переменных в обычном контексте аргумента значения переменной, может быть результатом в зависимости от неконтролируемых побочных эффектов, как значение IFS и имена файлов в рабочем каталоге. Вы не можете использовать глобализацию без разделения (без <code>set -f</code>). Вы не можете использовать список, разделённый символом переменной и безопасно обрабатывать расширение без кавычек (http://mywiki.woolledge.org/Bash). Возможно, всегда выбирайте вариант, который поддерживает массивы оболочки Bash. Это жизненно важная функция для написания чистых сценариев. Хорошо написанные сценарии используют разделение слов и исключений перечислены на разделении слов. Значительный список известных подводных камней (http://mywiki.woolledge.org/Bash) подпадает под эту категорию. <i>не читайте строки с for !</i> (http://mywiki.woolledge.org/Docs)

Синтаксис	Замена	Описани
<code>`COMMANDS`</code>	<code>\$(COMMANDS)</code>	<p>Это более старая форма за совместимая с Bourne. Оба <code>`COMMANDS`</code> и <code>\$(COMMANDS)</code> последний является <u>более</u> хотя первый, к сожалению, в распространен в скриптах. З новом стиле широко реализ современной оболочкой (а з другими). Единственная при обратных ссылок - это совме оболочкой Bourne (например подстановки обратных коман специальное экранирование примеры, найденные в дикой всего неправильно цитируют <i>Почему <code>\$(...)</code> предпочтитель (обратные ссылки)?</i> (http://mywiki.woolledge.org/Ba</p>
<code>[EXPRESSION]</code> и <code>test EXPRESSION</code>	<code>[[EXPRESSION]]</code>	<p><code>test</code> и <code>[</code> являются команд для оценки тестовых выраже идентичны и <code>[</code> несколько бо. распространены). Выражени обычных аргументов, в отлич Ksh/ <code>[[</code> Bash. Хотя проблема <code>((</code> , преимущества <code>[[</code> vs <code>[</code> потому что аргументы / расш объединяются в одно выраж классической <code>[</code> команде кол значительно. Если это вообщ используйте условное выраж тестовая команда") <code>[[</code> EXPR нет необходимости в совмес есть только несколько причин <code>[</code> . <code>[[</code> является одним из на переносимых и согласованн отличных от POSIX. Смотрит выражение и в чем <i>разница</i> (http://mywiki.woolledge.org/Ba</p>

Синтаксис	Замена	Описани
<pre>set -e , set -o errexit</pre> <p>и ERR ловушка</p>	<p>надлежащий поток управления и обработка ошибок</p>	<p>set -e приводит к тому, что ненулевые статусы выхода с фатальными. Это функция, предназначенная для исполнения во время разработки и не должна использоваться в производственном коде, особенно в скриптах инициализации и других сценариях. Не поддавайтесь соблазну об этом как об "обработке ошибок". Это просто способ найти место, где поместить обработку ошибок. Думайте об этом как о чем-то вроде strict на Perl или throws в JavaScript. Любимая, которая заставляет людей писать код. Многие руководства рекомендуют избегать его из-за кажущихся проблем, когда ненулевые статусы вызывают сбой работы скрипта. И наоборот, опытные программисты могут рекомендовать предписывать его использование. Поскольку он не предоставляет информации о местонахождении ошибки, он не сочетается с set -x или DEBGET функциями отладки Bash, и лучше устанавливать в самом скрипте.</p> <p>Большая часть этого также относится к тому, что хотя я видел, как он используется в оболочках, в которых pipefail или PIPESTATUS не POSIX, а set -e есть. factset -e одна функция Bash, которая относится к категории (в основном полезная). Эта set -e функция генерирует много вопросов и ложных сообщений в списке рассылки Bash, чем другие функции вместе взятые! Пожалуйста, не полагайтесь на set -e на логику. Вы по-прежнему отказываетесь от совета, убедитесь, что вы точно знаете, что это работает. Смотрите: <i>Почему errexit , или trap ERR не выдает ошибку?</i></p> <p>(http://mywiki.woledge.org/Bash:_errexit) http://www.fvue.nl/wiki/Bash:_errexit http://www.fvue.nl/wiki/Bash:_errexit</p>

Синтаксис	Замена	Описани
<code>set -u</code> или <code>set -o nounset</code>	Надлежащий поток управления и обработка ошибок	<code>set -u</code> вызывает попытки р неустановленные переменн как фатальные ошибки. Напр обходит поток управления и текущей среды оболочки. Ка статусы, неустановленные п являются обычной частью бс нетривиальных сценариев of <code>set -u</code> требует взломов, на для каждого расширения, ко отключено. Только самые со гарантируют, что расширени вызовет ошибку, если парам (http://austingroupbugs.net/view.php?id=104) (http://austingroupbugs.net/view.php?id=104) http://www.in-ulm.de/~mascheck/various/bourne_args/ (http://www.in-ulm.de/~mascheck/various/bourne_args/) различные/bourne_args/ (http://www.in-ulm.de/~mascheck/various/bourne_args/) ulm.de/~mascheck/various/bo видимому, некоторые счита отладки. Смотрите, как мне с <i>определена ли уже перемен</i> (http://mywiki.woolledge.org/BashFAQ/009) как правильно проверять опр переменные. Не используйте
<code>\${var?msg}</code> или <code>\${var:?msg}</code>	Надлежащий поток управления и обработка ошибок	Например <code>set -u</code> , это расш фатальную ошибку, которая завершает текущую среду об данный параметр не задан и выводит указанное сообщени от оператора. Если ожидае хотите создать утверждение лучше протестировать неопр переменные с помощью одн (http://mywiki.woolledge.org/BashFAQ/009) обработать ошибку вручную <code>die</code> функцию. Это расшире POSIX . Это лучше, чем <code>set</code> явно, но не намного. Это так случайно создавать забавно сообщения об ошибках: <pre>bash -c 'f() { definite "\${printf:?"\$1"}" - нет 1 }; f -v' bash: printf: -v - нет</pre>

В этой таблице перечислены функции, которые используются только в том случае, если у вас есть конкретная причина предпочесть его другой альтернативе. У них есть несколько законных применений, если вы знаете, что делаете, например, для тех, у кого особые требования к переносимости, или для того, чтобы использовать некоторые тонкие различия в поведении. Они часто (неправильно) используются без причины. Написание переносимых сценариев, которые выходят за рамки функций POSIX, требует знания того, как учитывать многие (часто недокументированные) различия во многих оболочках. Если вы случайно знаете, что делаете, не удивляйтесь, если кто-то скажет вам не использовать их.

Синтаксис	Замена	Описание
<code>function NAME { CMDS; }</code>	<code>NAME() COMPOUND-CMD</code>	<p>Это форма определения функции созданная для расширения формы Bourne и POSIX с измененным поведением и дополнительными функциями, такими как локальные переменные. Идея заключалась в чтобы функции нового стиля были аналогичны обычным встроенным функциям с их собственной средой области действия, в то время как функции в стиле POSIX больше полагались на специальные встроенные функции. <code>function</code> поддерживается почти оболочками, производными от <code>ksh</code> включая <code>Bash</code> и <code>Zsh</code>, но не задает POSIX. <code>Bash</code> обрабатывает все стили функций одинаково, но это необычно. <code>function</code> имеет некоторые предпочтительные характеристики многих вариантах <code>ksh</code>, что делает более переносимым для сценариев, которые используют расширения, отличные от POSIX, по некоторым показателям. Если вы собираетесь использовать <code>function</code> ключевое слово это означает, что вы либо ориентируетесь конкретно на <code>Ksh</code>, у вас есть подробные знания о том, как компенсировать различия между оболочками. Он всегда должен использоваться последовательно с <code>typeset</code>, но никогда не используется с <code>declare</code> или <code>local</code>. Также в <code>ks</code> фигурные скобки являются не группой команд, а обязательной частью синтаксиса (в отличие от <code>Bash</code> и <code>dk</code>). Смотрите Определения функций оболочки</p>

Синтаксис	Замена	Описание
<code>typeset</code>	<code>declare</code> , <code>local</code> , <code>export</code> , <code>readonly</code>	<p>Это тесно связано с вышеизложенным. Часто должно использоваться вместо <code>typeset</code> существует в первую очередь для <code>ksh</code> совместимости, но в <code>Bash</code> помечен как "устаревший" (хотя я совсем согласен с этим). Это имеет некоторый смысл, потому что будущая совместимость не может быть гарантирована, и любая совместимость вообще требует понимания отличий POSIX функций других оболочек и различий. Использование <code>declare</code> вместо <code>typeset</code> подчеркивает ваше намерение быть "только для" и определенно нарушает все остальные (кроме, возможно, <code>zsh</code>, если вам повезет). Проблема еще более усложняется из-за <code>Dash</code> и требований политики <code>Debian</code> (http://www.debian.org/doc/debian-policy/ch-files.html#s-scripts) к <code>local</code> встроенному компоненту, который сам по себе не полностью совместим с <code>Bash</code> и другими оболочками.</p>
<code>let 'EXPR'</code>	<code>((EXPR))</code> или <code>[\$((EXPR)) - ne 0]</code>	<p><code>let</code> является вариантом "простой команды" команды арифметической вычисления, которая принимает обычные аргументы. Оба <code>let</code> и <code>((expr))</code> присутствовали в <code>ksh88</code> что поддерживает одно, должно поддерживать другое. Ни один из них не является POSIX. Составной вариант предпочтительнее, потому что он принимает обычные аргументы без разделения слов и объединения, что делает его более безопасным и понятным. Кроме того, он обычно работает быстрее, особенно в <code>Bash</code> составные команды обычно выполняются значительно быстрее. Некоторые из (немногих) причин использования <code>let</code> подробно описаны на странице <code>let</code>. См. раздел Составная команда арифметической оценки</p>

Синтаксис	Замена	Описание
<code>eval</code>	Зависит. Часто код можно реструктурировать, чтобы использовать лучшие альтернативы.	<code>eval</code> приведен здесь для хорошей оценки, поскольку, к сожалению, и часто злоупотребляют, что любое использование <code>eval</code> (даже редкое) немедленно отвергается экспертами как неправильное, и с самых непосредственных решений которыми злоупотребляют новички. В самом деле, существуют правильные способы использования <code>eval</code> и для случаев, когда это необходимо, даже в таких сложных оболочках, как <code>Bash</code> и <code>Ksh</code> . <code>eval</code> необычен тем, что он подходит в более многофункциональных оболочках, чем в более минимальных оболочках, таких как <code>Dash</code> , где он используется для компенсации большого количества ограничений. Если вы обнаружите, что нуждаетесь в <code>eval</code> слишком часто, это может быть признаком того, что вам либо лучше использовать полностью другой язык, либо попытаться позаимствовать идеи из какой-либо другой парадигмы, которая плохо подходит для языка оболочки. Точно так же есть некоторые случаи, в которых слишком усердно работает, чтобы избежать <code>eval</code> , при этом к увеличению сложности и потере переносимости. Не заменяйте <code>eval</code> на что-то, что немного "слишком умно", просто чтобы избежать <code>eval</code> , все же примите разумные меры, чтобы избежать этого там, где это разумно. Встроенная команда <code>eval</code> и проблемы безопасности (http://mywiki.woolledge.org/BashFAQ)



Смотрите также

- Непереносимый синтаксис и использование команд
- Изменения в Bash
- BashFAQ 061 от Грегга: добавлен список основных функций (с тегом версии Bash) (<http://mywiki.woolledge.org/BashFAQ/061>)

- Руководство по переносимости Bash <-> POSIX с акцентом на Dash (<http://mywiki.woledge.org/Bashism>)
- <http://mywiki.woledge.org/BashPitfalls> (<http://mywiki.woledge.org/BashPitfalls>)



Обсуждение

 [scripting/obsolete.txt](#)  Последнее изменение: 2019/08/30 16:01 автор [ersen](#)

Этот сайт поддерживается Performing Databases - вашими экспертами по администрированию баз данных

Bash Hackers Wiki



Если не указано иное, содержимое этой вики лицензируется по следующей лицензии:
Лицензия GNU Free Documentation 1.3