Встроенная команда eval

Краткое описание

```
eval: eval [arg ...]
```

Описание

`eval` принимает ее аргументы, объединяет их, разделенные пробелами, и выполняет результирующую строку как код Bash в текущей среде выполнения. `eval` в Bash работает по существу так же, как и в большинстве других языков, которые имеют `eval` функцию. Возможно, проще всего подумать о `eval` том, что она работает так же, как запуск `bash -c "bash code…"` из скрипта, за исключением случая `eval`, когда данный код выполняется в текущей среде оболочки, а не в дочернем процессе.

Примеры

В этом примере буквальный текст в here-document выполняется как код Bash точно так, как если бы он должен был появиться в скрипте вместо `eval` команды под ним.

```
#!/usr/bin/env bash
{ myCode=$(
..
EOF

вычисление "$myCode"
```

Побочные эффекты расширения

Часто `eval` используется для создания побочных эффектов путем выполнения прохода расширения кода перед выполнением результирующей строки. Это позволяет делать то, что в противном случае было бы невозможно с обычным синтаксисом Bash. Это также, конечно, делает `eval` самую мощную команду во всем программировании оболочки (и в большинстве других языков, если на то пошло).

Этот код определяет набор идентичных функций, используя предоставленные имена. `eval` это единственный способ добиться этого эффекта.

```
main() {
local fun='() { echo "$FUNCNAME"; }' x

 для x в {f..n}; выполните
eval "${x} $ {fun}"
 Выполнено

 "$@"
}

главная "$ @"
```

Использование printf %q

Строка `printf %q` формата выполняет экранирование оболочки для своих аргументов. Это создает `printf %q` "анти-оценку" - при каждом прохождении строки через printf требуется, чтобы другая `eval` строка снова удаляла экранирование.

```
в то время как (( ++n <= 5 )) || ! evalBall="eval $evalBall"; выполни
ть
printf -v evalBall 'eval %q' "printf $n;${evalBall-printf '0\n'}"
выполнено
$evalBall
```

Приведенный выше пример в основном забавный и игровой, но иллюстрирует `printf %q` свойство.

Функции более высокого порядка

Поскольку во всех современных POSIX-совместимых оболочках отсутствует поддержка первоклассных функций (http://en.wikipedia.org/wiki/First-class_function), может быть заманчиво, а иногда и полезно смоделировать некоторые из их эффектов, используя `eval` для вычисления строки, содержащей код.

В этом примере показано частичное использование приложения (http://en.wikipedia.org/wiki/Partial_application) `eval` .

```
частичная функция {
 функция eval shift 2 \; "$1" \{ "$2" "$( printf'%q ' "${@:3}")"
'"$@"; }'
}

повтор функции {
 [[ $1 == +([0-9]) ]] || возвращает
набор типов n
while ((n++ < $1)); сделать
"${@:2}"
готово
}

частичная печать 3 повторите 3 printf '%s ' # Создайте новую функцию
с именем print3
print3 привет # Напечатать "привет" 3 раза
эхом
```

Это очень легко сделать неправильно и обычно не считается идиоматичным для Bash, если используется широко. Однако хорошим способом ее использования является абстрагирование eval от функций, которые проверяют их ввод и / или проясняют, какой ввод должен тщательно контролироваться вызывающим абонентом.

Соображения о переносимости

- К сожалению, поскольку eval является **специальным встроенным**, он получает свою собственную среду только в Bash и только тогда, когда Bash не находится в режиме POSIX. Во всех других оболочках плюс Bash в режиме POSIX среда eval будет просачиваться в окружающую среду. Можно обойти это ограничение, добавив префикс специальных встроенных функций к `command` обычным встроенным, но текущие версии ~~ksh93~~ и zsh не делают этого должным образом (исправлено (http://article.gmane.org/gmane.comp.programming.tools.ast.devel/686) в ksh 93v-2012-10-24 alpha). Более ранние версии zsh работают (с `setopt POSIX_BUILTINS` – похоже на регрессию). Это корректно работает в режиме Bash POSIX, Dash и mksh.
- `eval` это еще одна из немногих встроенных команд Bash с условным синтаксическим анализом аргументов, подобных ключевым словам, в форме составных присваиваний.

```
 $ ( eval a=( a b \\ c d ); printf '<%s> ' "${a[@]}"; echo ) # Работа
ет только в Bash.
<a> <b c> <d>
 Аргумент $ (x= a; eval "$ x"=( a b \\ c d ); printf '<%s> ' "${a
[@]}"; echo ) # больше не имеет формы допустимого присваивания, поэто
му применяются обычные правила синтаксического анализа.
-bash: синтаксическая ошибка рядом с неожиданным токеном `('
 $ (x = a; eval "$ x"'= ( a b \ c d )'; printf '<% s> ' "$ {a[@]}"; e
cho ) # Правильное цитирование тогда дает нам ожидаемые результаты.
<a> <b c> <d>
```

Мы не знаем, почему Bash это делает. Поскольку круглые скобки являются метасимволами, они должны обычно заключаться в кавычки или экранироваться при использовании в качестве аргументов. В первом примере выше та же ошибка, что и во втором, во всех оболочках, отличных от Bash, даже с составным назначением.

В случае `eval` не рекомендуется использовать это поведение, потому что, в отличие, например, от declare, начальное расширение по-прежнему подлежит всем расширениям, включая разделение слов и расширение пути.

```
 $ ( set -x; коснитесь 'x+=(\[[123]\]=*)' ' x+=([3]=yo)'; вычислить x
+=(*); echo "${x[@]}" )
+ коснитесь 'x+=(\[[123]\]=*)' ' x+=([3]=йо)'
+ eval 'x+=(\[[123]\]=*)' ' x+=([3]=йо)'
++ x+=(\[[123]\]=*)
++ x+=([3]=yo)
+ echo '[[123]]=*' yo
[[123]]=* yo
```

Другие команды, на которые, как известно, влияют составные аргументы присваивания, включают: let, declare, typeset, local, export и только для чтения. Другие странности ниже показывают как сходства, так и различия с такими командами, как declare . Правила для `eval` кажутся идентичными правилам let .

Смотрите также

- BashFAQ 48 - проблемы с оценкой и безопасностью (http://mywiki.wooledge.org/BashFAQ/048) – **ВАЖНО**
- Еще одна статья об оценке (http://fvue.nl/wiki/Bash:_Why_use_eval_with_variable_expansion%3F)
- Косвенное обращение через eval (http://mywiki.wooledge.org/BashFAQ/006#Assigning_indirect.2BAC8-reference_variables)
- Больше косвенности через eval (http://fvue.nl/wiki/Bash:_Passing_variables_by_reference)
- "Push" Мартина Вета (https://github.com/vaeth/push) – `printf %q` работает одинаково для POSIX.
- Взлом "волшебного псевдонима" (http://www.chiark.greenend.org.uk/~sgtatham/aliases.html)

Обсуждение

Крис Ф.А. Джонсон, 2012/06/03 22:13 ()

"просто не используйте eval" не обязательно; это просто неправильно!

Да, при использовании eval необходимо соблюдать осторожность, но в этом отношении она ничем не отличается от многих других команд.

Ян Шампера, 07.07.2012 11:46 ()

Я согласен. Это не должно быть проклято. Новички в оболочке должны быть действительно предупреждены. Если кто-то знает, что он делает, все в порядке

Дэн Дуглас, 26.07.2012 02:42 ()

Я тоже согласен (я написал страницу и эту заметку). eval не так плох, как некоторые его представляют, особенно на языке, самым фундаментальным принципом которого является: "возьмите строку, проанализируйте, оцените, повторно проанализируйте, снова оцените, повторяйте, пока в конечном итоге не выполните команду". Новичкам необходимы предупреждения. Требуется некоторый опыт, чтобы понять, где подходит eval. Бла-бла-бла, мы все это знаем.

Я бы также сказал, что чем более функциональную оболочку вы используете, тем реже целесообразно использовать eval.