

Замена команд

```
$( <КОМАНДЫ> )
```

```
` <КОМАНДЫ> `
```

Подстановка команд расширяется до вывода команд. Эти команды выполняются в подболочке, и их `stdout` данные - это то, до чего расширяется синтаксис подстановки.

Все **завершающие** новые строки удаляются (ниже приведен пример обходного пути).

На последующих этапах, **если они не указаны**, результаты подвергаются разделению на слова и расширению пути. Вы должны помнить об этом, потому что разделение слов также приведет к удалению встроенных новых строк и других `IFS` символов и разбиению результатов на несколько слов. Также вы, вероятно, получите неожиданные совпадения имен путей. **Если вам нужны буквенные результаты, процитируйте подстановку команд!**

Вторая форма ``COMMAND`` более или менее устарела для Bash, поскольку у нее есть некоторые проблемы с вложенностью ("внутренние" обратные знаки необходимо экранировать) и экранирующими символами. Используйте `$(COMMAND)`, это тоже POSIX!

Когда вы вызываете явную подболочку `(COMMAND)` внутри подстановки команд `$()`, тогда будьте осторожны, этот способ **неверен**:

```
$( ( КОМАНДА ) )
```

Почему? потому что это противоречит синтаксису арифметического расширения. Вам нужно отделить подстановку команд от внутренней `(COMMAND)` :

```
$( ( КОМАНДА ) )
```

Особенности

Когда внутренняя команда - это только перенаправление ввода, и ничего больше, например

```
$(  
# или  
`
```

затем Bash пытается прочитать данный файл и действовать только в том случае, если данная команда была `cat FILE`.

Более пристальный взгляд на две формы

В общем, вы действительно должны использовать только форму `$()`, она нейтральна к экранированию, она может быть вложена, она также POSIX. Но взгляните на следующие фрагменты кода, чтобы самостоятельно решить, какая форма вам нужна в конкретных обстоятельствах:

Вложенность

Форма обратной вставки ``...`` не может быть вложена напрямую. Вам придется избегать "внутренних" обратных ссылок. Кроме того, чем глубже вы продвигаетесь, тем больше `escape`-символов вам нужно. Некрасиво.


```
echo `echo`ls` # НЕПРАВИЛЬНОЕ
echo `echo ``ls`` # ПРАВИЛЬНОЕ
echo $(echo $ (ls)) # ИСПРАВИТЬ
```

Синтаксический анализ


Все основано на том факте, что форма обратной кавычки представляет собой простую замену символов, в то время как каждая `$()` конструкция открывает собственный последующий шаг синтаксического анализа. Все внутри `$()` интерпретируется так, как если бы оно было написано обычным образом в командной строке. Никакого специального экранирования **ничего** не требуется:

```
echo "$(echo "$(ls))" # вложенные двойные кавычки - без проблем
```

Конструкции, которых следует избегать

Это не все блестяще `$()`, по крайней мере, для моего текущего Bash (3.1.17(1) - release).  **Update:** исправлено, поскольку 3.2-beta вместе с неправильным толкованием `)` распознается как арифметическое расширение [redduck666]. Похоже, что эта команда неправильно закрывает шаг замены, и `echo` печатает `"ls"` и `"")`:

```
echo $(
# некоторые комментарии заканчиваются на )
ls
)
```

Кажется, что каждое закрытие `"")` сбивает с толку эту конструкцию. Также (очень необычная ) конструкция, подобная:

```
echo $(читается как VAR; регистр "$ var" в foo) бла ;; esac) # выдает
некоторую ошибку, когда видит ";;"
```

исправляет это:

```
echo $ (читать VAR; регистр "$ var" в (foo) blah ;; esac) # будет раб
отать, но просто оставьте это, пожалуйста ;-)
```

Заключение:

В общем, \$() предпочтительным методом должен быть метод:

- это чистый синтаксис
- это интуитивно понятный синтаксис
- это более читабельно
- это можно вложить
- его внутренний синтаксический анализ является отдельным

Примеры

Чтобы получить дату:

```
ДАТА="$(дата)"
```

Чтобы скопировать файл и получить с выводом ошибки:

```
COPY_OUTPUT="$(cp file.txt /некоторые/где 2>&1)"
```

Внимание: здесь вам нужно перенаправить `cp STDERR` на `STDOUT` цель, потому что подстановка команд только улавливает `STDOUT` !

Перехватить стандартный вывод и сохранить конечные новые строки:

```
var=$(echo -n $'\n'); echo -n "$var"; # $var == ""
var=$(echo -n $ '\n'; echo -n x); var="{var%x}"; echo -n "$var" # $v
ar == "\ n"
```

Это добавляет "x" к выводу, что предотвращает удаление завершающих строк вывода предыдущих команд с помощью \$().

Удалив этот "x" позже, мы остаемся с выводом предыдущих команд с завершающими символами новой строки.

Смотрите также

- Внутренний: введение в расширение и замену
- Внутренний: устаревший и устаревший синтаксис

Обсуждение

пол, [2010/06/01 11:51 \(\)](#), [2010/06/02 04:21 \(\)](#)

Отличная статья, интересно, можете ли вы чем-нибудь помочь...

Я выполняю пентест и нашел обход каталога (и, таким образом, восстановил источник неисправного скрипта), но что бы я ни пытался, я не могу выйти из подстановки команд и получить командную инъекцию... Это должно быть возможно....

Вот ошибочные строки скрипта;

```
#!/bin/sh
/bin/echo "<h2>Журнал аудита для " $QUERY_STRING "</h2>"
/bin/cat /mnt/video0/LOGS/`echo $QUERY_STRING | sed "s/-//g"` .ext
```

... обход каталога выполняется путем простой отправки файла на чтение, за которым следует пробел и любой символ (поэтому .ext добавляет и создает два вызова команды cat, второй из которых завершается ошибкой).

... есть идеи?

Спасибо, Пабб

Тони (<http://wiki.bash-hackers.org/syntax/expansion/cmdsubst>), [2010/09/09 16:10 \(\)](#)

Здравствуйте

Обычно я вижу заключительную кавычку, соответствующую ведущей кавычке, например, "за которой следует". Это также верно и для обратной ссылки.

Это оставалось верным, пока я не просмотрел некоторую устаревшую литературу по UNIX, где за обратным знаком следовала одинарная кавычка.

Теперь, если я правильно прочитал ваш <раздел специальностей>, это отклоняется, когда внутренняя команда используется только для перенаправления ввода. Является ли это сокращением синтаксиса или есть какие-либо побочные эффекты при использовании исходного режима сбалансированной пары кавычек с обратными метками, которые я не смог подобрать?

например, `

Ян Шампера, [2010/09/09 16:49 \(\)](#)

Мне очень жаль, это была опечатка.

Обозначения, которые вы нашли в старой литературе, являются очень распространенными текстовыми обозначениями (ничего технического). Может быть, им нравится использовать его, потому что при чтении его нельзя смешивать с (техническими) кавычками оболочки.

Например, смотрите справочную страницу `bash(1)`:

Расширения фигурных скобок могут быть вложенными. Результаты каждой развернутой строки не сортируются; сохраняется порядок с лева направо. Например, `a{d,c,b}` расширяется до ``ade ace ab e'`.

Грегг, [2011/12/31 09:01 \(\)](#)

Почему это не работает?

```
echo $(sed 's/local/$HOME\\/' /etc/hosts)
```

Я бы ожидал, что 'localhost' станет '/ home / user / host /', но он становится '\$ HOME / host'

Спасибо

Ян Шампера, [2011/12/31 16:51 \(\)](#)

Привет,

из-за цитирования. Используйте обычные двойные кавычки, чтобы заменить `Bash $HOME`, также используйте другой разделитель для `sed` (здесь это хэш-метка):

```
sed "s#local#$HOME/#" /etc/hosts
```

Ник Накс, [2012/03/18 06:04 \(\)](#), [2012/03/18 14:03 \(\)](#)

Вы правы, что `)` сбивает с толку синтаксис подстановки команды `$()`.

Например, это работает

```
s=`for((i=0;i<3;i++)); выполнить регистр $x в a) echo -n A ;; *)
echo -n B ;; esac; готово
```

пока это НЕ работает:

```
s= $( для((i=0;i<3;i++)); выполнить случай $ x в a) echo -n A;;
*) echo -n B;; esac; сделано)
```

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