Цитаты и экранирование

Цитирование и экранирование важны, поскольку они влияют на то, как Bash реагирует на ваш ввод. Существует три признанных типа:

- **Экранирование для каждого символа** с использованием обратной косой черты: `\$stuff`
- **слабое цитирование** с двойными кавычками: `"stuff"`
- **сильное цитирование** с помощью одинарных кавычек: `'stuff'`

Все три формы имеют одну и ту же цель: **они дают вам общий контроль над синтаксическим анализом, расширением и результатами расширения.**

Помимо этих основных вариантов, существуют некоторые специальные методы кавычек (например, интерпретация экранирования ANSI-C в строке), с которыми вы познакомитесь ниже.

ВНИМАНИЕ Символы кавычек (`"` , двойные кавычки и `'` , одинарные кавычки) являются синтаксическим элементом, влияющим на синтаксический анализ. Это не связано с символами кавычек, передаваемыми в виде текста в командной строке! Синтаксические кавычки удаляются перед вызовом команды! Пример:

```
### НЕТ, НЕТ, НЕТ: это передает три строки:
### (1) "мой
### (2) многословие
### (3) аргумент"
MYARG="\"мой многословный аргумент \""
somecommand $MYARG

### ЭТО НЕ (!) ТО ЖЕ САМОЕ, ЧТО ###
команда "мой многословный аргумент"

### ВАМ НУЖНО ###
MYARG="мой многословный аргумент"
команда "$MYARG"
```

Экранирование для каждого символа

Экранирование для каждого символа полезно при расширениях и заменах. В общем, символ, который имеет особое значение для Bash, например, знак доллара (`$`), может быть замаскирован, чтобы не иметь особого значения, используя обратную косую черту:

```
для echo \$HOME установлено значение \"$HOME\"
```

- `\$HOME` не будет расширяться, потому что его больше нет в синтаксисе расширения переменных
- Обратная косая черта превращает кавычки в литералы - в противном случае Bash интерпретировал бы их

Последовательность `\<newline>` (обратная косая черта без кавычек, за которой следует `<newline>` символ) интерпретируется как **продолжение строки**. Он удаляется из входного потока и, таким образом, фактически игнорируется. Используйте это, чтобы украсить свой код:

```
# escapestr_sed()
# чтение потока из стандартного ввода и экранирующих символов в текст
e, которые могут быть интерпретированы как
# специальные символы с помощью sed
escape_sed() {
sed \
 -e 's/\//\\\//g' \
 -e 's/\&/\\\&/g'
}
```

Обратная косая черта может использоваться для маскировки каждого символа, который имеет особое значение для bash. Исключение: внутри строки, заключенной в одинарные кавычки (см. Ниже).

Слабое цитирование

Внутри строки со слабыми кавычками **нет специальной интерпретации**:

- пробелы как разделители слов (при разделении начальной командной строки и при разделении слов!)
- одинарные кавычки для введения сильных кавычек (см. Ниже)
- символы для сопоставления с образцом
- расширение тильды
- расширение имени пути
- замена процесса

Все остальное, особенно расширение параметров, выполняется!

```
ls -l "*"
```

Не будут расширены. `ls` получает литерал `*` в качестве аргумента. Если у вас нет имени файла `*` , он выдаст ошибку.

```
echo "Ваш ПУТЬ: $PATH"
```

Будет работать так, как ожидалось. `$PATH` расширен, потому что он имеет двойные (слабые) кавычки.

Если возникает обратная косая черта в двойных кавычках ("слабое цитирование"), есть 2 способа справиться с этим

- если за baskslash следует символ, который имел бы особое значение даже внутри двойных кавычек, обратная косая черта удаляется, а следующий символ

теряет свое особое значение
- если за обратной косой чертой следует символ без особого значения, обратная косая черта не удаляется

В частности, это означает, что `"\$"` станет `$` , но `"\x"` станет `\x` .

Сильное цитирование

Сильное цитирование очень легко объяснить:

Внутри строки, заключенной в одинарные кавычки, **ничего** не интерпретируется, кроме одинарной кавычки, которая закрывает строку.

```
echo 'Ваш ПУТЬ: $PATH'
```

`$PATH` не будет расширен, он интерпретируется как обычный текст, потому что он окружен сильными кавычками.

На практике это означает, что для создания текста, например `Here's my test…` , в виде строки, заключенной в одинарные кавычки, вам нужно оставить и повторно ввести одинарную кавычку, чтобы получить символ `" ' "` в виде буквального текста:

```
# НЕПРАВИЛЬНОЕ
эхо "Вот мой тест ..."

# ПРАВИЛЬНОЕ
эхо 'Вот мой тест ...'

# АЛЬТЕРНАТИВА: Также возможно смешивать и сопоставлять кавычки для у
добства чтения:
эхо "Вот мой тест"
```

ANSI C как строки

Bash предоставляет еще один механизм цитирования: строки, содержащие ANSI C-подобные escape-последовательности. Синтаксис:

```
$'строка'
```

где следующие escape-последовательности декодируются в `string` :

Код	Значение
`\"`	двойные кавычки
`\'`	одинарная кавычка
`\\`	обратная косая черта
`\a`	символ предупреждения терминала (колокольчик)
`\b`	backspace

Код	Значение
`\e`	escape (ASCII () 033)
`\E`	escape (ASCII () 033) **\ E является нестандартным**
`\f`	форма подачи
`\n`	новая строка
`\r`	возврат каретки
`\t`	горизонтальная вкладка
`\v`	вертикальная вкладка
`\cx`	символ элемента управления-x, например, `$'\cZ'` для печати управляющей последовательности, состоящей из Ctrl-Z (`^Z`)
`\uXXXX`	Интерпретируется `XXXX` как шестнадцатеричное число и выводит соответствующий символ из набора символов (4 цифры) (Bash 4.2-alpha)
`\UXXXXXXXX`	Интерпретируется `XXXX` как шестнадцатеричное число и выводит соответствующий символ из набора символов (8 цифр) (Bash 4.2-alpha)
`\nnn`	восьмибитный символ, значением которого является восьмеричное значение nnn (от одной до трех цифр)
`\xHH`	восьмибитный символ, значением которого является шестнадцатеричное значение HH (одна или две шестнадцатеричные цифры)

Это особенно полезно, когда вы хотите передать специальные символы в качестве аргументов некоторым программам, например, передать перевод строки в sed .

Результирующий текст обрабатывается так, как если бы он был заключен в **одинарные кавычки**. Дальнейшего расширения не происходит.

`$'...'` Синтаксис взят из ksh93, но переносим на большинство современных оболочек, включая pdksh. Спецификация (http://austingroupbugs.net/view.php?id=249#c590) для него была принята для выпуска SUS 7. Все еще есть некоторые отставшие, такие как большинство вариантов ash, включая dash, (за исключением busybox, построенного с функциями "совместимости с bash").

I18N/L10N

Например, знак доллара, за которым следует строка в двойных кавычках

```
echo $"создание базы данных ..."
```

означает I18N. Если для этой строки доступен перевод, он используется вместо заданного текста. Если нет, или если языковой `C` стандарт равен / `POSIX` , знак доллара просто игнорируется, что приводит к обычной строке с двойными кавычками.

Если строка была заменена (переведена), результат заключен в двойные кавычки.

На случай, если вы программист на C: Цель `$"…"` та же, что и для `gettext()` или `_()`.

Полезные примеры локализации ваших скриптов см. в приложении I к руководству по написанию расширенных сценариев Bash (http://tldp.org/LDP/abs/html/localization.html).

Внимание: существует дыра в безопасности. Пожалуйста, ознакомьтесь с документацией gettext (http://www.gnu.org/software/gettext/manual/html_node/bash.html)

Распространенные ошибки

Списки строк в циклах for

Классический цикл for использует список слов для перебора. Список также может быть в переменной:

```
mylist="СОБАКА КОШКА ПТИЦА ЛОШАДЬ"
```

НЕПРАВИЛЬНЫЙ способ перебора этого списка:

```
для animal в "$ mylist"; сделать
echo $animal
сделано
```

Почему? Из-за двойных кавычек технически расширение `$mylist` рассматривается как **одно слово**. Цикл for повторяется ровно один раз, при `animal` этом устанавливается значение для всего списка.

ПРАВИЛЬНЫЙ способ перебора этого списка:

```
для animal в $mylist; сделать
echo $animal
готово
```

Разработка тестовой команды

Команда `test` или `[ … ]` (классическая тестовая команда) является обычной командой, поэтому применяются обычные синтаксические правила. Давайте возьмем сравнение строк в качестве примера:

```
[ СЛОВО = СЛОВО]
```

`]` В конце - это удобство; если вы введете `which [`, вы увидите, что на самом деле существует двоичный файл с таким именем. Итак, если бы мы писали это как тестовую команду, это было бы:

```
тестовое СЛОВО = СЛОВО
```

Когда вы сравниваете переменные, разумно заключать их в кавычки. Давайте создадим тестовую строку с пробелами:

```
mystring="моя строка"
```

А теперь проверьте эту строку на соответствие слову "testword":

```
[ $mystring = testword ] # НЕПРАВИЛЬНО!
```

Это не удается! Это слишком много аргументов для теста сравнения строк. После выполнения расширения вы действительно выполняете:

```
[ моя строка = тестовое слово ]
проверьте мою строку = testword
```

Что неверно, потому `my` что и `string` являются двумя отдельными аргументами.

Итак, что вы действительно хотите сделать, это:

```
[ "$mystring" = testword ] # ПРАВИЛЬНО!
```

```
тест 'моя строка' = тестовое слово
```

Теперь команда имеет три параметра, что имеет смысл для двоичного (с двумя аргументами) оператора.

Подсказка: внутри условного выражения (`[[ ]]`) Bash не выполняет разделение слов, и поэтому вам не нужно заключать ссылки на переменные в кавычки - они всегда рассматриваются как "одно слово".

Смотрите также

- Внутренний: несколько слов о словах ...
- Внутренний: разделение слов
- Внутренний: введение в расширения и замены
- Внешнее: Grymore: Shell Quoting (http://www.grymoire.com/Unix/Quote.html)

Обсуждение

Szilvi, 2012/09/02 19:56 (), 2012/09/03 09:19 ()

Спасибо за публикацию этой замечательной статьи! Это действительно помогло мне более четко увидеть эти волшебные вещи bash. Но у меня все еще есть проблема, которую я не могу решить. Я опубликую это здесь, может быть, кто-нибудь сможет мне помочь.

Итак, вот оно: у меня есть тестовая папка с несколькими вложенными папками:

```
# найти .
.
./etc
./etc/a
./a
./a/b
./e
./c
./c /d
```

Я хочу перечислить все папки и файлы, кроме папки etc и ее содержимого, я буду использовать эту команду, и я получу именно то, что хочу:

```
# найти . ! -целое имя "./etc*"
.
./a
./a/b
./e
./c
./c /d
```

До этого момента все в порядке. Но внутри скрипта bash МНЕ ЭТО НУЖНО КАК СТРОКА, потому что я создаю условие на основе некоторых внутренних значений. Теперь посмотрите на это:

```
# cond='! -полное имя "./etc*"'
```

И когда я снова запускаю поиск…

```
# найти . $ cond
.
./etc
./etc/a
./a
./a/b
./e
./c
./c /d
```

... в нем перечислены папка etc и ее содержимое, которых не должно быть.

Я уверен, что это проблема с цитатами, и я перепробовал все известные мне варианты, но не смог решить проблему. Где ошибка?

Я ценю вашу помощь, Силви

Ян Шампера, 2012/09/03 09:27 ()

Да, это проблема с цитированием.

Текст, который вы пишете в переменной (`"./etc"`), на самом деле является текстом. Кавычки, которые вы указываете в командной строке (`find . ! -wholename "./etc*"`), - это **синтаксис**. Вы не можете "хранить синтаксис в

переменных". Синтаксис (цитирование) используется, чтобы сообщить Bash, что такое слово, когда оно не может его автоматически обнаружить (и особенно здесь, чтобы не заставлять Bash расширять сам подстановочный знак, а передавать его в виде текста `find`).

В общем, вы должны создать массив, в котором каждый элемент содержит одно "слово", а весь массив формирует аргументы, которые вы хотите передать `find` :

```
cond=(! -wholename "./etc*")
# используйте его
для поиска... "${cond[@]}" ...
```

В любом случае, наиболее правильным решением было бы использовать `-prune` тест / действие из `find` . Пожалуйста, посмотрите статью find в вики Грега (http://mywiki.wooledge.org/UsingFind)

Аарон, 2012/10/20 00:01 (), 2012/10/20 10:59 ()

Из командной строки это работает

```
mailx -s "Журналы обращений GM $ HOST для заявки" aaron.brandt@XX
XXX.com
```

Но изнутри скрипта это не так, как же так?

Не работает.. У меня проблема со старой версией mail, так что это единственное решение

```
# для ваших записей.
 printf "\n\ E[1;33mSend ваше устранение неполадок в тикете?\033
[m [y / N] "
 БИЛЕТ для ЧТЕНИЯ,
если [ "$ TICKET" == "y" ];
затем
mailx -s "Билет #: $ TKT" smc@XXXX.com

Билет: $ TKT " $USR@XXXX.com
```

Ян Шампера, 2012/10/20 11:01 ()

Что это `\r.\r` там? Просто дайте mailx текстовый файл в качестве входных данных и протестируйте.

Аарон, 2012/10/20 15:37 ()

Это возврат средств, если я использую mailx -s "Ticket #: $ TKT" smc@XXXX.com

Из приглашения bash \r.\r работает, но если я использую его в скрипте, это не так. Это как если бы \r.\r даже не было.

Если я использую mail -s, он не включает строку темы, только mailx -s.

Ян Шампера, 2012/10/20 16:49 ()

Лучшим способом для вас было бы написать письмо (в формате mail, с заголовками и т. Д.) И доставить его с вашим MTA `sendmail -oi` (или эквивалентом).

Все остальное - слишком много догадок.

sshaw (http://github.com/sshaw), 2013/12/31 03:14 ()

```
set -x
```

Это хороший способ для отладки и / или лучшего понимания цитирования bash:

```
~ > dirs='/etc /tmp'
~ >(set -x; ls "$dirs" > /dev/null)
+ ls '/etc /tmp'
ls: /etc /tmp: нет такого файла или каталога
~ >(set -x; ls $dirs > /dev/null)
+ ls /etc /tmp
~ > dirs="/etc/*"
~ >(set -x; ls "$dirs" > /dev/null)
+ ls '/etc/*'
ls: /etc/*: нет такого файла или каталога
~ >(set -x; ls $dirs > /dev/null)
+ ls /etc/RemoteManagement.launchd /etc/afpovertcp.cfg /etc/alias
es
# ...
```

syntax/quoting.txt Последнее редактирование: 2019/10/30 17:39 автор ersen

Bash Hackers Wiki