Условное выражение

Краткое содержание

```
[[ <ВЫРАЖЕНИЕ> ]]
```

Описание

Условное выражение подразумевается как современный вариант классической тестовой команды. Поскольку это **не** обычная команда, Bash не нужно применять обычные правила синтаксического анализа командной строки, такие как распознавание `&&` как оператор списка команд.

Функции тестирования в основном те же (см. Списки для классической команды тестирования), с некоторыми дополнениями и расширениями.

Оператор	Описание
(`<EXPRESSION>` **)**	Используется для группировки выражений, чтобы влиять на приоритет операторов
`<EXPRESSION1>` **&&** `<EXPRESSION2>`	`TRUE` если `<EXPRESSION1>` **и** `<EXPRESSION2>` являются `TRUE` (**не** используйте `-a` !)
`<EXPRESSION1>` **||** `<EXPRESSION2>`	`TRUE` если `<EXPRESSION1>` **или** `<EXPRESSION2>` `TRUE` (**не** используйте `-o` !)
`<STRING>` **==** `<PATTERN>`	`<STRING>` проверяется на соответствие шаблону `<PATTERN>` - `TRUE` при совпадении *Но примечание1, цитирование шаблона приводит к буквальному сравнению.*
`<STRING>` **=** `<PATTERN>`	эквивалентно `==` оператору
`<STRING>` **!=** `<PATTERN>`	`<STRING>` проверяется на соответствие шаблону `<PATTERN>` - `TRUE` **нет совпадения**

Оператор	Описание
`<STRING> =~ <ERE>`	`<STRING>` проверяется на соответствие расширенному регулярному выражению (https://en.wikipedia.org/wiki/Regular_expression#POSIX_extended) `<ERE>` - `TRUE` на совпадение
См . Классические операторы тестирования	**Не** используйте `test` -типичные операторы `-a` and `-o` для AND и OR .
См. Также арифметические сравнения	Используя `(( <EXPRESSION> ))` составную команду арифметического выражения

При `== !=` использовании операторов and строка справа от оператора считается шаблоном и сопоставляется в соответствии с правилами сопоставления с образцом. Если опция оболочки `nocasematch` включена, сопоставление выполняется без учета регистра буквенных символов.

1. Любая часть шаблона может быть заключена в кавычки, чтобы заставить его сопоставляться как литеральная строка.

Когда используются операторы `<` и `>` (порядок сортировки строк), тест выполняется с использованием текущей локали, когда `compat` уровень превышает "40".

Приоритет оператора (самый высокий ⇒ самый низкий):

- `( <EXPRESSION> )`
- `! <EXPRESSION>`
- `<EXPRESSION1> && <EXPRESSION2>`
- `<EXPRESSION1> || <EXPRESSION2>`

Не используйте `test` -типичные операторы `-a` и `-o` для И и ИЛИ, они не известны условному выражению. Вместо этого используйте операторы `&&` и `||` .

Разделение слов

Разделение слов и расширение пути не выполняются в приведенном вами выражении. Это означает, что переменная, содержащая пробелы, может использоваться без кавычек:

```
предложение ="Будьте либеральны в том, что вы принимаете, и консерват
ивны в том, что вы отправляете"
checkme="Будьте либеральны в том, что вы принимаете, и консервативны
в том, что вы отправляете"
если [[ $ sentence == $checkme ]]; затем
повторите "Соответствует ...!"
else
echo "Извините, нет совпадения :-("
fi
```

Сравните это с классической тестовой командой, где выполняется разделение слов (потому что это обычная команда, а не что-то особенное):

```
предложение ="Будьте либеральны в том, что вы принимаете, и консерват
ивны в том, что вы отправляете"
checkme="Будьте либеральны в том, что вы принимаете, и консервативны
в том, что вы отправляете"
если [ "$ sentence" == "$checkme" ]; затем
повторите "Соответствует ...!"
else
echo "Извините, нет совпадения :-("
fi
```

Вам нужно заключить в кавычки эту ссылку на переменную в классической тестовой команде, поскольку (из-за пробелов) разделение слова в противном случае нарушит его!

Сопоставление регулярных выражений

Используя оператор =~ , левый операнд сопоставляется с **расширенным регулярным выражением (ERE)** в правой части.

Это согласуется с сопоставлением с шаблонами: каждая заключенная в кавычки часть регулярного выражения воспринимается буквально, даже если она содержит специальные символы регулярного выражения.

Наилучшая практика - поместить регулярное выражение для сопоставления в переменную. Это делается для того, чтобы избежать ошибок синтаксического анализа оболочки в действительных регулярных выражениях.

```
Регулярное ВЫРАЖЕНИЕ="^[[:верхний:]]{2}[[:нижний:]]*$"

# Тест 1
СТРОКА=Привет
, если [[ $STRING =~ $REGEX ]]; затем
повторите "Совпадение".
else
повторяет "Нет совпадения".
фи
# ==> "Нет совпадения".

# Тест 2
СТРОКА=Привет
, если [[ $STRING =~ $REGEX ]]; затем
повторите "Совпадение".
else
повторяет "Нет совпадения".
фи
# ==> "Совпадение".
```

На интерпретацию специальных символов регулярного выражения в кавычках можно повлиять, установив параметры оболочки `compat31` и `compat32` (`compat*` в общем случае). Смотрите Список опций оболочки.

Специальная переменная массива BASH_REMATCH

Переменная массива, члены которой назначаются `=~` двоичным оператором `[[` условной команде.

Элемент с индексом 0 - это часть строки, соответствующая всему регулярному выражению. Элемент с индексом n - это часть строки, соответствующая n-му подвыражению, заключенному в скобки.

См. BASH_REMATCH .

Пример:

```
если [[ "Быстрый, рыжий лис" =~ ^, To\ (.*),\ (.*)\ fox$ ]]; затем
повторите "${BASH_REMATCH[0]} - это $ {BASH_REMATCH[1]} и $ {BASH_REM
ATCH[2]}".;
фи

==> Быстрая, рыжая лиса - быстрая и рыжая.
```

Различия в поведении по сравнению со встроенной тестовой командой

Начиная с версии Bash 4.1 alpha, тестовые праймериз '<' и '>' (лексикографическое сравнение строк) используют текущие настройки локали, в то время как те же примитивы для встроенной тестовой команды этого не делают. Это приводит к следующей ситуации, когда они ведут себя по-разному:

```
$ ./cond.sh
[[ ' 4' < '1' ]] --> выход 1
[[ 'шаг +' <'шаг-' ]] --> выход 1
[ ' 4' \< '1' ] --> выход 0
[ 'шаг +' \< 'шаг-'] --> выход 0
```

Оно не будет выровнено. Условное выражение продолжает учитывать locate, как введено в 4.1-alpha, встроенная `test [` команда / продолжает вести себя по-другому.

Неявный арифметический контекст

Когда вы используете числовое сравнение, аргументы вычисляются как арифметическое выражение. Арифметическое выражение должно быть заключено в кавычки, если оно содержит пробелы и не является результатом расширения.

```
[[ 'i=5, i+=2' -уравнение 3+4 ]] && echo true # выводит true.
```

Примеры

Соображения о переносимости

- `[[ ... ]]` функциональность не определяется POSIX (R), хотя это зарезервированное слово
- Среди основных "языков надмножеств POSIX-оболочки" (за неимением лучшего термина), которые есть `[[` , составная команда test expression является одной из самых переносимых функций, отличных от POSIX. Помимо `=~` оператора, почти все основные функции совместимы между Ksh88, Ksh93, mksh, Zsh и Bash. Ksh93 также добавляет большое количество уникальных функций сопоставления с образцом, не поддерживаемых другими оболочками, включая поддержку нескольких разных диалектов регулярных выражений, которые вызываются с использованием синтаксиса, отличного от синтаксиса Bash `=~` , хотя `=~` по-прежнему поддерживается ksh и по умолчанию используется ERE.
- Как расширение POSIX ERE, большинство программ GNU поддерживают обратные ссылки в ERE, включая Bash. Согласно POSIX, только BRE должен их поддерживать. Для этого требуется, чтобы Bash был связан с glibc, поэтому он не обязательно будет работать на всех платформах. Например,

```
$(m='(abc(def))(\1)(\2)'; [[ abcdefabcdefdef =~ $m ]]; printf
'<%s> ' $? "${BASH_REMATCH[@]}" )
```

  даст `<0> <abcdefabcdefdef> <abcdef> <def> <abcdef> <def>` .
- оператор `=~` (regex) был введен в Bash 3.0, и его поведение изменилось в Bash 3.2: начиная с 3.2, строки и подстроки, заключенные в кавычки, по умолчанию сопоставляются как литералы.
- поведение операторов `<` and `>` (порядок сопоставления строк) изменилось с версии Bash 4.0

Смотрите также

- Внутренний: язык сопоставления с образцом
- Внутренняя: классическая тестовая команда
- Внутреннее: предложение if
- В чем разница между test, [и [[? (http://mywiki.wooledge.org/BashFAQ/031)- BashFAQ 31 - вики Грега.

1

Bash Hackers Wiki