**Глава 7. Проверка условий**

практически любой язык программирования включает в себя условные операторы, предназначенные для проверки условий, чтобы выбрать тот или иной путь развития событий в зависимости от этих условий. В Bash, для проверки условий, имеется команда **test**, различного вида скобочные операторы и условный оператор **if/then**.

**7.1. Конструкции проверки условий**

* Оператор **if/then** проверяет -- является ли [код завершения](https://www.opennet.ru/docs/RUS/bash_scripting_guide/c2105.html#EXITSTATUSREF) списка команд 0 (поскольку 0 означает "успех"), и если это так, то выполняет одну, или более, команд, следующие за словом then.
* Существует специальная команда -- **[** ([левая квадратная скобка](https://www.opennet.ru/docs/RUS/bash_scripting_guide/c301.html#LEFTBRACKET)). Она является синонимом команды **test**, и является [встроенной](https://www.opennet.ru/docs/RUS/bash_scripting_guide/c5358.html#BUILTINREF) командой (т.е. более эффективной, в смысле производительности). Эта команда воспринимает свои аргументы как выражение сравнения или как файловую проверку и возвращает код завершения в соответствии с результатами проверки (0 -- истина, 1 -- ложь).
* Начиная с версии 2.02, Bash предоставляет в распоряжение программиста конструкцию [[[ ... ]]](https://www.opennet.ru/docs/RUS/bash_scripting_guide/c2171.html#DBLBRACKETS) *расширенный вариант команды test*, которая выполняет сравнение способом более знакомым программистам, пишущим на других языках программирования. Обратите внимание: **[[** -- это [зарезервированное слово](https://www.opennet.ru/docs/RUS/bash_scripting_guide/c5358.html#KEYWORDREF), а не команда.

Bash исполняет **[[ $a -lt $b ]]** как один элемент, который имеет код возврата.

Круглые скобки [(( ... ))](https://www.opennet.ru/docs/RUS/bash_scripting_guide/x4862.html) и предложение [let ...](https://www.opennet.ru/docs/RUS/bash_scripting_guide/c5358.html#LETREF) так же возвращают код 0, если результатом арифметического выражения является ненулевое значение. Таким образом, [арифметические выражения](https://www.opennet.ru/docs/RUS/bash_scripting_guide/c11565.html#ARITHEXPREF) могут учавствовать в операциях сравнения.

Предложение let "1<2" возвращает 0 (так как результат сравнения "1<2" -- "1", или "истина")

(( 0 && 1 )) возвращает 1 (так как результат операции "0 && 1" -- "0", или "ложь")

* Условный оператор **if** проверяет код завершения любой команды, а не только результат выражения, заключенного в квадратные скобки.
* if cmp a b &> /dev/null # Подавление вывода.
* then echo "Файлы a и b идентичны."
* else echo "Файлы a и b имеют различия."
* fi
* if grep -q Bash file
* then echo "Файл содержит, как минимум, одно слово Bash."
* fi
* if COMMAND\_WHOSE\_EXIT\_STATUS\_IS\_0\_UNLESS\_ERROR\_OCCURRED
* then echo "Команда выполнена успешно."
* else echo "Обнаружена ошибка при выполнении команды."
* fi
* Оператор **if/then** допускает наличие вложенных проверок.
* if echo "Следующий \*if\* находится внутри первого \*if\*."
* if [[ $comparison = "integer" ]]
* then (( a < b ))
* else
* [[ $a < $b ]]
* fi
* then
* echo '$a меньше $b'
* fi

*Это детальное описание конструкции "if-test" любезно предоставлено Stephane Chazelas.*

**Пример 7-1. Что есть "истина"?**

#!/bin/bash

echo

echo "Проверяется \"0\""

if [ 0 ] # ноль

then

echo "0 -- это истина."

else

echo "0 -- это ложь."

fi # 0 -- это истина.

echo

echo "Проверяется \"1\""

if [ 1 ] # единица

then

echo "1 -- это истина."

else

echo "1 -- это ложь."

fi # 1 -- это истина.

echo

echo "Testing \"-1\""

if [ -1 ] # минус один

then

echo "-1 -- это истина."

else

echo "-1 -- это ложь."

fi # -1 -- это истина.

echo

echo "Проверяется \"NULL\""

if [ ] # NULL (пустое условие)

then

echo "NULL -- это истина."

else

echo "NULL -- это ложь."

fi # NULL -- это ложь.

echo

echo "Проверяется \"xyz\""

if [ xyz ] # строка

then

echo "Случайная строка -- это истина."

else

echo "Случайная строка -- это ложь."

fi # Случайная строка -- это истина.

echo

echo "Проверяется \"\$xyz\""

if [ $xyz ] # Проверка, если $xyz это null, но...

# только для неинициализированных переменных.

then

echo "Неинициализированная переменная -- это истина."

else

echo "Неинициализированная переменная -- это ложь."

fi # Неинициализированная переменная -- это ложь.

echo

echo "Проверяется \"-n \$xyz\""

if [ -n "$xyz" ] # Более корректный вариант.

then

echo "Неинициализированная переменная -- это истина."

else

echo "Неинициализированная переменная -- это ложь."

fi # Неинициализированная переменная -- это ложь.

echo

xyz= # Инициализирована пустым значением.

echo "Проверяется \"-n \$xyz\""

if [ -n "$xyz" ]

then

echo "Пустая переменная -- это истина."

else

echo "Пустая переменная -- это ложь."

fi # Пустая переменная -- это ложь.

echo

# Кргда "ложь" истинна?

echo "Проверяется \"false\""

if [ "false" ] # это обычная строка "false".

then

echo "\"false\" -- это истина." #+ и она истинна.

else

echo "\"false\" -- это ложь."

fi # "false" -- это истина.

echo

echo "Проверяется \"\$false\"" # Опять неинициализированная переменная.

if [ "$false" ]

then

echo "\"\$false\" -- это истина."

else

echo "\"\$false\" -- это ложь."

fi # "$false" -- это ложь.

# Теперь мв получили ожидаемый результат.

echo

exit 0

**Упражнение.** Объясните результаты, полученные в [Пример 7-1](https://www.opennet.ru/docs/RUS/bash_scripting_guide/c2171.html#EX10).

if [ condition-true ]

then

command 1

command 2

...

else

# Необязательная ветка (можно опустить, если в ней нет необходимости).

# Дополнительный блок кода,

# исполняемый в случае, когда результат проверки -- "ложь".

command 3

command 4

...

fi

|  |  |
| --- | --- |
| Note | Когда *if* и *then* располагаются в одной строке, то конструкция *if* должна завершаться точкой с запятой. И *if*, и *then* -- это [зарезервированные слова](https://www.opennet.ru/docs/RUS/bash_scripting_guide/c5358.html#KEYWORDREF). Зарезервированные слова начинают инструкцию, которая должна быть завершена прежде, чем в той же строке появится новая инструкция.  if [ -x "$filename" ]; then |

**Else if и elif**

elif

**elif** -- это краткая форма записи конструкции else if. Применяется для построения многоярусных инструкций if/then.

if [ condition1 ]

then

command1

command2

command3

elif [ condition2 ]

# То же самое, что и else if

then

command4

command5

else

default-command

fi

Конструкция **if test condition-true** является точным эквивалентом конструкции **if [ condition-true ]**, где левая квадратная скобка **[** выполняет те же действия, что и команда **test**. Закрывающая правая квадратная скобка **]** не является абсолютно необходимой, однако, более новые версии Bash требуют ее наличие.

|  |  |
| --- | --- |
| Note | Команда **test** -- это [встроенная](https://www.opennet.ru/docs/RUS/bash_scripting_guide/c5358.html#BUILTINREF) команда Bash, которая выполняет проверки файлов и производит сравнение строк. Таким образом, в Bash-скриптах, команда **test** *не* вызывает внешнюю (/usr/bin/test) утилиту, которая является частью пакета *sh-utils*. Аналогично, [ не производит вызов утилиты /usr/bin/[, которая является символической ссылкой на /usr/bin/test.  bash$ **type test**  test is a shell builtin  bash$ **type '['**  [ is a shell builtin  bash$ **type '[['**  [[ is a shell keyword  bash$ **type ']]'**  ]] is a shell keyword  bash$ **type ']'**  bash: type: ]: not found |

**Пример 7-2. Эквиваленты команды test -- /usr/bin/test, [ ], и /usr/bin/[**

#!/bin/bash

echo

if test -z "$1"

then

echo "Аргументы командной строки отсутствуют."

else

echo "Первый аргумент командной строки: $1."

fi

echo

if /usr/bin/test -z "$1" # Дает тот же рузультат, что и встроенная команда "test".

then

echo "Аргументы командной строки отсутствуют."

else

echo "Первый аргумент командной строки: $1."

fi

echo

if [ -z "$1" ] # Функционально идентично вышеприведенному блоку кода.

# if [ -z "$1" эта конструкция должна работать, но...

#+ Bash выдает сообщение об отсутствующей закрывающей скобке.

then

echo "Аргументы командной строки отсутствуют."

else

echo "Первый аргумент командной строки: $1."

fi

echo

if /usr/bin/[ -z "$1" # Функционально идентично вышеприведенному блоку кода.

# if /usr/bin/[ -z "$1" ] # Работает, но выдает сообщение об ошибке.

then

echo "Аргументы командной строки отсутствуют."

else

echo "Первый аргумент командной строки: $1."

fi

echo

exit 0

Конструкция [[ ]] более универсальна, по сравнению с [ ]. Этот *расширенный вариант команды test* перекочевал в Bash из *ksh88*.

|  |  |
| --- | --- |
| Note | Внутри этой конструкции не производится никакой дополнительной интерпретации имен файлов и не производится разбиение аргументов на отдельные слова, но допускается подстановка параметров и команд. |

file=/etc/passwd

if [[ -e $file ]]

then

echo "Файл паролей найден."

fi

|  |  |  |
| --- | --- | --- |
| Tip | | Конструкция **[[ ... ]]** более предпочтительна, нежели **[ ... ]**, поскольку поможет избежать некоторых логических ошибок. Например, операторы &&, ||, < и > внутри [[ ]] вполне допустимы, в то время как внутри [ ] порождают сообщения об ошибках. |
| Note | Строго говоря, после оператора **if**, ни команда **test**, ни квадратные скобки ( [ ] или [[ ]] ) не являются обязательными.  dir=/home/bozo  if cd "$dir" 2>/dev/null; then # "2>/dev/null" подавление вывода сообщений об ошибках.  echo "Переход в каталог $dir выполнен."  else  echo "Невозможно перейти в каталог $dir."  fi  Инструкция "if COMMAND" возвращает код возврата команды COMMAND.  Точно так же, условие, находящееся внутри квадратных скобок может быть проверено без использования оператора **if**.  var1=20  var2=22  [ "$var1" -ne "$var2" ] && echo "$var1 не равно $var2"  home=/home/bozo  [ -d "$home" ] || echo "каталог $home не найден." | |

Внутри [(( ))](https://www.opennet.ru/docs/RUS/bash_scripting_guide/x4862.html) производится вычисление арифметического выражения. Если результатом вычислений является ноль, то возвращается 1, или "ложь". Ненулевой результат дает код возврата 0, или "истина". То есть полная противоположность инструкциям **test** и [ ], обсуждавшимся выше.

**Пример 7-3. Арифметические выражения внутри (( ))**

#!/bin/bash

# Проверка арифметических выражений.

# Инструкция (( ... )) вычисляет арифметические выражения.

# Код возврата противоположен коду возврата инструкции [ ... ] !

(( 0 ))

echo "Код возврата \"(( 0 ))\": $?." # 1

(( 1 ))

echo "Код возврата \"(( 1 ))\": $?." # 0

(( 5 > 4 )) # true

echo "Код возврата \"(( 5 > 4 ))\": $?." # 0

(( 5 > 9 )) # false

echo "Код возврата \"(( 5 > 9 ))\": $?." # 1

(( 5 - 5 )) # 0

echo "Код возврата \"(( 5 - 5 ))\": $?." # 1

(( 5 / 4 )) # Деление, все в порядке

echo "Код возврата \"(( 5 / 4 ))\": $?." # 0

(( 1 / 2 )) # Результат деления < 1.

echo "Код возврата \"(( 1 / 2 ))\": $?." # Округляется до 0.

# 1

(( 1 / 0 )) 2>/dev/null # Деление на 0.

echo "Код возврата \"(( 1 / 0 ))\": $?." # 1

# Для чего нужна инструкция "2>/dev/null" ?

# Что произойдет, если ее убрать?

# Попробуйте убрать ее и выполнить сценарий.

exit 0