

chmod [ugoa] [+-=] [rwx] файл(ы)

Где:  
  
[ugoa] указывает, к кому применяются изменения:  
u: владелец файла.  
g: группа файла.  
o: остальные пользователи (не владелец и не входящие в группу).  
a: все (при использовании заменяет собой ugo).  
  
[+-=] определяет, что нужно сделать с разрешениями:  
+: добавить разрешение.  
-: удалить разрешение.  
=: установить разрешение точно (заменить текущие разрешения).  
  
— [rwx] представляет собой комбинацию символов `r`, `w` и `x`, обозначающих разрешения:  
r: разрешение на чтение.  
w: разрешение на запись.  
x: разрешение на выполнение.  
  
Примеры использования символьной нотации:  
  
chmod u+x файл — добавить право выполнения для владельца файла.  
chmod go-rw файл — удалить право чтения и записи для группы и остальных пользователей.  
chmod a=rwx файл — установить полные права (чтение, запись и выполнение) для всех.

Интерпретаторы

#!/bin/bash - интерпретатор BASH  
  
#!/usr/bin/python - интерпретатор Python  
  
#!/usr/bin/python3 - интерпретатор Python3

**Команда *echo***. Помимо перечня стандартных команд интерпретатора BASH, самое простое, что можно выполнить в скрипте – это вывод текстовых комментариев пользователю с помощью команды **echo**. Например, данную команду можно использовать для приглашения пользователя выполнить какое-нибудь действие или использовать для приветствия. В следующем примере с помощью **echo** отображается приветствие.

***$ echo "Добрый вечер! "***

**Команда *read*.** При создании скриптов вы можете также применять переменные и сценарии, написанные для других интерпретаторов и команд. Команда **read** служит командой запроса пользователя на ввод информации. Она сочетается с подсказкой пользователю, которая приглашает ввести нужную информацию. Приведем пример использования **read** :

***$ echo "Введите ваше имя: "***

***$ read your\_name***

***$ echo "Добрый день, " $your\_name***

**Использование переменных.** Как и в других языках программирования, в shell Вы можете использовать переменные, а присваивать значения переменным можно через оператор присваивания равно «=».

Сначала вводится имя переменной, потом **без пробела !!!** знак «=», затем без пробела значение переменной. Имя переменной может состоять из любого количества буквенных символов, включая символ подчеркивания. Имя может содержать и цифры, однако имя не должно начинаться с цифры. Все остальные символы (в частности, восклицательный знак, амперсанд и пробел) в имя входить не должны. Такие символы зарезервированы интерпретатором для специальных целей. Как следствие, имя может состоять только из одного слова, поскольку при синтаксическом анализе команд интерпретатор рассматривает пробел как разделитель имен команд и аргументов. Значение переменной может состоять из любой последовательности символов. В следующем примере присвоим переменной «ppp» значение «www123yyy».

***ppp="www123yyy"***

*Название переменной и знак «=» должны быть обязательно записаны слитно.*

На значение переменной можно ссылаться посредством ее имени, которое предваряется оператором $ :

***echo $ppp***

Теперь пример с приглашением можно переписать с помощью переменных:

***echo "Введите ваше имя: "***

***read your\_name***

***greet="Добрый день"***

***echo $greet $your\_name***

**Аргументы командной строки.** В сценарии, как и в командах, можно использовать аргументы.

Аргумент в сценарии обозначается оператором $ с указанием номера позиции в командной строке. Нумерация параметров начинается с единицы, а заканчивается девятью.

Аргумент $0 резервируется для имени shell-сценария, в качестве которого выступает первое слово, находящееся в командной строке.

Первый параметр задается переменной $1, второй — $2 и т.д. По умолчанию, имеется возможность устанавливать 9 переменных от $1 до $9 . Если вводится несколько аргументов, можно отдельно обращаться к каждому из них по его номеру.

Иногда требуется определить в сценарии точное количество введенных аргументов, эту задачу можно выполнить с помощью встроенного аргумента $# .

Например:

***./script.sh arg1 aaaa***

Если в скрипте прописать команды:

**echo $1 echo $2 echo $#**

то будут выведены значения «arg1» , «aaaa» и общее численное значение (количество) введенных аргументов.

**Команда *export.*** Иногда для разных файлов-сценария требуется воспользоваться определенной переменной, которая уже была определена. Переменные, которые определяются в интерпретаторе shell, являются локальными для него. В некотором смысле такая переменная принадлежит своему интерпретатору. Непосредственно определить переменную для другого интерпретатора нельзя, однако можно экспортировать определение переменной из одного интерпретатора в другой (например, дочерний) с помощью команды **export**.

**export** — это одна из встроенных команд интерпретатора BASH.

***aaa=123456***

***export aaa***

**Арифметические операции – команда *let*.** Команда let — это команда интерпретатора BASH shell, обеспечивающая выполнение операций над арифметическими величинами. С помощью этой команды можно сравнивать числовые значения или выполнять над ними арифметические операции, такие как сложение или умножение. Формат команды: ***let значение1 оператор значение2***. Приведем пример:

***let umn=2\*5***

***echo $umn 10***

Произведите сложение двух величин для тренировки работе с командой ***let***.

**Арифметические операции – команда *expr*.** Команда **expr** используется для проверки целочисленных значений, но может применяться также и при обработке строк. Формат команды: ***expr значение1 оператор значение2***.

Приведем пример:

***expr 10 + 10***

*Все аргументы команды* ***expr*** *должны быть записаны через пробел.* При работе с операциями умножения или деления знаки арифметических операций должны быть экранированы:

***expr 10 \\* 10***

либо

***expr 10 "\*" 10***

При необходимости записать значение арифметической операции в переменную воспользуйтесь следующим форматом команды:

***umn=`expr 10 \\* 10`***

***echo $umn***

Примеры работы команды **expr** со строками в описании лабораторной работы не рассматриваются и предлагаются для самостоятельного изучения.

**Управляющие конструкции.** Управляющие конструкции предназначены для управления ходом выполнения команд shell-сценария. Эти конструкции позволяют организовать повторное выполнение определенной последовательности команд или выбирать для выполнения команды, необходимые в конкретной ситуации. Управляющая конструкция состоит из двух основных компонентов: ***операции проверки*** и ***команд***. В результате выполнения сравнения (проверки условия) возвращается значение «истина» или «ложь», а затем на основании полученного результата выполняются определенные заданные команды. Существует два вида управляющих конструкций: *циклические (циклы)* и *условные (условия).* Циклическая конструкция используется для повторного выполнения команд, тогда как условная — для выполнения последовательности *команд,* которая удовлетворяет определенным условиям. В интерпретаторе BASH shell можно использовать три циклические конструкции ( **while**, **for** и **for-in** ) и две условные — **if** и **case**.

Управляющие конструкции **while** и **if** — это конструкции общего назначения, которые обычно используются при решении таких задач, как итерационные вычисления и проверка различных условий. Управляющие конструкции **case** и **for** ориентированы на более узкий круг задач. Конструкция **case** является многовариантным оператором и представляет собой частный случай условного оператора **if**. Эта конструкция часто используется при создании меню. Конструкция **for** представляет собой цикл, однократно обрабатывающий всю информацию для каждого значения, включенного в список, до тех пор, пока не встретится окончание списка.

Кроме сравнения значений или переменных, управляющие конструкции **if** и **while** можно применять для проверки того, успешно или неудачно была выполнена системная команда Linux. Напомним, что в Linux каждая выполняемая команда возвращает код завершения. Если выполнение команды было успешным, ее код

завершения равен 0. Если по какой-либо причине команда не была выполнена успешно, кодом завершения будет некоторое положительное значение, указывающее тип ошибки. Управляющие конструкции **if** и **while** позволяют проверить, чему был равен код завершения: 0 или некоторому другому значению. Если код завершения равен нулю, значит, выполнение команды было успешным и управляющая конструкция **if** или **while** будет завершена.

**Условные конструкции: *if, if-else, elif, case*.** Интерпретатор BASH shell включает несколько условных управляющих конструкций (табл. 3), которые позволяют выбирать для выполнения определенные команды Linux. Многие из этих конструкций напоминают условные управляющие конструкции в языках программирования, но имеются и некоторые отличия.

|  |  |
| --- | --- |
| **Таблица 1**  **Управляющие конструкции интерпретаторов bash Shell** | **Функция** |
| **Условные управляющие конструкции** | **if, case** и др. |
| **if** команда **then** команда **fi** | Конструкция **if** вызывает выполнение  действия в случае, если результат проверки истинен |
| **if** команда **then** команда **else**  команда **fi** | Конструкция **if-else** вызывает выполнение действия в  случае, если код завершения проверяемой команды paвен значению  «истина», в противном случае выполняется действие **else** |
| **if** команда **then** команда **elif** команда  **then** команда **else** команда **fi** | Конструкция **elif** дает возможность вкладывать конструкции **if**, что позволяет выбрать один из многих вариантов; если истинно условие, проверяемое первой конструкцией **if**, выполняются предусмотренные в ней команды и следующей конструкции **elif**  управление не передается |
| **case** строка **in** шаблон) команда**;; еsас** | Конструкция **case** сравнивает строковое значение с одним из нескольких шаблонов (образцов). При обнаружении совпадения выполняются команды, соответствующие этому  шаблону |
| команда && команда | Логическая операция И возвращает значение 0 («истина»), если обе команды возвращают значение 0; если же одна из команд возвращает ненулевое значение, результат операции И равен «ложь» и данная  операция возвращает ненулевое |

|  |  |
| --- | --- |
|  | значение |
| команда **| |** команда | Логическая операция ИЛИ, возвращающая значение 0 («истина») в случае, если одна или обе команды возвращают значение 0 («истина»); если обе команды возвращают ненулевое значение, то результат  операции ИЛИ — «ложь» и операция возвращает ненулевое значение |
| **!** команда | Логическая операция НЕ, инвертирует  код завершения команды |
| **Циклические управляющие конструкции** | **while, until, for, for-in, select** |
| **while** команда **do** команды **done** | Конструкция **while** выполняет действие до тех пор, пока команда проверки  возвращает значение «истина» |
| **until** команда **do** команды **done** | Конструкция **until** выполняет действие до тех пор, пока команда проверки  возвращает значение «ложь» |
| **for** переменная **in** список-значений **do**  команды **done** | Конструкция **for-in** предназначена для обработки списка значений. Переменной последовательно  присваиваются значения из списка |
| **for** переменная **do** команды **done** | Конструкция **for** предназначена для последовательной обработки аргументов сценария. Переменной  последовательно присваивается значение каждого аргумента |
| **select** строка **in** перечень-элементов **do**  команды **done** | Конструкция **select** создает меню на основе элементов заданного списка, а затем выполняется указанная команда  (обычно это команда **case**) |

**Условная конструкция *if-then*** . Условная конструкция if ставит условие на выполнение команды. Этим условием является код завершения какой-то конкретной команды Linux. Если команда выполнена успешно (то есть код завершения равен 0), то команды внутри конструкции if выполняются. Если код завершения отличен от 0, то команды внутри конструкции if выполняться не будут. Иногда требуется выбрать один из двух вариантов, в зависимости от того, как была выполнены команда. Ключевое слово else конструкции if позволяет выбрать один из двух вариантов.

***if [условие] then команды … else команды … fi***

**Конструкция *case*.** Конструкция case применяется для выбора одного из нескольких вариантов конструкции, при этом выбор осуществляется путем сравнения заданного в конструкции значения с несколькими возможными шаблонами. Каждое возможное значение проверяемой переменной связывается с набором операций. Если происходит совпадение значений переменных, то выполняются соответствующие данному значению операции. Конструкция case начинается ключевым словом **case**, за которым следует имя сравниваемой переменной и ключевое слово **in**. Затем перечисляется набор шаблонов. Каждый шаблон представляет собой регулярное выражение, заканчивающееся закрывающейся круглой скобкой. После закрывающейся круглой скобки следуют команды, связанные с этим шаблоном. Список команд требуется завершить двумя символами «**;;**» в отдельной строке, указывающими на завершение команд. Полную конструкцию завершает ключевое слово **esac**. Синтаксис конструкции следующий:

***саsе строка in***

***шаблон) команды;;***

***шаблон) команды;;***

***шаблон) команды;;***

***\*) команды, заданные по умолчанию;;***

***esac***

**Цикл *while*.** Циклическая команда while используется для повторения команд. Цикл начинается словом while, за которым идет команда, в следующей строке используется ключевое слово do, по окончанию цикла используется слово done. Далее покажем синтаксис команды while:

***while [условие] do***

***команды … done***

*либо*

***while [условие] ; do команды …***

***done***

**Конструкция *for-in*.** Конструкция for-in предназначена для последовательного обращения к значениям, перечисленным в списке. Данная связка включает два операнда — переменную и список значений. Каждое значение из списка поочередно присваивается переменной конструкции for-in. Конструкция является циклом и каждой переменной присваивается новое значение из списка. Полный цикл оканчивается при достижении конца списка. Основа цикла for-in начинается ключевым словом do и завершается ключевым словом done. Покажем синтаксис цикла for-in.

***for переменная in [перечень значений] do***

***команды … done***

**Цикл *for*.** Конструкция for удобна при использовании аргументов командной строки при вызове файла shell в качестве списка значений for. Все аргументы из этого списка автоматически присваиваются переменной, используемой в команде for. В первом цикле переменной присваивается значение первого аргумента, во втором цикле — значение второго аргумента, в третьем – значение третьего аргумента и т. д.

***for переменная do***

***команды … done***

|  |  |  |
| --- | --- | --- |
| **Таблица 2 Управляющие конструкции**  **интерпретаторов bash Shell** | **Примеры синтаксиса** | **Примеры синтаксиса** |
| **if** <условие1>  **then**  <команды> [**elif** <условие2> **then**  <команды> [**else**  <команды>]]  **fi** | **if** [ $# -eq 0 ] **then** #comment echo "Zero:" **fi** | **if** [ $# -eq 1 ] **then** #comment echo "Zero:"  **elif** [ "5" -gt "1" ]  **then** echo "5" **fi** |
| **for** var [**in** <список>]  **do**  <команды>  **done** | **for** i **in `***seq* 10**` do**  *echo* $i  **done** | **for** file **in** \*  **do**  *echo* $file  **done** |
| **for**((<Cи итератор>))  **do**  <команды>  **done** | **for** ((i=0;i<10;i++))  **do**  *echo* $i  **done** | **for** (( j=10; j > 0; j--))  **do**  *echo* -n $j' '  **done** |
| **while** <условие>  **do**  <команды>  **done** | i=6  **echo** $i  **while** ((i<10))  **do**  *echo* $((i++))  **done**  **echo** $i | i=10  **echo** $i  **while** [ $i **-le** 10 ]  **do**  *let* i+=1;  **done**  **echo** $i |

|  |  |  |
| --- | --- | --- |
| **until** <условие>  **do**  <команды>  **done** |  | dividend=10 divisor=2 remainder=1  until [ "$remainder" -eq 0 ] do  let "remainder = $dividend  % $divisor" dividend=$divisor divisor=$remainder done  echo "divident=$dividend" |
| **select** var [**in** <list>]  **do**  <команды> [**break**] **done** | **select** vegetable in "бобы" "морковь" "картофель" "лук" "брюква"  **do** echo "" **break done** |  |
| **case** var in  ) ;;  ) ;;  ) ;;  \*)  **esac** | **case** "$last\_sym" in "1") units="a"  ;;  "2" | "3" | "4" ) units="b"  ;;  \* ) units="c"  ;;  **esac** |  |
| **function** имя {  <тело функции>  } | **function** f { echo ""  } |  |
|  |  |  |
|  |  | let i=i+1 i=$((i+1))  ((i=i+1)) |

Список литературы:

1) Ермак В. В., Козлов А. В., Савченко В.Ю. ОС LINUX для разработчиков и пользователей ПО САПР БИС: учеб. пособие. – М. МИЭТ, 2010. – 248с.: ил.

Блок А\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* export PS1='\t j\j \u@\h:\w\n\$ '

Обычно в переменной PS1 с помощью специальных последовательностей символов определяют формат приглашения для ввода команд. Подробный список этих последовательностей можно почитать в документации к bash, в данном примере:

\t — вывод «текущего времени», на самом деле получается время завершения выполнения предыдущей команды.

j\j — выводит символ j и после него количество запущенных job, т.е. процессов в фоне.

\u@\h — имя пользователя и название сервера. Если работаете с несколькими серверами через удаленные терминалы — чтобы не путаться.

\w — после двоеточия — рабочая директория.

\n — поскольку строка получилась хоть и информативной, но длинной, то приглашаем вводить команды с новой строки, а эта верхняя строка будет наглядно отделять от результата работы предыдущей команды.

$ — на новой строке будет выводиться символ, либо $ для обычного пользователя, либо # для root'а и, отделив его пробелом, можно приглашать вводить новую команду.

export LANG=en\_US

export USER\_DIR=/home/user9 export PATH=${USER\_DIR}:$PATH

echo 'puts "Tclsh version: [info patchlevel]";exit 0' | tclsh export EDITOR=/usr/bin/gedit

alias l='ls -la'

Блок Б\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Чтение строк из файла на примере вложенных блоков

#!/bin/bash

# Чтение строк из файла /etc/fstab. File=/etc/fstab

{

read line1 read line2

} < $File

echo "Первая строка в $File :" echo "$line1"

echo

echo "Вторая строка в $File :" echo "$line2"

exit 0

Вложенные блоки кода, заключенные в {фигурные скобки} исполняются в пределах того же процесса, что и сам скрипт (т.е. не вызывают запуск дочернего процесса).

Посимвольное ASCII-сравнение

#!/bin/bash

veg1=carrots veg2=tomatoes

if [[ "$veg1" < "$veg2" ]]

then echo "Несмотря на то, что слово $veg1 предшествует слову $veg2," echo "это никак не отражает мои кулинарные предпочтения."

else echo "Интересно. Каким словарем вы пользуетесь?" fi

Конвейер

Передает вывод предыдущей команды на ввод следующей или на вход командного интерпретатора shell. Этот метод часто используется для

связывания последовательности команд в единую цепочку. Конвейер исполняется в дочернем процессе, а посему -- не имеет доступа к переменным сценария. Если одна из команд в конвейере завершается аварийно, то это приводит к аварийному завершению работы всего конвейера.

#!/bin/bash echo ls -l | sh

# Передает вывод "echo ls -l" командному интерпретатору shell, #+ тот же результат дает простая команда "ls -l".

cat \*.lst | sort | uniq

# Объединяет все файлы ".lst", сортирует содержимое и удаляет повторяющиеся строки.

Операции сравнения

1. Cравнение целых чисел

|  |  |  |
| --- | --- | --- |
| -eq | равно | if [ "$a" -eq "$b" ] |
| -ne | не равно | if [ "$a" -ne "$b" ] |
| -gt | больше | if [ "$a" -gt "$b" ] |
| -ge | больше или равно | if [ "$a" -ge "$b" ] |
| -lt | меньше | if [ "$a" -lt "$b" ] |
| -le | меньше или равно | if [ "$a" -le "$b" ] |
| < | меньше (внутри двойных круглых скобок) | (("$a" < "$b")) |
| <= | меньше или равно (внутри двойных круглых  скобок) | (("$a" <= "$b")) |
| > | больше (внутри двойных круглых скобок) | (("$a" > "$b")) |
| >= | больше или равно (внутри двойных круглых  скобок) | (("$a" >= "$b")) |

1. Cравнение строк

|  |  |  |
| --- | --- | --- |
| = | равно | if [ "$a" = "$b" ] |
| == | равно(синоним оператора =) | if [ "$a" == "$b" ] |
| != | не равно | if [ "$a" != "$b" ] |
| < | меньше, в смысле величины ASCII-кодов Обратите внимание!  Символ "<" необходимо экранировать внутри [ ] | if [[ "$a" < "$b" ]]  if [ "$a" \< "$b" ] |
| > | больше, в смысле величины ASCII-кодов Обратите внимание!  Символ "<" необходимо экранировать внутри [ ] | if [[ "$a" > "$b" ]]  if [ "$a" \> "$b" ] |
| -z | строка "пустая", т.е. имеет нулевую длину |  |
| -n | строка не "пустая" |  |