os

Bash

Useful guide

Оглавление

Справка по команде

Для получения **подробного** справочного руководства по любой команде можно набрать в консоли *man название_команды*, для **краткой справки** – *название_команды --help*.
Примеры: *man man* – справочное руководство по команде **man**; *man bash* – справочное руководство по интерпретатору **bash**.

Shell скрипты

Shell-скрипт – это обычный текстовый файл, в который последовательно записаны команды, которые пользователь может обычно вводить в командной строке. Файл выполняется командным интерпретатором – шеллом (**shell**).

В Linux- и Unix-системах для того, чтобы бинарный файл или скрипт смогли быть запущены на выполнение, для пользователя, который запускает файл, должны быть установлены соответствующие права на выполнение. Это можно сделать с помощью команды *chmod u+x имя_скрипта*.

В первой строке скрипта указывается путь к интерпретатору с помощью конструкции *#! /bin/bash*.

Для создания скрипта можно воспользоваться текстовым редактором **nano** или **vi**, набрав имя редактора в командной строке.

Ниже приводятся основные правила программирования на языке bash.

Комментарии

Строки, начинающиеся с символа **#** (за исключением комбинации **#!**), являются комментариями. Комментарии могут также располагаться и в конце строки с исполняемым кодом.

Особенности работы со строками

Одиночные кавычки (' '), ограничивающие подстроки с обеих сторон, служат для предотвращения интерпретации специальных символов, которые могут находиться в строке.

Двойные кавычки (" ") предотвращают интерпретацию специальных символов, за исключением **$**, ` (обратная кавычка) и **\ (escape** – обратный слэш).

Желательно использовать двойные кавычки при обращении к переменным.

При необходимости вывести специальный символ можно также использовать экранирование: символ **** предотвращает интерпретацию следующего за ним символа.

Переменные

Имя переменной аналогично традиционному представлению об идентификаторе, т.е. именем может быть последовательность **букв**, **цифр** и **подчеркиваний**, начинающаяся с **буквы** или **подчеркивания**.

Когда интерпретатор встречает в тексте сценария имя переменной, то он вместо него подставляет значение этой переменной. Поэтому ссылки на переменные называются подстановкой переменных.

Если *variable1* – это имя переменной, то *$variable1* – это ссылка на ее значение.

"Чистые" имена переменных, без префикса **$**, могут использоваться только при объявлении переменной или при присваивании переменной некоторого значения.

В отличие от большинства других языков программирования, **Bash не производит разделения переменных по типам**. По сути, переменные Bash являются **строковыми переменными**, но,

в зависимости от контекста, Bash допускает целочисленную арифметику с переменными.

Оператор присваивания

При использовании оператора присваивания **нельзя** ставить пробелы слева и справа от знака равенства.

Если в процессе присваивания требуется выполнить арифметические операции, то перед записью арифметического выражения используют оператор **let**, например:

```
Let a=2\*2
```

(оператор умножения является специальным символом и должен быть экранирован).

Арифметические операторы:

- `+` — сложение
- `-` — вычитание
- `*` — умножение
- `/` — деление (целочисленное)
- `**` — возведение в степень
- `%` — остаток от деления

Специальные переменные.

Для Bash существует ряд зарезервированных имен переменных, которые хранят определенные значения.

Позиционные параметры

Аргументы, передаваемые скрипту из командной строки, хранятся в зарезервированных переменных `$0, $1, $2, $3...`, где `$0` – это название файла сценария, `$1` – это первый аргумент, `$2` – второй, `$3` – третий и так далее.

Аргументы, следующие за *$9*, должны заключаться в фигурные скобки, например: *${10}, ${11}, ${12}*.

Передача параметров скрипту происходит в виде перечисления этих параметров после имени скрипта через пробел в момент его запуска.

Другие зарезервированные переменные

- *$DIRSTACK* — содержимое вершины стека каталогов
- *$UID* — идентификатор пользователя.
- *$HOME* — домашний каталог пользователя
- *$HOSTNAME* — hostname компьютера
- *$HOSTTYPE* — архитектура машины.
- *$PWD* — рабочий каталог
- *$OSTYPE* — тип ОС
- *$PATH* — путь поиска программ
- *$PPID* — идентификатор родительского процесса
- *$SECONDS* — время работы скрипта (в секундах)
- *$#* — общее количество параметров, переданных скрипту
- *$** — все аргументы, переданные скрипту (выводятся в строку)
- *$@* — то же самое, что и предыдущий, но параметры выводятся в столбик
- *$!* — PID последнего запущенного в фоне процесса
- *$$* — PID самого скрипта

Код завершения

Команда **exit** может использоваться для завершения работы сценария, точно так же как и в программах на языке C. Кроме того, она может возвращать некоторое значение, которое может быть проанализировано вызывающим процессом.

Команде **exit** можно явно указать код возврата, в виде *exit nnn*, где **nnn** – это код возврата (число в диапазоне 0– 255).

Оператор вывода

echo переменные_или_строки

Оператор ввода

read имя_переменной

Одна команда **read** может прочитать (присвоить) значения сразу для нескольких переменных.

Если переменных в **read** больше, чем их введено (через пробелы), оставшимся присваивается пустая строка.

Если передаваемых значений больше, чем переменных в команде **read**, то лишние игнорируются.

Условный оператор.

```
If команда;
then команда;
[else команда];
fi
```

Если команда вернула после выполнения значение "истина", то выполняется команда после **then**. Если есть необходимость сравнивать значения переменных и/или констант, после **if** используется специальная команда *[[выражение]]*.

Обязательно ставить пробелы между выражением и скобками, например:

```
if [[ "$a" -eq "$b" ]]
then echo "a = b"
fi
```

Операции сравнения:

Для строк

- `-z` — строка пуста
- `-n` — строка не пуста
- `= (==)` — строки равны
- `!=` — строки не равны
- `<` — меньше
- `>` — больше

Для числовых значений

- `-eq` — равно
- `-ne` — не равно
- `-lt` — меньше
- `-le` — меньше или равно
- `-gt` — больше
- `-ge` — больше или равно

Для логических выражений

- `!` — отрицание логического выражения
- `-a (&&)` — логическое «И»
- `-o (||)` — логическое «ИЛИ»

Множественный выбор.

Для множественного выбора может применяться оператор **case**.

```
case переменная in
значение_1 )
команда 1
;;
значение_2 )
команда 2
;;
esac
```

Выбираемые значения обозначаются правой скобкой в конце значения. Разделитель ситуаций — *;;*

Цикл for.

Существует два способа задания цикла for.

Стандартный

for *переменная* **in** *список_значений* **do** *команды* **done**.

```
for i in 0 1 2 3
do
echo $i
done
```

С-подобный

```
for ((i=0; c ⩽3; i++))
do
echo $i
done
```

Цикл while

```
while условие;
do;
команда;
done
```

Синтаксис записи условия такой же, как и в условном операторе.

Управление циклами

Для управления ходом выполнения цикла служат команды `break` и `continue`. Они точно соответствуют своим аналогам в других языках программирования.

Команда **break** прерывает исполнение цикла, в то время как **continue** передает управление в начало цикла, минуя все последующие команды в теле цикла.

Управление вводом-выводом команд (процессов)

У любого процесса по умолчанию всегда открыты три файла – **stdin** (стандартный ввод, клавиатура), **stdout** (стандартный вывод, экран) и **stderr** (стандартный вывод сообщений об ошибках на экран).

Эти и любые другие открытые файлы могут быть переправлены. В данном случае термин "перенаправление" означает: получить вывод из файла (команды, программы, сценария) и передать его на вход в другой файл (команду, программу, сценарий).

- `команда > файл` — перенаправление стандартного вывода в файл, содержимое существующего файла удаляется.
- `команда >> файл` — перенаправление стандартного вывода в файл, поток дописывается в конец файла.

- *команда_1 | команда_2* — перенаправление стандартного вывода первой команды на стандартный ввод второй (образование конвейера команд).
- *команда_1 $(команда_2)* — передача вывода *команда_2* в качестве параметров при запуске *команда_1*. Внутри скрипта конструкция *$(команда_2)* может использоваться, например, для передачи результатов работы *команда_2* в параметры цикла *for ... in*.

Работа со строками (внутренние команды bash)

- *${#string}* — выводит длину строки (*string* – имя переменной);
- *${string:position:length}* — извлекает *$length* символов из *$string*, начиная с позиции *$position*.
- *${string:position}* — извлекает подстроку из *$string*, начиная с позиции *$position*.
- *${string#substring}* — удаляет самой короткой из найденных подстроки *$substring* в строке *$string*. Поиск ведется с начала строки.
- *${string##substring}* — удаляет самую длинную из найденных подстроки *$substring* в строке *$string*. Поиск ведется с начала строки.
- *${string/substring/replacement}* — замещает первое вхождение *$substring* строкой *$replacement*.
- *${string//substring/replacement}* — замещает все вхождения *$substring* строкой *$replacement*.

Работа со строками (внешние команды - утилиты)

Для каждой утилиты доступно управление с помощью передаваемых команде параметров. Рекомендуем ознакомиться с документацией по этим командам с помощью команды *`man команда`*.

- **sort** — сортирует поток текста в порядке убывания или возрастания, в зависимости от заданных опций.
- **uniq** — удаляет повторяющиеся строки из отсортированного файла.
- **cut** — извлекает отдельные поля из текстовых файлов (поле – последовательность символов в строке до разделителя).
- **head** — выводит начальные строки из файла на stdout.
- **tail** — выводит последние строки из файла на stdout.
- **wc** — подсчитывает количество слов/строк/символов в файле или в потоке
- **tr** — заменяет одни символы на другие.

Полнофункциональные многоцелевые утилиты

grep

Многоцелевая поисковая утилита, использующая регулярные выражения.

sed

Неинтерактивный "потоковый редактор".

Принимает текст либо из **stdin**, либо из текстового файла, выполняет некоторые операции над строками и затем выводит результат в **stdout** или в файл.

Sed определяет, по заданному адресному пространству, над какими строками следует выполнить операции. Адресное пространство строк задается либо их порядковыми номерами, либо шаблоном.

Например, команда `3d` заставит **sed** удалить третью строку, а команда `/windows/d` означает, что все строки, содержащие **"windows"**, должны быть удалены.

Наиболее часто используются команды:

- `p` — печать (на **stdout**)
- `d` — удаление
- `s` — замена.

awk

Утилита контекстного поиска и преобразования текста, инструмент для извлечения и/или обработки полей (колонок) в структурированных текстовых файлах.

Awk разбивает каждую строку на отдельные поля. По умолчанию поля – это последовательности символов, отделенные друг от друга пробелами, однако имеется возможность назначения других символов в качестве разделителя полей.

Awk анализирует и обрабатывает каждое поле в отдельности.

Регулярные выражения

Это набор символов и/или метасимволов, которые наделены особыми свойствами.
Их основное назначение – поиск текста по шаблону и работа со строками. При построении регулярных выражений используются нижеследующие конструкции (в порядке убывания приоритета), некоторые из которых могут быть использованы только в расширенных версиях соответствующих команд (например, при запуске `grep` с ключом `-E`).

- *c* — Любой неспециальный символ *c* соответствует самому себе.
- *\c* — Указание убрать любое специальное значение символа *c* (экранирование).
- *^* — Начало строки.
- *$* — Конец строки. Выражение *"^$"* соответствует пустой строке.
- *.* — Любой одиночный символ, за исключением символа перевода строки.
- *[...]* — Любой символ из " *...* ". Допустимы диапазоны типа *a-z*. Возможно объединение: *[a-z0-9]*.
- *[^ ...]* — Любой символ не из " *...* ". Так же поддерживаются диапазоны.
- *r** — 0 или более вхождений символа *r*. Также работает с диапазонами.
- *r+* — 1 или более вхождение символа *r*. Также работает с диапазонами.
- *r?* — 0 или 1 вхождение символа *r*. Также работает с диапазонами.
- *\< ... \>* — Границы слова.
- *\{ \}* — Число вхождений выражения. Например, выражение *"[0-9]\{5\}"* соответствует подстроке из пяти десятичных цифр.
- *r1r2* — За *r1* следует *r2*.
- *r1|r2* — *r1* или *r2*.
- *(r)* — Регулярное выражение *r*. Может быть вложенным.

Классы символов POSIX

- `[:class:]` — альтернативный способ указания диапазона символов.
- `[:alnum:]` — соответствует алфавитным символам и цифрам. Эквивалентно выражению `[A-Za-z0-9]`.
- `[:alpha:]` — соответствует символам алфавита. Эквивалентно выражению `[A-Za-z]`.
- `[:blank:]` — соответствует символу пробела или символу табуляции.
- `[:digit:]` — соответствует набору десятичных цифр. Эквивалентно выражению `[0-9]`.
- `[:lower:]` — соответствует набору алфавитных символов в нижнем регистре. Эквивалентно выражению `[a-z]`.
- `[:space:]` — соответствует пробельным символам (пробел и горизонтальная табуляция).
- `[:upper:]` — соответствует набору символов алфавита в верхнем регистре. Эквивалентно выражению `[A-Z]`.
- `[:xdigit:]` — соответствует набору шестнадцатеричных цифр. Эквивалентно выражению `[0-9A-Fa-f]`.