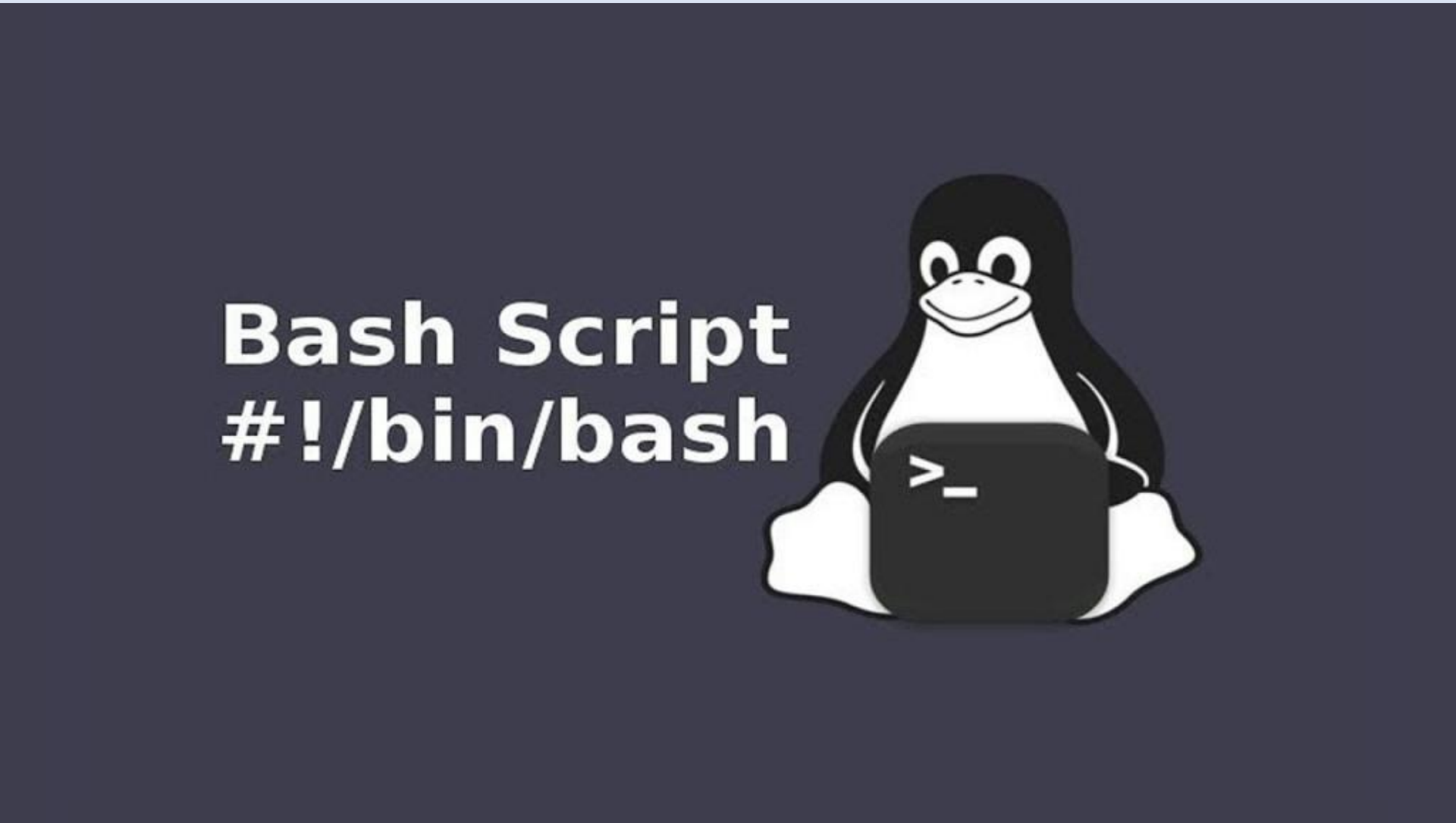
Тема:
Введение в скрипты bash. Планировщики.



План занятия:

1. Правила написания скриптов. Переменные.

2. Условный оператор if. Циклы for и while.

4. Регулярные выражения и утилиты для работы с регулярными выражениями.

5. Планировщики задач cron и at.

1. Правила написания скриптов. Переменные.

Bash — это командный интерпретатор, работающий, как правило, в интерактивном режиме в текстовом окне.

Bash-скрипты — это сценарии командной строки, то есть наборы обычных команд, которые пользователь вводит с клавиатуры. Для автоматизации каких-то рутинных вещей эти команды объединяются в файл-сценарий, который и носит название скрипт.

Bash-скрипты — это файлы, содержащие последовательность команд, которые могут быть выполнены с помощью интерпретатора Bash.

Bash-скрипты могут быть использованы для автоматизации повторяющихся задач. Например:

-создание и отправка отчётов по электронной почте;

-проверка и обновление системы и приложений;

-сканирование и мониторинг сети;

-уведомление по почте, когда не работает сервер или служба, не создана резервная копия и т.д.

shebang (шебанг) — особая запись в начале файла, которая выглядит как сочетание символов #! и путь до интерпретатора. Например, #!/bin/bash укажет системе, что для выполнения кода после этой строки необходимо использовать bash. Запись #!/usr/bin/perl укажет, что для выполнения кода нужно использовать perl. После этой строки можем писать скрипт.

Комментарии в bash определяются символом #. Комментарий может быть добавлен в начале строки или встроен в код, например:

- комментарий в начале строки: # Определяем переменные;
- комментарий в коде: echo “text” # Выводим сообщение на экран терминала

```
1 #!/bin/bash
2 echo "hello World!"
```

Переменные необходимы для хранения информации. С ними можно выполнить два действия:

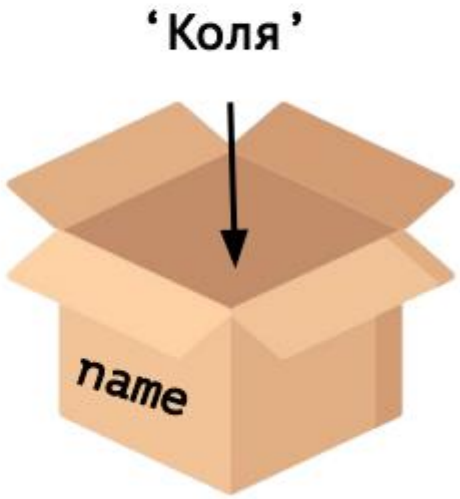
- установить значение переменной;
- прочитать значение переменной.

В bash нет строгих различий между типами переменных. С точки зрения командного интерпретатора любая переменная является строкой.

Переменная – это объект, которому дано имя. Необходимо для хранения данных и промежуточных результатов вычислений.

Объект – это:

- число
- строка
- практически что угодно



В командном интерпретаторе Bash нет строгих различий между типами переменных. Внутренне все переменные в Bash представляются как строки, и командный интерпретатор не выполняет автоматического преобразования типов данных.

Это означает, что при выполнении операций или сравнении значений переменных, Bash рассматривает их как строки. Например, при сложении двух переменных, Bash просто конкатенирует их значения вместо выполнения арифметической операции, если значения не были предварительно преобразованы в числа.

```
num1=5
num2=10
result=$num1$num2
echo $result  # Вывод: 510
```

Однако можно явно преобразовывать значения переменных в другие типы данных, если это необходимо, используя соответствующие операторы и функции в Bash.

Например, для выполнения арифметических операций можно использовать оператор (( )), а для преобразования строки в число - команду **let** или арифметическое выражение $(( )).

```
num1=5
num2=10
result=$((num1 + num2))
echo $result  # Вывод: 15
```

```
num1=5
num2=10
let "result = num1 + num2"
echo $result  # Вывод: 15
```

В Bash нельзя использовать операторы арифметики с числами с плавающей точкой.

Bash поддерживает только целочисленную арифметику.

Для работы с числами с плавающей точкой, можно использовать утилиту bc.

```
x=5.2
y=13.985
result=$(echo "$x + $y" | bc)
echo $result
```

scale=2 - точность результата, количество знаков после запятой

```
x=5.2
y=3.14
z=$(echo "scale=2; $x / $y" | bc)
echo $z
```

bash. Типы данных

Строки (Strings): Строки представляют последовательность символов и обозначаются с помощью одинарных или двойных кавычек. Например:

```
name="John"
message='Hello, World!'
```

Числа (Numbers): В Bash числа могут быть целыми (integer) или с плавающей точкой (floating-point). Числа могут быть заданы напрямую или присвоены переменным. Например:

```
count=10
pi=3.14159
```

Массивы (Arrays): Массивы в Bash позволяют хранить наборы элементов. Элементы массива могут быть любого типа данных, включая строки и числа.

```
fruits=("apple" "banana" "orange")
numbers=(1 2 3 4 5)
```

Пустое значение (Null): В Bash пустое значение обозначается ключевым словом null или пустыми кавычками ''.

```
empty_value=null
```

Булевы значения (Boolean)

```
is_true=true
is_false=false
```

Кроме того, Bash поддерживает различные специальные типы данных и структуры, такие как ассоциативные массивы (associative arrays) и файловые дескрипторы (file descriptors), но они не так часто используются в повседневном программировании на Bash.

Важно отметить, что в Bash типы данных не явно объявляются, и переменные могут изменять свой тип в процессе выполнения программы.

Правила написания скриптов. Переменные

Переменные bash

Переменные окружения	Пользовательские переменные	Специальные переменные
$PWD — текущий каталог. $ID — покажет имя текущего пользователя и группы, в которых он состоит. $PATH — покажет путь до исполняемых файлов	a=123 присвоит переменной a значение 123 a=$(ls) присвоит переменной a результат работы команды ls	Переменные подстановки: $0 $1 ..$9 $? — статус выполнения предыдущей команды или скрипта

В оболочке Bash **переменные подстановки** $0, $1, $2, ..., $9 используются для доступа к аргументам командной строки переданным скрипту или команде. Вот их значение:

$0: Эта переменная содержит имя самого скрипта или команды, которая была запущена..

$1, $2, ...: Эти переменные содержат значения аргументов командной строки, переданных скрипту или команде. $1 содержит значение первого аргумента, $2 - второго и так далее. Всего доступно 9 аргументов: от $1 до $9.

Например, предположим, у вас есть скрипт my_script.sh, который вы запускаете следующим образом:

```
./my_script.sh arg1 arg2 arg3
```

В этом случае:

$0 будет равно my_script.sh (или полному пути к скрипту).

$1 будет равно arg1.

$2 будет равно arg2.

$3 будет равно arg3.

$4, $5, ..., $9 будут пустыми

Некоторые переменные окружения

- **USER**: текущее имя пользователя, использующего систему
- **HOME**: домашний каталог текущего пользователя
- **PATH**: список каталогов, разделенных двоеточиями, в которых система ищет команды
- **PS1**: основная строка приглашения (для определения отображения приглашения оболочки)
- **PWD**: текущий рабочий каталог
- **_**: самая последняя команда, выполненная в системе пользователем
- **SHELL**: оболочка, используемая для интерпретации команд в системе, она может быть много разных (например, bash, sh, zsh или другие)
- **LANG**: кодировка языка, используемая в системе
- **UID**: текущий UID для пользователя
- **HOSTNAME**: имя компьютера системы
- **TERM**: указывает тип терминала
- **OLDPWD**: предыдущий рабочий каталог
- **BASHOPTS**: список параметров, которые использовались при выполнении bash.
- **IFS**: внутренний разделитель полей для разделения ввода в командной строке. По умолчанию это пробел.
- **SHELLOPTS**: параметры оболочки, которые можно установить с помощью параметра set.

Пользовательские переменные

Пользовательские переменные Bash Linux могут быть названы любой текстовой строкой длиной до 20 символов, состоящей из букв, цифр и символа подчёркивания.

В названии учитывается регистр букв, поэтому переменная **Var1** не является переменной **var1**.

Присвоение значения переменной Bash выполняется с помощью знака равенства (=).

Слева и справа от знака не должно быть разделяющих символов по типу пробела.

Обращение к пользовательским переменным осуществляется так же, как и к системным, — с помощью знака доллара ($). Он не используется, когда переменной присваивается значение.

```
var1=50
var2=-120
var3=hello
var4="Hello World!!!"

echo $var1
(( sum=$var1 + $var2))
echo $sum
```

Арифметические операции

В bash существует множество способов выполнения арифметических операций.

Рассмотрим некоторые из них.

В bash нет родной поддержки чисел с плавающей точкой. Но есть утилиты, которые умеют это делать.

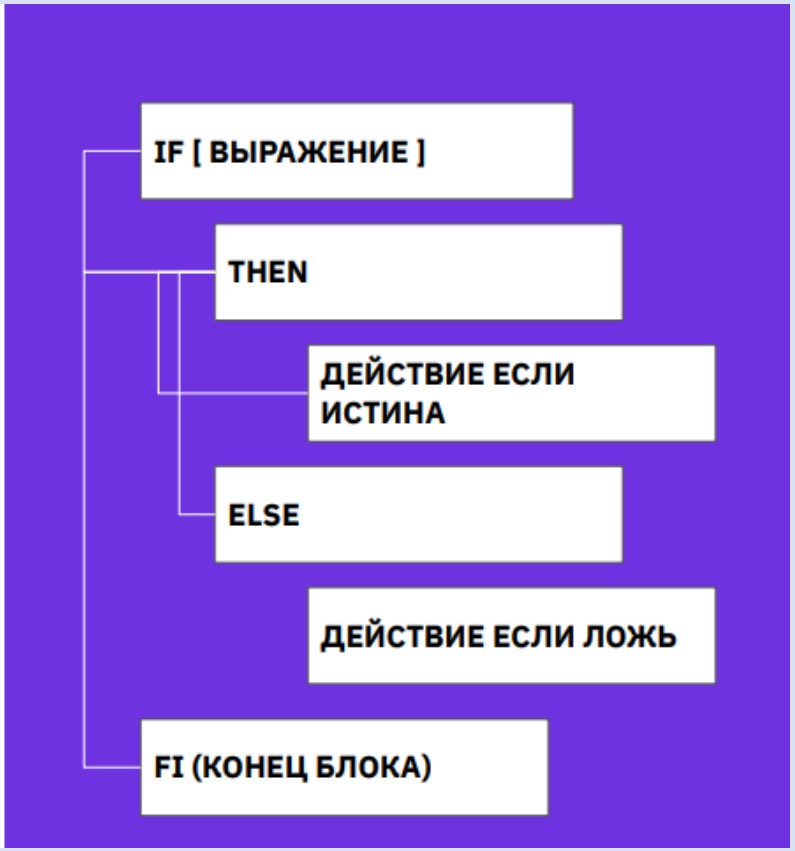
```
#!/bin/bash
# Арифметика с let

let sum=2+2 # если выражение без кавычек, пробелов быть не должно
echo $sum

let sum="2 + 2"
echo $sum

let sum++  # инкремент, sum=sum+1
echo $sum

let mult="4 * 5"
echo $mult

let x=mult/3
echo $x
```

```bash
#!/bin/bash
# Простая арифметика с двойными скобками

A=$(( 4 + 5 ))  # Базовый синтаксис. Можно ставить пробелы без использования кавычек.
echo $A # 9

A=$((3+5))      # Работает и без пробелов.
echo $A # 8

B=$(( A + 3 ))  # Можно использовать переменные без $ перед ними.
echo $B # 11

B=$(( $A + 4 )) # А можно и с $
echo $B # 12

(( B++ ))       # Увеличение переменной на 1. Символ $ не нужен.
echo $B # 13

(( B += 3 ))    # Увеличение переменной на 3. Это краткая форма записи b = b + 3.
echo $B # 16

A=$(( 4 * 5 ))  # символ * не нужно экранировать.
echo $A # 20
```

```bash
#!/bin/bash
# expr вместо сохранения результата в переменную по умолчанию печатает ответ.

expr 2 + 3    # базовый синтаксис, результат  5

expr "5 + 7"  # в терминал выведет строку "5 + 7"

expr 2+3      # если пробелов нет, выражение будет выведено в терминал без вычисления

expr 20 \* 5  # Некоторые символы нужно экранировать

expr 15 % 4   # остаток от целочисленного деления двух чисел

sum=$( expr 8 + 7 )  # сохраняем результат в переменную sum
echo $sum
```

2. Условный оператор if и циклы.

Условный оператор if и циклы



Условный оператор if и циклы

Операции сравнения (наиболее используемые)

Проверка файлов:

- -e возвращает true (истина), если файл существует (exists);
- -d возвращает true (истина), если каталог существует (directory).

Сравнение строк:

- = или == возвращает true (истина), если строки равны;
- != возвращает true (истина), если строки не равны;
- -z возвращает true (истина), если строка пуста;
- -n возвращает true (истина), если строка не пуста.

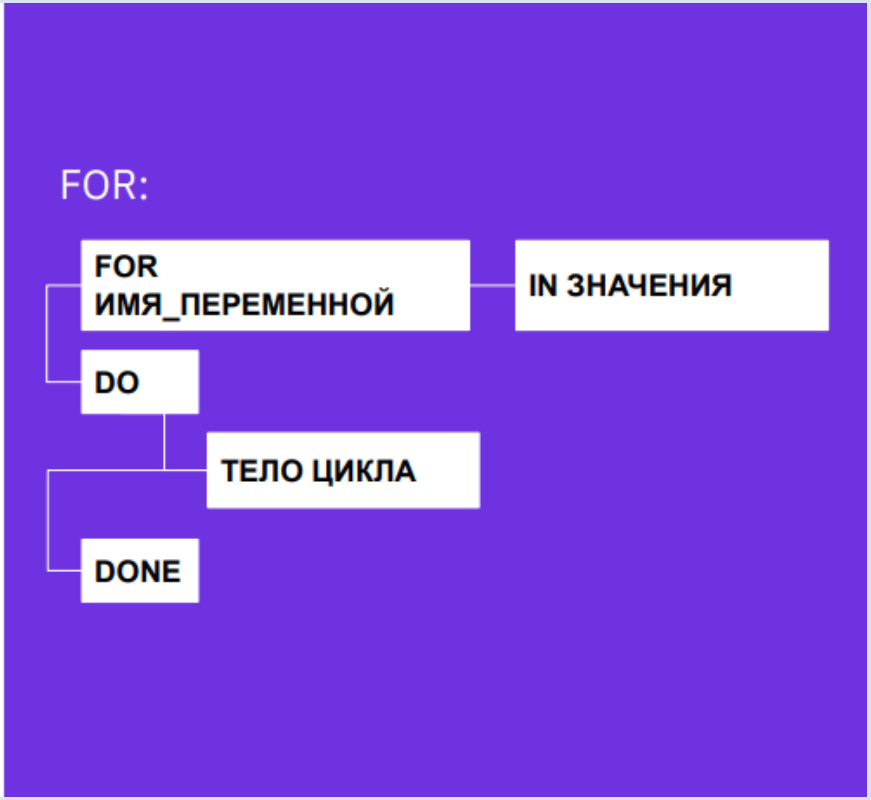
Сравнение целых чисел:

- -eq возвращает true (истина), если числа равны (equals);
- -ne возвращает true (истина), если числа не равны (not equal).

```bash
#!/bin/bash
# проверка числа на четность
echo -n "Введите число: "
read number   # Эта команда берет ввод и сохраняет его в переменной

if [ $number -eq 0 ]
then
  echo "Число равно нулю"
elif [ $(($number % 2)) -eq 0 ]
then
  echo "Вы ввели $number. Число четное"
else
  echo "Вы ввели $number. Нечетное число."
fi
```

Условный оператор if и циклы

Цикл — последовательность, которая позволяет выполнить определённый участок кода заданное количество раз.

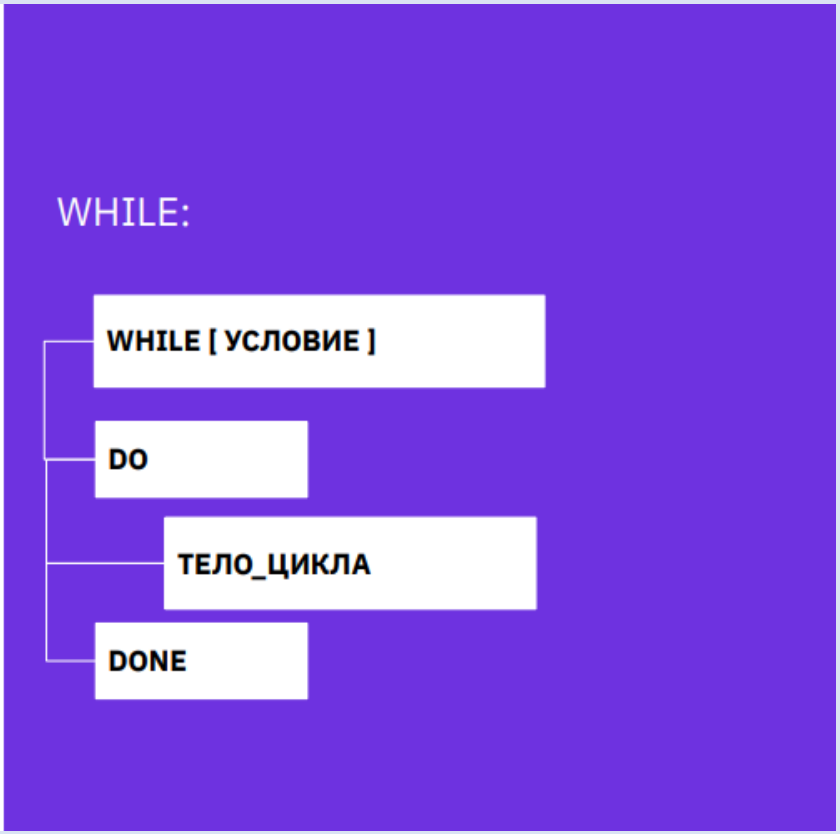


```
!/bin/bash
# цикл for

for number in 1 2 3 4 5
do
    echo $number
done

for number in {100..105} # {100..105} - массив от 100 до 105 включительно
do
    echo $number
done

for smb in {a..z}  # {100..105} - массив от a до z включительно
do
    echo $smb
done

for ((i=1; i < 10; i++))  # цикл сос счетчиком
do
    echo $i
done
```

Условный оператор if и циклы

Здесь в качестве [условие] осуществляются операции сравнения и проверки, аналогичные условному оператору if.

Тело_цикла — команды, которые будут выполняться до тех пор, пока условие возвращает true (истина).



```
!/bin/bash
# -gt - больше, -ge - больше или равно
# -lt - меньше, -le - меньше или равно

x=1
while [ $x -lt 5 ]
do
echo "Значение счетчика: $x"
x=$(( $x + 1 ))
done

# с помощью while мы можем прочитать несколько строк из стандартного ввода. выход - Ctrl+D
while read line
do
echo $line
done
```

Команда read

Команда read позволяет считывать значения, которые пользователь вводит с клавиатуры, и сохранять их в переменные.

Синтаксис команды read выглядит следующим образом:

```
read [опции] [переменная]
```

Некоторые распространенные опции команды read включают:

-p "prompt": Отображает приглашение (prompt) для ввода пользователю.

-s: Вводимые символы не будут отображаться на экране.

-n count: Ограничивает количество символов, которое пользователь может ввести.

-t timeout: Устанавливает время ожидания ввода в секундах. Если пользователь не введет значение в указанное время, выполнение команды read продолжится с пустым значением.

- a: запись ввода в массив.

```
echo "Введите ваше имя:"
read name

echo "Привет, $name! Как дела?"
```

```
echo "Введите пароль:"
read -s password
echo "Вы ввели пароль: $password"
```

```
echo "Введите несколько слов через пробел:"
read -a words
echo "Вы ввели следующие слова:"
for word in "${words[@]}"; do
    echo "$word"
done
```

```
#!/bin/bash
read -p "What is your name: " name
echo "Hello, $name"
```

4. Регулярные выражения и утилиты для работы с регулярными выражениями.

Регулярные выражения — инструмент, предназначенный для поиска, а также обработки текста по заданному шаблону.

Используя регулярные выражения, мы можем изменять текст, искать строки в файле, фильтровать список файлов согласно каким-то условиям и т. д.

Регулярные выражения — неотъемлемая часть командного интерпретатора bash. Они постоянно применяются в работе с командной строкой.

Регулярные выражения и утилиты для работы с регулярными выражениями

Основные символы-модификаторы

- \ — с обратной косой черты начинаются буквенные спецсимволы,
- * — указывает, что предыдущий символ может повторяться 0 или больше раз;
- + — указывает, что предыдущий символ должен повториться больше 1 или больше раз;
- ? — предыдущий символ может встречаться 0 или 1 раз;
- {n} — указывает сколько раз (n) нужно повторить предыдущий символ;
- {N,n} — предыдущий символ может повторяться от N до n раз;
- . — любой символ, кроме перевода строки;
- [az] — любой символ, указанный в скобках;
- x|y — символ x или символ y;
- [^az] — любой символ, кроме тех, что указаны в скобках;
- [a-z] — любой символ из указанного диапазона;
- [^a-z] — любой символ, которого нет в диапазоне;
- \b — обозначает границу слова с пробелом;
- \B — означает, что символ должен не быть окончанием слова;
- \d — означает, что символ — цифра;
- \D — нецифровой символ;
- \n — символ перевода строки;
- \s — любой пробельный символ: пробел, табуляция и так далее;
- \S — любой непробельный символ;
- \t — символ табуляции;
- \v — символ вертикальной табуляции;
- \w — любой буквенный символ, включая подчёркивание;
- \W — любой буквенный символ, кроме подчёркивания;
- \uXXX — конкретный указанный символ Unicode.

Регулярные выражения и утилиты для работы с регулярными выражениями

grep находит на вводе целые строки, отвечающие заданному регулярному выражению, и выводит их, если вывод не отменён специальным ключом. Grep работает с регулярными выражениями POSIX (BRE), а также все остальные, включая PCRE в разных режимах. У него есть модификация egrep, которая позволяет расширенный синтаксис (ERE).

Если нужно найти конкретную папку или один файл среди сотни других, то мы можем передать вывод команды **ls** в **grep** через вертикальную черту (|), а уже **grep**-у параметром передать нужное слово.

```
$ ls | grep Documents
```

Если же нужно найти не одно слово, а словосочетание или целое предложение, то параметр команды **grep** должно быть выделено кавычками.

```
$ ls | grep 'My Documents'
```

Поиск указанных в кавычках слов в файле Students.txt

```
$ grep 'Class 1' Students.txt
```

Регулярные выражения и утилиты для работы с регулярными выражениями

02

sed — потоковый текстовый редактор. Позволяет редактировать потоки данных на основе заданных правил. С помощью SED можно провести простые операции по поиску и замене слов в тексте.

Регулярные выражения и утилиты для работы с регулярными выражениями

awk — более мощная, чем SED, утилита для обработки потока данных. С точки зрения AWK данные разбиваются на наборы полей, то есть наборы символов, разделённых разделителем. AWK — это практически полноценный язык программирования, в котором есть свои переменные, операторы выбора и циклы. AWK — родоначальник языка perl.

Часто востребованная задача - выборка полей из стандартного вывода.

По умолчанию **awk** разделяет поля пробелами. Если вы хотите напечатать первое поле, вам нужно просто использовать функцию echo и передать ей параметр **$1**, если функция одна, то скобки можно опустить:

```
$ echo 'one two three four' | awk '{print $1}'
```

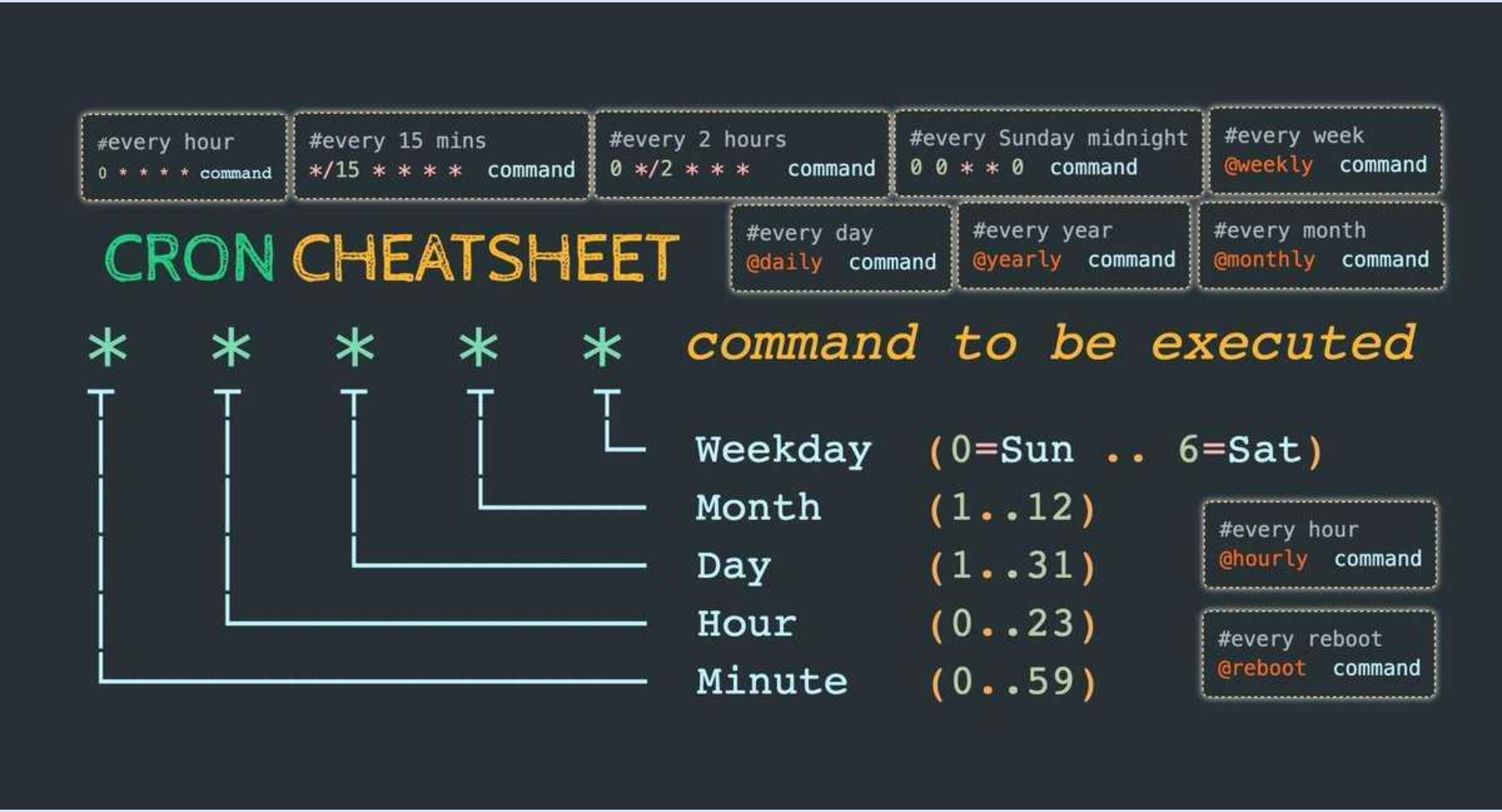
Если поля разделены не пробелами, а другим разделителем, просто укажите в параметре **-F** нужный разделитель в кавычках, например ":"

```
$ echo 'one mississippi:two mississippi:three mississippi:four mississippi' | awk -F":" '{print $4}'
```

Иногда нужно обработать данные с неизвестным количеством полей. Если вам нужно выбрать последнее поле можно воспользоваться переменной **$NF**. Вот так вы можете вывести последнее поле

```
$ echo 'one two three four' | awk '{print $NF}'
```

5. Планировщики задач cron и at.



Зачем нужны планировщики?

Планировщики задач играют важную роль в автоматизации и планировании выполнения задач в компьютерных системах. Они предоставляют способ запуска задач по расписанию или в определенное время без необходимости вмешательства пользователя. Вот некоторые основные причины, по которым планировщики задач являются полезными:

Автоматизация: Планировщики задач позволяют автоматизировать выполнение повторяющихся задач, таких как ежедневные резервные копии, отправка отчетов, обновление программного обеспечения и т. д. Вместо того, чтобы каждый раз выполнять эти задачи вручную, планировщик может запускать их автоматически в установленное время или по расписанию.

Эффективное использование ресурсов: Планировщики задач позволяют оптимизировать использование ресурсов компьютерной системы. Например, вы можете запланировать выполнение определенных задач в периоды низкой активности, чтобы избежать перегрузки системы в пиковые часы.

Улучшение надежности и точности: Планировщики задач помогают гарантировать, что задачи будут выполняться вовремя и без ошибок. Вы можете предварительно запланировать задачи и быть уверенными, что они будут запущены в указанное время, что особенно важно для критических операций.

Гибкость и настраиваемость: Планировщики задач предлагают широкий спектр настроек и возможностей. Вы можете определить точное время запуска задачи, указать периодичность выполнения задачи, использовать различные условия и применять фильтры для определения условий выполнения задач.

Основные принципы работы планировщиков задач включают:

Задание расписания: Планировщик задач позволяет задать расписание, включающее время и периодичность выполнения задачи. Расписание может быть определено в виде конкретного времени и даты, повторения через определенные интервалы времени, ежедневно, еженедельно и т. д. Это позволяет указать, когда и как часто задача должна быть выполнена.

Запуск и выполнение задач: Планировщик следит за расписанием и запускает задачи в соответствии с указанными условиями. Когда наступает заданное время или условия выполнения задачи соблюдаются, планировщик запускает задачу. В зависимости от планировщика, задача может быть выполнена в контексте текущего пользователя или в фоновом режиме, без взаимодействия с пользователем.

Управление завершением задач: Планировщик отслеживает выполнение задачи и управляет ее завершением. По завершении задачи, планировщик может предоставить отчет о выполнении или выполнить дополнительные действия, такие как отправка уведомления или запуск следующей задачи в расписании.

Обработка ошибок и исключений: Планировщики задач обычно обрабатывают ошибки и исключения, связанные с выполнением задачи. Если задача не может быть выполнена по какой-либо причине (например, отсутствие доступных ресурсов или неправильные параметры), планировщик может предпринять соответствующие действия, такие как повторная попытка выполнения задачи или отправка уведомления об ошибке.

Интерфейс управления: Планировщики задач обычно предоставляют интерфейс управления, который позволяет пользователям создавать, редактировать и удалять задачи в расписании. Это может быть командная строка, графический интерфейс или специальные инструменты и API для программистов.

Планировщик задач crontab

Crontab (Cron Table) - это стандартный планировщик задач в операционных системах Unix и Linux. Он предоставляет возможность создавать и управлять расписанием задач, которые должны быть выполнены в определенное время или по расписанию.

В crontab каждый пользователь может иметь свой собственный файл crontab, в котором определены задачи для выполнения.

Задача в crontab представляет собой строку, содержащую расписание выполнения и команду, которую нужно выполнить.

Расписание определяется с помощью пяти полей, определяющих минуты, часы, дни месяца, месяцы и дни недели, когда задача должна быть выполнена.

Возможности crontab включают:

Задание расписания: Crontab позволяет задать точное расписание выполнения задачи, указывая минуты, часы, дни месяца, месяцы и дни недели.

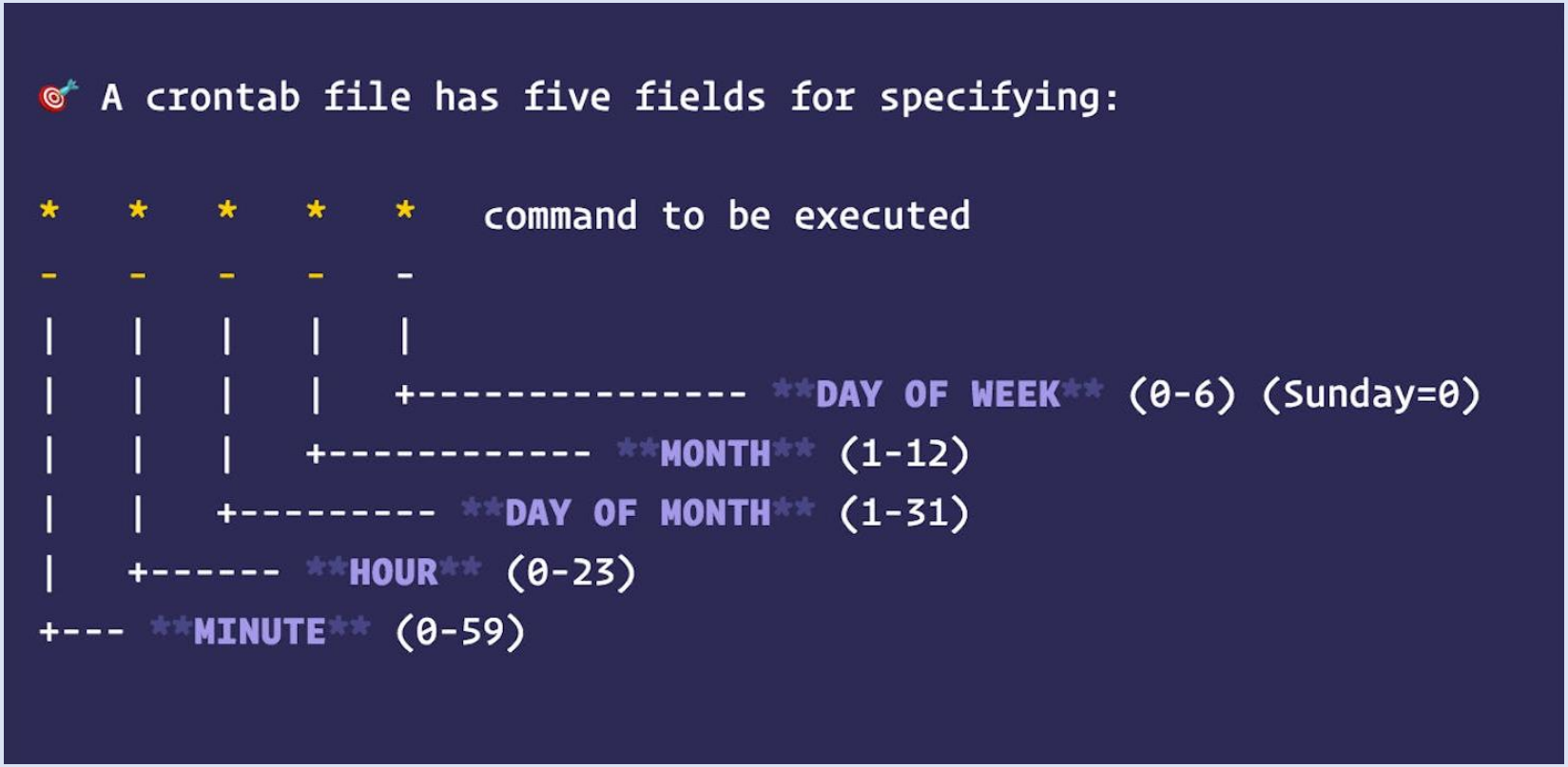
Повторение: Можно настроить повторение задачи через определенные интервалы времени с помощью символа */n, где n - интервал.

Специальные символы: Crontab поддерживает специальные символы, такие как * (звездочка), которая означает "каждый", и , (запятая), которая позволяет задать несколько значений.

Перенаправление вывода: Результат выполнения задачи может быть перенаправлен в файл или отправлен по электронной почте.

Управление crontab: Пользователь может создавать, редактировать и удалять свои crontab-файлы с помощью команды crontab в командной строке.

Формат строки следующий:



Вместо * введите значения для минут (0-59), часов (0-23), дней месяца (1-31), месяцев (1-12) и дней недели (0-6, где 0 - воскресенье).

Например, чтобы выполнить задачу каждый день в 8:30 утра, строка будет выглядеть так:

```
30 8 * * * command_to_execute
```

Запуск команды каждую неделю в определенный день и время(command_to_execute будет выполняться каждую среду в 12:00.):

```
0 12 * * 3 command_to_execute
```

Запуск команды каждый год в определенный месяц, день и время, например каждый год 1-й января в 9:00 утра.

```
0 9 1 1 * command_to_execute
```

В Unix-подобных операционных системах, таких как Linux, существуют два типа crontab-файлов: системный crontab и пользовательский crontab.

Системный crontab:

Системный crontab применяется для задания глобальных задач, которые выполняются от имени системы. Файл системного crontab обычно располагается в директории /etc и называется crontab. Для редактирования системного crontab-файла требуются привилегии суперпользователя (root).

Пользовательский crontab:

Каждый пользователь системы может иметь свой собственный crontab-файл. Он используется для задания персональных задач, которые выполняются от имени конкретного пользователя. Каждый пользователь может редактировать свой crontab-файл, независимо от других пользователей. Пользовательские crontab-файлы обычно хранятся в директории /var/spool/cron или /var/spool/cron/crontabs и называются именем пользователя.

Оба типа crontab-файлов имеют одинаковый синтаксис и позволяют задавать расписание выполнения задач. Однако, системный crontab работает в контексте системы и может выполнять задачи, требующие привилегий суперпользователя. Пользовательский crontab работает от имени конкретного пользователя и может выполнять задачи с правами этого пользователя.

Системный crontab

Для планирования задач в системном crontab необходимо отредактировать файл /etc/crontab.

Кроме того, директории /etc/cron.hourly, /etc/cron.daily, /etc/cron.weekly и /etc/cron.monthly представляют собой специальные директории в Linux, где можно размещать скрипты или команды, которые должны выполняться с определенной периодичностью.

Когда системный cron-демон запускается, он проверяет эти директории и автоматически выполняет все исполняемые файлы, находящиеся внутри них, согласно соответствующему расписанию (каждый час, день, неделю или месяц).

Пользовательский cron

Пользовательский cron в Linux позволяет каждому пользователю планировать и автоматизировать выполнение задач на своем уровне. Каждый пользователь может создать свой собственный crontab-файл, в котором указываются задачи и их расписание.

Для создания и редактирования пользовательского cron-файла используется команда crontab. Каждый пользователь может иметь только один активный crontab-файл.

Чтобы открыть редактор для редактирования crontab-файла текущего пользователя, введите команду:

crontab -e

Команда crontab -l позволяет просмотреть содержимое пользовательского crontab-файла, то есть список задач и их расписание для текущего пользователя. Если вы выполните эту команду, вы увидите задачи, которые были добавлены в ваш crontab-файл.

Команда crontab -r используется для удаления пользовательского crontab-файла, то есть всех задач и расписания, связанных с текущим пользователем. После выполнения этой команды весь пользовательский crontab будет удален и задачи больше небудут выполняться автоматически.

Планировщик at

Планировщик at в Linux предоставляет возможность запускать одноразовые задачи в определенное время в будущем. Он отличается от cron, который предназначен для периодического запуска задач.

С помощью at вы можете запланировать выполнение команды или скрипта на определенный момент времени. Задачи, запланированные с использованием at, выполняются только один раз по указанному расписанию.

Некоторые основные команды, связанные с at:

at: Команда at используется для запланирования выполнения команды или скрипта в определенное время. Вы можете указать время запуска команды явно или с использованием относительных значений, таких как "now + 1 hour". Например:

```
at 10:00 PM
at now + 1 hour
```

Домашнее задание:

1. Повторить материал лекции.
2. Изучить дополнительные материалы.