GeekBrains

Общая версия Linux. Уровень 1

Введение в скрипты bash. Планировщики задач crontab и at

На этом уроке

1. Познакомимся с основами написания скриптов в bash.
2. Разберём, какие переменные существуют в bash.
3. Посмотрим, как использовать условный оператор if, а также циклы for и while.
4. Разберём правила написания регулярных выражений в bash, а также утилиты, которые пригодятся в работе с регулярными выражениями.
5. Научимся составлять задания для планировщиков задач.

Оглавление

Глоссарий

Правила написания скриптов. Переменные

- Переменные
  - Переменные окружения
  - Пользовательские переменные
  - Специальные переменные

Условный оператор if и циклы

- Условный оператор if
  - Операции проверки файлов (наиболее используемые)
  - Операции сравнения строк (наиболее используемые)
  - Операции сравнения целых чисел (наиболее используемые)
- Циклы for и while

Регулярные выражения и утилиты для работы с регулярными выражениями

- grep
- SED
- AWK

Crontab и at

- At
- Crontab
  - Дополнительные переменные cron
  - Утилита crontab

Практическое задание

Дополнительные материалы

Используемые источники

Глоссарий

Bash — это командный интерпретатор, работающий, как правило, в интерактивном режиме в текстовом окне. Он также может читать команды из текстового файла, который называется скриптом. Как и все Unix-оболочки, он поддерживает автодополнение имён файлов и директорий, подстановку вывода результата команд, переменные, контроль за порядком выполнения, операторы ветвления и цикла.

Переменные окружения — это специальные переменные, определённые оболочкой и используемые программами во время выполнения. Они могут определяться системой и пользователем. Системные переменные окружения Linux определяются системой и используются программами системного уровня.

POSIX — набор стандартов, описывающих интерфейсы между операционной системой и программным обеспечением. Стандарт создан для обеспечения совместимости различных Unix-подобных операционных систем.

Регулярные выражения — инструмент для поиска текста по шаблону, обработки и изменения строк, который можно применять для следующих задач: поиск и замена текста в файле, проверка строки на соответствие шаблону и т. д.

Планировщик заданий — программа (служба или демон), часто называемая сервисом операционной системы, которая запускает другие программы в зависимости от различных критериев, таких как, например, наступление определённого времени.

Правила написания скриптов. Переменные

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. По сути, любое наше действие в командной строке — это мини-скрипт, например поиск файлов, начинающихся с точки в домашней директории. Можно выполнить команду `ls -A` и пересчитать все файлы, которые скрыты, то есть начинаются с точки, а можно перенаправить через `| (pipe)` вывод работы команды `ls -A` на ввод утилите grep, которая согласно шаблону найдёт нужные нам файлы и передаст найденные строки утилите wc, которая подсчитает их количество: `ls -A |grep '^\.'|wc -l`. Таким образом мы упростили себе задачу, для этого и нужны скрипты.

Скрипты можно описать в виде файла сценария, а можно в виде однострочного скрипта, то есть последовательности команд, разделённых между собой знаком «;». Такая запись удобна, когда

необходимо выполнить набор простых действий, например создать каталог и в этом каталоге создать файл: `mkdir dir; touch file`. Эти же команды можно записать в файл, используя свой любимый редактор, при этом вместо точки с запятой мы используем перенос строки:

Запустить такой скрипт можно двумя способами:

1. Выполнить команду `bash script`.
2. Сделать файл исполняемым и просто указать имя файла, используя полный путь до него: `/home/user/script`, либо перед именем указать **./**, что говорит о том, что файл запускается именно из текущей директории: `./script`.

Недостаток первого способа: перед исполнением файла нам необходимо указать интерпретатор и путь до файла. Недостаток второго способа: в нём будет использоваться интерпретатор команд по умолчанию, а у пользователя может в качестве оболочки стоять интерпретатор, отличный от bash, и тогда скрипт может отработать с ошибкой. Чтобы избежать этих недостатков, существует особая запись в начале файла — **shebang (шебанг),** которая выглядит как сочетание символов **#!** и путь до интерпретатора. Например, `#!/bin/bash` укажет оболочке, 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.

Правилом хорошего тона считается комментировать свой код, и это касается не только bash, но и других языков программирования. 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**.

Пробел после символа **#** не обязателен и нужен для удобства чтения.

Переменные

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

- установить значение переменной;
- прочитать значение переменной.

Переменные могут быть определены и вызваны в любом месте скрипта. Для вызова используется символ $ перед именем переменной. Например:

`a=123` — присвоение переменной **a** значения 123.

`echo $a` — вызов значения переменной $a (на экран будет выведено число 123).

В качестве имени переменной могут быть использованы буквы или слова, написанные латиницей, а также сочетания букв и цифр. Значением переменной может быть любое слово или группа слов, цифра, а также команда. В bash нет строгих различий между типами переменных. С точки зрения командного интерпретатора любая переменная является строкой.

Переменные в bash можно разбить на три группы:

Переменные окружения

Это специальные переменные, которые определяются оболочкой и используются программами в момент исполнения. Их можно разделить на две группы:

- **Пользовательские** — переменные, которые получают своё значение в момент входа пользователя в систему. Пользовательские переменные окружения прописываются в файлах **.bashrc, .profile**, которые расположены в домашнем каталоге пользователя.
- **Системные** — переменные, которые получают своё значение при старте системы, и влияют на всё системное окружение: на всех пользователей, запускаемые службы и программы. Прописываются в файлах **/etc/profile, /etc/bash.bashrc**.

Примеры переменных окружения:

$PWD — текущий каталог.

$ID — покажет имя текущего пользователя и группы, в которых он состоит.

$PATH — покажет путь до исполняемых файлов.

Переменные можно переопределить в рамках текущей сессии. Например, `PATH=$PATH:/opt/my_progs/bin` — для сохранения всех путей до исполняемых файлов мы при переопределении переменной **PATH** присваиваем её предыдущее значение **$PATH** и добавляем, используя разделитель «**:**», новый путь: `/opt/my_progs/bin`. Без сохранения в файлы bashrc или .profile это переопределение будет действовать до конца сеанса пользователя или же всё время работы терминала.

Пользовательские переменные

Это переменные, которые определяет пользователь в момент написания скрипта. Например, присвоение строки hello world: `a="hello world"`. Если переменная состоит из нескольких слов, строка берётся в кавычки. Если значение переменной — команда, то либо она берётся в обратные апострофы: `a=\`ls\``, либо используется **$(command)**, например, `a=$(ls)`.

Специальные переменные

Это ряд переменных bash, отличных от переменных окружения, которые предопределены операционной системой. Например:

- **Переменные подстановки: $0 $1 ..$9**, где **$0** — это имя скрипта, **$1..$9** — аргументы, которые мы можем передать скрипту.
- **$?** — статус выполнения предыдущей команды или скрипта, в своём роде логическая переменная. Если возвращает 0 — скрипт завершился успешно, любое другое значение — ошибка выполнения.

Чтобы выполнять арифметические операции с переменными, в **bash** существует специальная запись: **$(( ))**. В скобках указываются числа и операции над ними.

Например:

```
a=$((5+5))
echo $a
```

Запись выведет на экран 10.

Условный оператор if и циклы

Условный оператор if

Условные операторы предоставляют возможность решить, продолжать дальнейшие действия или нет. Решение принимается на основе вычисления выражения. В bash, как и в других языках программирования, основной оператор выбора — конструкция **if/then/else/fi (если/тогда/иначе/конец_блока)**.

Конструкция выглядит следующим образом:

```
if [ выражение ]
    then
        Действия, если выражение истинно
    else
        Действия в противоположном случае
fi
```

Оператор **fi** обязателен, им закрывается проверка условия. Оператор **else** не обязателен, поскольку действий в противном условию случае может не быть.

[ ] — аналог команды **test**. Это команда, которая проверяет типы файлов и сравнивает значения. Подробности можно прочитать на странице справочного руководства **man test**.

[[ ]] — аналог оператора **[ ]**, но с более широкими возможностями. К примеру, у него лучшая поддержка регулярных выражений.

(( )) — используется для арифметических операций.

Используя условный оператор **if**, мы можем осуществить операции проверки и сравнений.

Операции проверки файлов (наиболее используемые)

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

Например:

```
if [ -e file_name ]
   then
       echo "true"
  else
      echo "false"
fi
```

Вернётся true, если файл существует, и false, если такого файла нет. Аналогично выполняется проверка существования каталога.

Операции сравнения строк (наиболее используемые)

- **=** 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.

Простой пример проверки передачи параметра скрипту:

```
#!/bin/bash
a=$1 #присваиваем переменной a значение переменной подстановки $1, используем
как параметр
#проверяем, что параметр задан
if [ -z $a ]
   then
       echo "Error" #если строка пустая, сообщаем об ошибке
       exit # завершаем скрипт
  else
      echo $a # в противном случае выводим на экран значение параметра
fi
```

Операции сравнения целых чисел (наиболее используемые)

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

Например:

```
#!/bin/bash
a=$1
b=$2
if [ $a -eq $b ]
 then
       echo "true"
else
       echo "false"
fi
```

Скрипт сравнивает два числа, переданные в качестве параметров. В случае равенства вернёт true (истина), в противном случае вернёт false (ложь).

Циклы for и while

Цикл — последовательность, которая позволяет выполнить определённый участок кода заданное количество раз. Существует несколько типов циклов. Мы рассмотрим два наиболее часто используемых.

Цикл **for** позволяет организовать перебор последовательности значений. Структура цикла:

```
for имя_переменной in значения
do
   тело_цикла
done
```

Здесь **имя_переменной** — переменная, которая будет получать значения из массива «значения». В качестве такого массива может быть задана последовательность чисел, какой-то набор слов, результат работы команды. Т**ело_цикла** — команды, которые будут обрабатывать переменную.

Пример:

`for i in $(ls); do echo $i;done` — здесь переменная **i** получает значения из работы команды **ls**, и команда **echo** выводит это значение на терминал.

Цикл **while** выполняется до тех пор, пока условие истинно. Структура цикла:

```
while [ условие ]
do
   Тело_цикла
done
```

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)**.

Пример бесконечного цикла:

`while [ true ];do echo "true"; done` — так как условие всегда «истина», то на экран терминала всегда будет выводиться слово true.

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 можно разделить на два типа: **POSIX** и **PCRE** (perl-совместимые регулярные выражения). Основное различие — набор используемых символов. В bash используются регулярные выражения POSIX, которые делятся на два типа: BRE (базовые регулярные выражения) и ERE (расширенные регулярные выражения). Они различаются используемыми символами: в ERE их больше, и синтаксис ближе к регулярным выражениям PCRE.

В регулярных выражениях используется два типа символов:

1. **Обычные символы** — буквы, цифры, знаки препинания — всё, из чего состоят слова и строки.
2. **Метасимволы** — специальные символы, при помощи которых усиливается регулярное выражение. Эти символы используются для замены других символов или их последовательностей, например для группировки символов.

Метасимволы также можно разделить на группы:

- **Символы-якоря** — символы, определяющие позицию шаблона в тексте. **^** (символ «каретка») обозначает начало строки, **$** (знак доллара) обозначает конец строки.
- **Символы-модификаторы** — символы, определяющие количество повторов предыдущего символа или набора символов.

Основные символы-модификаторы:

- **** — с обратной косой черты начинаются буквенные спецсимволы, также она применяется, если нужно использовать спецсимвол в виде знака препинания (экранирование);
- ***** — 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.

Для работы с регулярными выражениями в bash есть множество инструментов. Наиболее популярные утилиты: grep, SED и AWK.

grep

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), но у него есть модификация egrep, которая позволяет расширенный синтаксис (ERE). Утилита имеет множество параметров, об этом более подробно можно прочитать на страницах справочного руководства man grep. Наиболее используемые опции:

- **grep -ir** — искать заданный шаблон без учёта регистра, рекурсивно включая вложенные каталоги, например `grep -ir error /var/log/*` будет искать строки, содержащие в себе слово error, в каталоге /var/log, включая вложенные подкаталоги;
- **grep -v** исключит из поиска строки, содержащие шаблон, например `cat /var/log/syslog |grep -v named` выведет на экран содержимое файла **/var/log/syslog**, за исключением строк, содержащих в себе слово **named.**

SED

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. В общем случае синтаксис выглядит следующим образом: `sed 's/шаблон/замена/g' file`. Здесь **s** — искать; **шаблон** — то, что ищем; **замена** — то, на что меняем текст; **g** — глобально, то есть во всём файле с именем **file**.

Простой пример: `sed 's/test/text/g' file` найдёт все вхождения слова **test** в файле **file** и заменит на слово **text**, при этом результат работы выведет на экран, не изменяя основного файла. Если мы хотим сразу применить изменения, то следует команде **sed** передать параметр **i** (**sed -i**). Данный параметр позволяет сразу править файл: `sed -i 's/test/text/g' file`.

AWK

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.

В AWK используются переменные трёх типов: числовые (**x=5**), строковые (**x=abc**) и переменные поля, которые обозначаются **$1, $2** и т. д. В отличие от скрипта bash, они означают номера полей, на которые разбита строка.

AWK можно использовать как самостоятельный язык для написания сценариев или вызывать его из командной строки для обработки потока данных. Вызов происходит следующим образом: `поток_данных | awk '{ скрипт_обработки_данных}'`. Здесь **поток_данных** — любая команда ОС или скрипт, результат работы которого будем передавать через **pipe (|)** на обработку AWK.

`'{скрипт_обработки_данных}'` — скрипт, написанный с использованием синтаксиса AWK. Например, `ls -l |awk '{ print $1 }'` выведет на экран первый столбец из вывода команды ls -l.

Crontab и at

Используя скрипты, мы можем автоматизировать множество задач. Чтобы они выполнялись автоматически и с определённой периодичностью, нужен планировщик.

В ОС Linux есть два типа планировщиков: для выполнения разовых задач (например, перезагрузки сервера ночью) используют планировщик **at**, для выполнения задач с определённой периодичностью используют планировщик **cron**.

At

At позволяет планировать выполнение разовых задач в определённое время без редактирования конфигурационных файлов. Это управляющий демон планировщика **atd**. **At** не всегда идёт в списке стандартных пакетов при установке ОС. В случае необходимости его можно установить командой `sudo apt install at -y`. После установки убедитесь, что служба запущена: `sudo systemctl status atd`.

Задача добавляется с использованием следующего синтаксиса: `at -f file_task date_time`. Здесь, используя параметр **-f**, мы говорим **at,** что задание надо прочитать из скрипта с именем **file**.

Data_time — дата и время выполнения скрипта. Также задания можно передавать **at**, используя **pipe** (|).

Пример: `at -f script 11:04am 03/20/2020` — скрипт будет выполнен 20.03.2020 в 11:04. Дата записывается в формате MM/DD/YYYY (месяц/день/год).

С использованием pipe: `rm -rf /opt/test_file| at 11:06am 03/20/2020` — задача на удаление файла test будет выполнена 20.03.2020 в 11:06.

Crontab

Cron — программа-демон, предназначенная для выполнения заданий в определённое время или через определённые промежутки времени. Список заданий, которые будут выполняться автоматически в указанные моменты времени, содержится в файле /etc/crontab и файлах /var/spool/cron. Посмотрим содержимое /etc/crontab:

```
root@vlamp:~# cat /etc/crontab
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab'
# command to install the new version when you edit this file
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs do.
SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
# m h dom mon dow user  command
17 *    * * *   root    cd / && run-parts --report /etc/cron.hourly
25 6    * * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts
--report /etc/cron.daily )
47 6    * * 7   root    test -x /usr/sbin/anacron || ( cd / && run-parts
--report /etc/cron.weekly )
52 6    1 * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts
--report /etc/cron.monthly )
```

Минуты могут принимать значения от 0 до 59, часы — от 0 до 23, дни месяца — от 1 до 31, месяцы — от 1 до 12, дни недели — от 0 (воскресенье) до 6 (суббота). Дальше указываем пользователя (если делаем через утилиту crontab, это не нужно) и саму команду. Обратите внимание на SHELL и PATH. Не всё будет работать так же, как в консоли или скрипте.

Кроме числовых значений, доступны и другие знаки. Например, «*» определяет все допустимые значения. Если на месте всех значений звёздочки, скрипт будет запускаться каждую минуту, каждый день.

Через **запятую** (,) можно указать несколько значений: 1,3,4,7,8.

Тире (-) определяет диапазон значений, например, 1-6, что эквивалентно 1,2,3,4,5,6.

Звёздочка (*) определяет все допустимые значения поля. Например, звёздочка в поле «Часы» будет эквивалентна значению «каждый час».

Слеш (/) может использоваться для пропуска данного числа значений. Например, ***/3** в поле «Часы» эквивалентно строке **0,3,6,9,12,15,18,21**. Звёздочка означает «каждый час», но **/3** диктует использовать только первое, четвёртое, седьмое (и так далее) значения, определённые звёздочкой. Например, каждые полчаса можно задать как ***/30**.

Минимальное время — одна минута. Cron каждую минуту просматривает список заданий и ищет те, которые нужно выполнить. Если требуется совершить действие с интервалом менее одной минуты, можно пойти на хитрость — использовать в команде sleep:

```
*/30 * * * *   user    echo каждые полчаса запускаемся. Время $(date) >>~/mylog
*/30 * * * *   user   sleep 30; echo каждые полчаса через 30 секунд. Время
$(date) >>~/mylog
```

Дни недели и месяцы в трёхбуквенном варианте:

```
sun mon tue wed thu fri sat

jan feb mar apr may jun jul aug sep oct nov dec
```

Дополнительные переменные cron

Переменная	Описание	Эквивалент
@reboot	Запуск при загрузке	
@yearly	Раз в год	0 0 1 1 *
@annually	То же, что и @yearly	
@monthly	Раз в месяц	* * 1* *
@weekly	Раз в неделю	0 0 * * 0
@daily	Раз в день	0 0 * * *
@midnight	В полночь (00:00)	То же, что и @daily
@hourly	Каждый час	0 * * * *

Теперь мы можем поставить скрипт backup в cron:

```
0 0 * * 1-6   user    ~/backup.sh
0 0 * * 0   user    ~/backup.sh --rotate
```

Отдельно стоит сказать о выводе команд. По умолчанию cron отправляет вывод скрипта на почту пользователя, который его запустил. Для любого локального пользователя можно настроить внешний ящик, куда будет отправляться предназначенная ему почта. Эти ящики можно вписать в конфиг /etc/aliases. После его редактирования нужно запустить команду newaliases, настройку подробнее рассмотрим на последнем занятии. Поведение можно изменить, используя директиву MAILTO. Укажем имя пользователя, которому будет отправлено сообщение о выполнении задания:

```
MAILTO=username
```

Вместо имени также можно использовать электронный адрес:

```
MAILTO=example@example.org
```

Пример:

```
 # как обычно, с символа '#' начинаются комментарии
 # в качестве командного интерпретатора использовать /bin/sh
 SHELL=/bin/sh
 # результаты работы отправлять по этому адресу
 MAILTO=paul@example.org
 # добавить в PATH домашний каталог пользователя
 PATH=/bin:/usr/bin:/home/paul/bin
 #### Здесь начинаются задания,

 # выполнять каждый день в 0 часов 5 минут, результат складывать в log/daily
 5 0 * * * $HOME/bin/daily.job >> $HOME/log/daily 2>&1
 # выполнять 1 числа каждого месяца в 14 часов 15 минут
 15 14 1 * * $HOME/bin/monthly
 # каждый рабочий день в 22:00
 0 22 * * 1-5 echo "Пора домой" | mail -s "Уже 22:00" john
 23 */2 * * * echo "Выполняется в 0:23, 2:23, 4:23 и т. д."
 5 4 * * sun echo "Выполняется в 4:05 в воскресенье"
 0 0 1 1 * echo "С Новым годом!"
 15 10,13 * * 1,4 echo "Эта надпись выводится в понедельник и четверг в 10:15 и
13:15"
 0-59 * * * * echo "Выполняется ежеминутно"
 0-59/2 * * * * echo "Выполняется по чётным минутам"
 1-59/2 * * * * echo "Выполняется по нечётным минутам"
 # каждые 5 минут
 */5 * * * * echo "Прошло пять минут"
```

```
 # каждое первое воскресенье каждого месяца. -eq 7 — код дня недели, т.е. 1 -> понедельник , 2 -> вторник и т. д.
 0 1 1-7 * * [ "$(date '+\%u')" -eq 7 ] && echo "Эта надпись выводится каждое первое воскресенье каждого месяца в 1:00"
```

Утилита crontab

Утилита позволяет править файл заданий, вызывая указанный по умолчанию редактор: vi, mcedit, nano. Как и в visudo, правится не /etc/crontab, а пользовательские файлы в /var/spool/cron.

Добавление файла расписания:

```
$ crontab имя_файла_расписания
```

Вывести содержимое текущего файла расписания:

```
$  crontab -l
```

Удаление текущего файла расписания:

```
$ crontab -r
```

Редактирование текущего файла расписания. При первом запуске будет выведен список поддерживаемых текстовых редакторов:

```
$ crontab -e
```

Этот ключ позволяет выполнять вышеописанные действия для конкретного пользователя:

```
# crontab -u username
```

Практическое задание

1. Написать скрипт, который удаляет из текстового файла пустые строки и заменяет маленькие символы на большие. Воспользуйтесь tr или SED.
2. Создать однострочный скрипт, который создаст директории для нескольких годов (2010–2017), в них — поддиректории для месяцев (от 01 до 12), и в каждый из них запишет

несколько файлов с произвольными записями. Например, 001.txt, содержащий текст «Файл 001», 002.txt с текстом «Файл 002» и т. д.

3. Использовать команду AWK на вывод длинного списка каталога, чтобы отобразить только права доступа к файлам. Затем отправить в конвейере этот вывод на sort и uniq, чтобы отфильтровать все повторяющиеся строки.
4. Используя grep, проанализировать файл /var/log/syslog, отобрав события на своё усмотрение.
5. Создать разовое задание на перезагрузку операционной системы, используя at.
6. Написать скрипт, делающий архивную копию каталога etc, и прописать задание в crontab.

Дополнительные материалы

Регулярные выражения

SED

AWK

Используемые источники

Advanced Bash-Scripting Guide

Костромин В. Linux для пользователя

Регулярные выражения в Linux

Cron

at