Постнов Александр Вячеславович М8О-101Б-21

Зачет № 1

**Вопрос № 1. Утилиты UNIX**

Варианты по формуле:

(N + 4 + 3 \* i) % 39 i=[0, 1, …, 9] N = 17 (2 можно выбросить - красный цвет)

***21 24*** *27* ***30******33*** *36* ***1******3******6 9***

**21) ed**

ed ( /iːdiː/ ) — строчный текстовый редактор из 1969 года, когда терминалы были медленными, ОЗУ была дорогой и её было мало, а экраны были маленькими. ed стандартизирован по стандартам POSIX и Open Group (X/Open) и встречается практически в каждой UNIX или UNIX-like системе. Знание этого инструмента может понадобиться не только для написания скриптов или работы с дистрибутивами, куда редактор vi не включен по умолчанию, но он также может оказаться достаточно простым и удобным для решения повседневных задач.

В отличие от остальных редакторов ed не создаёт копию файла, в которой выполняет редактирование с последующей заменой файла, а работает с оригинальным файлом. Это может быть полезным, когда необходимо изменить файл при этом не изменяя его inode number, но буфер становится ограничен адресным пространством процесса.

**Команды ed и их комбинации**

| **P** | **Отобразить приглашение командной строки** |
| --- | --- |
| **a** | **Добавить строки после текущей** |
| **p** | **Напечатать текущую строку** |
| **pn** | **Напечатать текущую строку с номером** |
| **,p** | **Напечатать содержимое всего буфера** |
| **,pn** | **Напечатать содержимое всего буфера с номерами строк** |
| **w <имя\_файла>** | **Записать содержимое буфера в файл** |
| **q** | **Выйти из редактора** |
| **u** | **Отменить последнее действие** |
| **!<команда>** | **Запустить программу команда** |

**24) bzip2**

**Команда bzip2 предназначена для сжатия данных без потерь с помощью соответствующей утилиты, использующей** [**алгоритм Барроуза-Уилера**](https://ru.wikipedia.org/wiki/Bzip2)**. Целью использования данной утилиты является экономия дискового пространства. Упомянутый алгоритм позволяет достичь лучшей степени сжатия данных, чем тот, который реализован в рамках утилит gzip и zip, но зачастую худшей степени сжатия данных, чем тот, который реализован в рамках утилиты xz. Кроме того, на уровне декомпресии данных он является более ресурсоемким, чем алгоритм, который реализован в рамках утилиты xz. Последнее обстоятельство обуславливает переход пальмы первенства от утилиты bzip2 к утилите xz, но рассматриваемая утилита все еще активно используется и в некоторых случаях позволяет сэкономить больше дискового пространства, чем утилита xz.**

**Базовый синтаксис команды выглядит следующим образом:**

**$ bzip2 [параметры] <имена файлов>**

**Чаще всего bzip2 используется вообще без каких-либо параметров, причем в качестве аргументов может передаваться неограниченное количество имен файлов, которые следует сжать. По умолчанию оригинальные версии файлов заменяются на их сжатые версии с соответствующими метаданными (то есть, меткой времени модификации, правами доступа, именами владельца и группы владельцев и так далее). Если вас не устраивает такое положение дел, вы можете воспользоваться параметром -k для сохранения оригинальных версий файлов. Кроме того, вы можете использовать параметр -s для того, чтобы сэкономить оперативную память, но приготовьтесь к длительному ожиданию, ведь в этом случае скорость обработки данных снизится практически вдвое. Параметры из диапазона от -1 до -9 позволяют задать степень сжатия (от самой низкой до самой высокой соответственно), при этом чем выше степень сжатия, тем больше системных ресурсов требуется утилите. Параметр -t предназначен для тестирования целостности сжатого файла и не оказывает какого-либо влияния на него (следует комбинировать его с параметром -v для подробного вывода). Параметр -d позволяет восстановить оригинальные версии файлов с переданными именами на основе их сжатых версий.**

**Если же вам нужно создать архив с несколькими файлами внутри, одной утилиты bzip2 будет явно мало. Для этой цели также понадобится утилита tar, с помощью которой можно создать архив с файлами, после чего сжать этот архив с помощью утилиты bzip2. Например, вы можете использовать следующую последовательность команд для создания архива с именем archive.tar.bz2:**

**$ tar -cf archive.tar <имена файлов>**

**$ bzip2 archive.tar**

**Параметры -c и -f утилиты tar предназначены для указания на необходимость добавления всех файлов в один архив (-c) и чтения имени файла архива из следующего аргумента (-f). Альтернативным вариантом является замена последней команды на параметр -j утилиты tar, позволяющий автоматически сжать полученный архив с помощью bzip2:**

**$ tar -cfj archive.tar.bz2 <имена файлов>**

**30) file**

**Команда file — одна из самых полезных, поскольку позволяет узнать тип данных, которые на самом деле содержатся внутри документа.**

Синтаксис команды file достаточно простой. Записывать её в эмуляторе терминала или консоли следует так:

file опции название\_документа

Что же касается опций, то их у этой команды несколько десятков. Мы рассмотрим лишь основные:

-b, --brief — запрет на демонстрацию имен и адресов файлов в выводе команды;

-i, --mime — определение MIME-типа документа по его заголовку;

--mime-type, --mime-encoding — определение конкретного элемента MIME;

-f, --files-from — анализ документов, адреса которых указаны в простом текстовом файле;

-l, --list — список паттернов и их длина;

-s, --special-files — предотвращение проблем, которые могут возникнуть при чтении утилитой специальных файлов;

-P — анализ определенной части файла, которая обозначается различными параметрами;

-r, --raw — отказ от вывода /ooo вместо непечатных символов;

-z — анализ содержимого сжатых документов.

Для того, чтобы ознакомиться с полным списком опций, выполните в терминале команду:

**33) xargs**

Возможность объединения нескольких команд Linux в терминале и использования их в качестве конвейера, когда каждая следующая команда получает вывод предыдущей - очень мощный и гибкий инструмент. Но команды можно объединять не только так. С помощью утилиты xargs вывод предыдущей команды можно передать в аргументы следующей.

интаксис команды немного запутанный, но в нём можно разобраться:

$ первая\_команда | xargs опции вторая\_команда аргументы

Сначала выполняется любая первая команда и весь её вывод по туннелю передается в xargs. Затем этот вывод разбивается на строки и для каждой строки вызывается вторая команда, а полученная строка передаётся ей в аргументах.

Для настройки поведения утилиты xargs можно использовать опции. Давайте их рассмотрим:

-0, --null - использовать в качестве разделителя нулевой символ. Обычно он находится в конце строки. По умолчанию, в качестве разделителя используется пробел, перевод строки или табуляция;

-a, --arg-file - прочитать аргументы, которые надо передать команде из файла;

-d, --delimiter - использовать нестандартный разделитель строк;

-E, -e, --eof - индикатор конца файла, все символы после вхождения этой строки игнорируются;

-l, --max-lines - количество строк, передающихся в одну команду по умолчанию все;

-n, --max-args - количество параметров, которые передаются в одну команду, по умолчанию все;

-o, --open-tty - открывать новый терминал для дочерних процессов;

-p, --interactive - спрашивать пользователя о запуске каждой следующей команды;

-r, --no-run-if-empty - если входной поток данных пустой, команда не будет выполнена;

--show-limits - посмотреть ограничения на длину параметров в командной строке;

-t, --verbose - максимально подробный вывод утилиты.

**1) cmp**

**Команда cmp позволяет использовать утилиту для побайтового сравнения двух произвольных файлов. Она может использоваться как для сравнения бинарных, так и для сравнения текстовых файлов. В случае текстовых файлов для улучшения информативности вывода могут использоваться дополнительные параметры.**

**Базовый синтаксис команды выглядит следующим образом:**

**$ cmp [параметры] <первый файл> <второй файл>**

**По умолчанию утилита выводит информацию о позиции первого отличающего байта, либо сообщение о том, что файлы являются идентичными. При обработке текстовых файлов следует использовать параметр -c, активирующий режим обработки байтов как текстовых символов. Для вывода позиций и значений всех отличающихся байтов в первом и во втором файле предназначен параметр -l, причем вместе с ним может использоваться параметр -n с числовым значением, соответствующим максимальному количеству искомых различий.**

**Если необходимо пропустить заданное количество байт в начале каждого из файлов, может использоваться параметр -i, требующий обязательного указания количества пропускаемых байтов в обоих файлах, либо в каждом из файлов по отдельности, но в этом случае значения должны отделяться с помощью символа двоеточия. Для использования утилиты из сценариев без вывода каких-либо сообщений может использоваться параметр -s, причем результат сравнения может быть получен из кода завершения работы утилиты (0 — файлы идентичны, 1 — файлы отличаются, 2 — ошибка доступа или недопустимый аргумент).**

### Примеры использования

**В качестве тестовых файлов для сравнения будут использоваться текстовые файлы со следующим содержимым:**

**$ cat file1.txt**

**Linux FAQ cmp command tutorial**

**$ cat file2.txt**

**linux faq man command tutorial**

#### Поиск первого различия в файлах

**Для поиска первого отличающегося байта в двух файлах не нужно использовать каких-либо параметров:**

**$ cmp file1.txt file2.txt**

**file1.txt file2.txt различаются: байт 1, строка 1**

**В случае обработки текстовых файлов рекомендуется использовать параметр -c; благодаря ему утилита выведет не только позицию отличающегося байта, но и его значения в обоих файлах, а также соответствующие им символы:**

**$ cmp -c file1.txt file2.txt**

**file1.txt file2.txt различаются: байт 1, строка 1 равен 114 L 154 l**

#### Поиск всех различий в файлах

**Для поиска всех различий в файлах следует использовать параметр -l. Вы увидите таблицу в первом столбце которой будет находится позиционный номер отличающегося байта, во втором — его значение из первого файла, а в третьем — его значение из второго файла. Другими словами, без параметра -c утилита будет выводить лишь значения отличающихся байтов:**

**$ cmp -l file1.txt file2.txt**

**1 114 154**

**7 106 146**

**8 101 141**

**9 121 161**

**11 143 155**

**12 155 141**

**13 160 156**

**Параметр -c в данном случае будет действовать аналогичным образом, причем немного изменится формат вывода: в первом столбце таблицы будет находиться позиционный номер отличающегося байта, во втором и третьем — его значение и соответствующий символ из первого файла, а в четвертом и пятом — его значение и соответствующий символ из второго файла:**

**$ cmp -l -c file1.txt file2.txt**

**1 114 L 154 l**

**7 106 F 146 f**

**8 101 A 141 a**

**9 121 Q 161 q**

**11 143 c 155 m**

**12 155 m 141 a**

**13 160 p 156 n**

#### Поиск всех различий файлов с пропуском начальных байтов

**Для осуществления поиска различий с определенных позиций используется параметр -i. В данном случае осуществляется пропуск 10 байтов с начала каждого из файлов:**

**$ cmp -l -c -i 10:10 file1.txt file2.txt**

**1 143 c 155 m**

**2 155 m 141 a**

**3 160 p 156 n**

**3) wc**

Анализ файлов - неотъемлемая часть работы с ними. Иногда возникает необходимость подсчитать количество строк или слов в тексте. С этой задачей эффективно справляется команда wc Linux.

## СИНТАКСИС КОМАНДЫ WC

Для запуска утилиты откройте терминал и введите:

wc

Терминал будет ожидать ввода данных. После нажатия комбинации клавиш **Ctrl** + **D** командный интерпретатор завершит работу программы и выведет три числа, обозначающих количество строк, слов и байт введённой информации.

Утилита может обрабатывать файлы. Стандартная инструкция выглядит так:

wc file

* **wc** — имя утилиты;
* **file** — название обрабатываемого файла.

Программа также может принимать параметры для анализа отдельных значений. Наиболее используемые из них приведены в таблице ниже:

| Параметр | Длинный вариант | Значение |
| --- | --- | --- |
| -c | --bytes | Отобразить размер объекта в байтах |
| -m | --count | Показать количесто символов в объекте |
| -l | --lines | Вывести количество строк в объекте |
| -w | --words | Отобразить количество слов в объекте |

**6) grep**

Иногда может понадобится найти файл, в котором содержится определённая строка или найти строку в файле, где есть нужное слово. В Linux всё это делается с помощью одной очень простой, но в то же время мощной утилиты grep. С её помощью можно искать не только строки в файлах, но и фильтровать вывод команд, и много чего ещё.

ЧТО ТАКОЕ GREP?

Команда grep (расшифровывается как global regular expression print) - одна из самых востребованных команд в терминале Linux, которая входит в состав проекта GNU. Секрет популярности - её мощь, она даёт возможность пользователям сортировать и фильтровать текст на основе сложных правил.

Утилита grep решает множество задач, в основном она используется для поиска строк, соответствующих строке в тексте или содержимому файлов. Также она может находить по шаблону или регулярным выражениям. Команда в считанные секунды найдёт файл с нужной строчкой, текст в файле или отфильтрует из вывода только пару нужных строк. А теперь давайте рассмотрим, как ей пользоваться.

СИНТАКСИС GREP

Синтаксис команды выглядит следующим образом:

$ grep [опции] шаблон [имя файла...]

Или:

$ команда | grep [опции] шаблон

Опции - это дополнительные параметры, с помощью которых указываются различные настройки поиска и вывода, например количество строк или режим инверсии.

Шаблон - это любая строка или регулярное выражение, по которому будет вестись поиск

Файл и команда - это то место, где будет вестись поиск. Как вы увидите дальше, grep позволяет искать в нескольких файлах и даже в каталоге, используя рекурсивный режим.

Возможность фильтровать стандартный вывод пригодится,например, когда нужно выбрать только ошибки из логов или найти PID процесса в многочисленном отчёте утилиты ps.

ОПЦИИ

Давайте рассмотрим самые основные опции утилиты, которые помогут более эффективно выполнять поиск текста в файлах grep:

-b - показывать номер блока перед строкой;

-c - подсчитать количество вхождений шаблона;

-h - не выводить имя файла в результатах поиска внутри файлов Linux;

-i - не учитывать регистр;

- l - отобразить только имена файлов, в которых найден шаблон;

-n - показывать номер строки в файле;

-s - не показывать сообщения об ошибках;

-v - инвертировать поиск, выдавать все строки кроме тех, что содержат шаблон;

-w - искать шаблон как слово, окружённое пробелами;

-e - использовать регулярные выражения при поиске;

-An - показать вхождение и n строк до него;

-Bn - показать вхождение и n строк после него;

-Cn - показать n строк до и после вхождения;

Все самые основные опции рассмотрели и даже больше, теперь перейдём к примерам работы команды grep Linux.

**9) tail**

## КОМАНДА TAIL В LINUX

**$ tail опции файл**

**По умолчанию утилита выводит десять последних строк из файла, но ее поведение можно настроить с помощью опций:**

* **-c - выводить указанное количество байт с конца файла;**
* **-f - обновлять информацию по мере появления новых строк в файле;**
* **-n - выводить указанное количество строк из конца файла;**
* **--pid - используется с опцией -f, позволяет завершить работу утилиты, когда завершится указанный процесс;**
* **-q - не выводить имена файлов;**
* **--retry - повторять попытки открыть файл, если он недоступен;**
* **-v - выводить подробную информацию о файле;**

**В качестве значения параметра -c можно использовать число с приставкой b, kB, K, MB, M, GB, G T, P, E, Z, Y. Еще есть одно замечание по поводу имен файлов. По умолчанию утилита не отслеживает изменение имен, но вы можете указать что нужно отслеживать файл по дескриптору, подробнее в примерах.**

**Вопрос № 2. Командный язык UNIX(bash)**

Вариант по формуле:

(N + 3) % 10 + 1 = (17 + 3) % 10 + 1 = 1

Вопрос: переменные и параметры скрипта

**ПАРАМЕТРЫ СКРИПТА**

Наиболее распространённый способ передачи данных сценариям заключается в использовании параметров командной строки. Вызвав сценарий с параметрами, мы передаём ему некую информацию, с которой он может работать. Выглядит это так:

$ ./myscript 10 20

В данном примере сценарию передано два параметра — «10» и «20». Всё это хорошо, но как прочесть данные в скрипте?

## Чтение параметров командной строки

Оболочка bash назначает специальным переменным, называемым позиционными параметрами, введённые при вызове скрипта параметры командной строки:

* $0 — имя скрипта.
* $1 — первый параметр.
* $2 — второй параметр — и так далее, вплоть до переменной $9, в которую попадает девятый параметр.

Вот как можно использовать параметры командной строки в скрипте с помощью этих переменных:

#!/bin/bash

echo $0

echo $1

echo $2

echo $3

Взглянем на ещё один пример использования параметров. Тут мы найдём сумму чисел, переданных сценарию:

#!/bin/bash

total=$[ $1 + $2 ]

echo The first parameter is $1.

echo The second parameter is $2.

echo The sum is $total.

## Проверка параметров

Если скрипт вызван без параметров, но для нормальной работы кода предполагается их наличие, возникнет ошибка. Поэтому рекомендуется всегда проверять наличие параметров, переданных сценарию при вызове. Например, это можно организовать так:

#!/bin/bash

**if** [ -n "$1" ]

**then**

echo Hello $1.

**else**

echo "No parameters found. "

**fi**

## Команда shift

Использовать команду shift в bash-скриптах следует с осторожностью, так как она, в прямом смысле слова, сдвигает значения позиционных параметров.

Когда вы используете эту команду, она, по умолчанию, сдвигает значения позиционных параметров влево. Например, значение переменной $3 становится значением переменной $2, значение $2 переходит в $1, а то, что было до этого в $1, теряется. Обратите внимание на то, что при этом значение переменной $0, содержащей имя скрипта, не меняется.

Воспользовавшись командой shift, рассмотрим ещё один способ перебора переданных скрипту параметров:

#!/bin/bash

count=1

**while** [ -n "$1" ]

**do**

echo "Parameter #$count = $1"

count=$(( $count + 1 ))

shift

**done**

Скрипт задействует цикл while, проверяя длину значения первого параметра. Когда длина станет равна нулю, происходит выход из цикла. После проверки первого параметра и вывода его на экран, вызывается команда shift, которая сдвигает значения параметров на одну позицию.

**ПЕРЕМЕННЫЕ В BASH**

# **Переменные в bash**

Переменная в языке shell - это макрос, который может быть подставлен в строку команды перед её разбором. Оператор ${VAR} заменяется текстовой строкой, хранящейся в переменной с именем VAR. После подстановки значение переменной будет разбито на отдельные слова по пробельным символам. Если содержимое переменной должно интерпретироваться как одно слово, то оператор подстановки надо взять в кавычки "${VAR}"

Интерпретатор `bash` использует как минимум три типа переменных, отличающихся областью видимости:

A=BCD # Локальная переменная, доступная из данного скрипта

A=BCD cmd # Именованный параметр, передаваемый в окружение программы cmd не влияющий на текущий скрипт

export A=BCD # Глобальная переменная, видимая в текущем скрипте и в запускаемых программах

Переменные в bash не имеют типа. Можно считать, что переменная всегда содержит текстовую строку. Если требуется интерпретация переменной как числа, то проводится разбор строки слева. Значение определяют цифры до первого нечислового символа. Строка не содержащая цифр интерпретируется как 0.

set – выдает список всех переменных с их значениями

VAR="string" - определяет локальную переменную интерпретатора и присваивает ей значение. Разделителями в команде являются символ = после имени переменной и первый пробел после значения. Пробелы вокруг знака равенства недопустимы. Пробел слева от знака равенства интерпретируется как часть имени, пробел справа – как пустая присваеваемая строка. Если строка содержит пробелы, то она должна быть заключена в кавычки.

$VAR – подстановка значения переменной VAR. При разборе строки именем переменной считается максимальная последовательность из букв, цифр и знаков подчеркивания.

${VAR} - то же что и $VAR позволяет избавиться от неоднозначности при разборе строки

export VAR делает переменную доступной для дочерних процессов (делает переменную глобальной)

export VAR=string допустимое объявление глобальной переменной

unset VAR уничтожает определение переменной