

Цель работы: практическое знакомство со способами эффективной обработки текста при помощи интерфейса командной строки и набора стандартных утилит.

Ход работы:

1. Ознакомиться с теоретическим материалом.
2. Выполнить задания.
3. Ответить на контрольные вопросы.

Задание:

1. Используя утилиты `hexdump` и `strings`, вывести на экран содержимое одного из перечисленных ниже файлов из каталога `/bin`. Позиция файла для распечатки определяется номером бригады. Имена файлов для выполнения задания 1: `tar`, `sort`, `sed`, `ping`, `vi`, **`unlink`**, `uname`, `touch`, `sleep`, `sty`.

2. Подсчитать общее количество файлов (каталогов) в одном из перечисленных ниже каталогов. Каталог для подсчета количества определяется номером бригады. Имена каталогов для выполнения задания 2: `/bin`, `/etc`, `/lib`, `/proc`, `/usr`, **`/var`**, `/dev`, `/sbin`, `/sys`, `/root`.

3. Найти общее количество процессов, выполняющихся в системе в данный момент.

4. Вывести список выполняющихся процессов, в именах которых присутствует слово `manager` и отсутствует слово `grep`.

5. Создать текстовый файл, содержащий набор строк вида:

```
123
178
176
755
713
873
```

С помощью утилиты `grep` найти строки, в которых есть цифра 7, после которой находится одна из цифр — 1, 3 или 5.

6. Создать текстовый файл, содержащий набор строк вида:

```
starfish
samscripтер
stellar
```

microsrar
ascender
sacrifice
scalar

С помощью утилиты `grep` найти строки, начинающиеся на букву `s` и заканчивающиеся на букву `r`.

7. Создать текстовый файл, содержащий простейшие адреса электронной почты вида `username@website.com`.

С помощью утилиты `grep` найти строки, содержащие правильные простейшие адреса. Проверить возможность использования более сложного регулярного выражения для распознавания адресов, содержащих другие допустимые символы.

8. На произвольном примере продемонстрировать работу утилиты `tr`.

9. Создать текстовый файл, содержащий допустимые и недопустимые IP-адреса, например

127.0.0.1
255.255.255.255
12.34.56
123.256.0.0
1.23.099.255
0.79.378.111

С помощью утилиты `grep` и руководства `man` найти строки, содержащие допустимые четырехбайтовые IP адреса.

10. Создать текстовый файл, содержащий корректные и некорректные номера телефонов ведомственной АТС объемом 399 номеров, номера с 000 до 399 – корректные, 0, 400, 900 – некорректные. С помощью утилиты `grep` и руководства `man` найти строки, содержащие допустимые четырехбайтовые IP адреса.

.

Описание выполнения работы

1. Используя утилиты `hexdump` и `strings`, выведем на экран содержимое файла `unlink` из каталога `/bin`, как показано рисунках 6.1, 6.2.

```
student@student-VirtualBox:~$ hexdump -C /bin/touch
00000000  7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00 |.ELF.....|
00000010  03 00 3e 00 01 00 00 00 70 42 00 00 00 00 00 00 |..>.....pB....|
00000020  40 00 00 00 00 00 00 00 f8 81 01 00 00 00 00 00 |@.....|
00000030  00 00 00 00 40 00 38 00 0d 00 40 00 1e 00 1d 00 |...@.8...@....|
00000040  06 00 00 00 04 00 00 00 40 00 00 00 00 00 00 00 |.....@.....|
00000050  40 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 |@.....@.....|
00000060  d8 02 00 00 00 00 00 00 d8 02 00 00 00 00 00 00 |.....|
00000070  08 00 00 00 00 00 00 00 03 00 00 00 04 00 00 00 |.....|
00000080  18 03 00 00 00 00 00 00 18 03 00 00 00 00 00 00 |.....|
00000090  18 03 00 00 00 00 00 00 1c 00 00 00 00 00 00 00 |.....|
000000a0  1c 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00 |.....|
000000b0  01 00 00 00 04 00 00 00 00 00 00 00 00 00 00 00 |.....|
000000c0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
000000d0  c0 29 00 00 00 00 00 00 c0 29 00 00 00 00 00 00 |.).....).|
000000e0  00 10 00 00 00 00 00 00 01 00 00 00 05 00 00 00 |.....|
000000f0  00 30 00 00 00 00 00 00 00 30 00 00 00 00 00 00 |.0.....0.....|
00000100  00 30 00 00 00 00 00 00 61 e0 00 00 00 00 00 00 |.0.....a.....|
00000110  61 e0 00 00 00 00 00 00 00 10 00 00 00 00 00 00 |a.....|
00000120  01 00 00 00 04 00 00 00 00 20 01 00 00 00 00 00 |.....|
00000130  00 20 01 00 00 00 00 00 00 20 01 00 00 00 00 00 |.....|
```

Рисунок 6.1 — Фрагмент результата использования утилиты `hexdump`

```
student@student-VirtualBox:~$ strings -n3 /bin/uname
ELF
/lib64/ld-linux-x86-64.so.2
GNU
GNU
GNU
V'I
MB#F-
,cr
libc.so.6
fflush
__printf_chk
setlocale
mbtowc
fopen
strncmp
optind
strchr
dcgettext
error
__stack_chk_fail
fgets_unlocked
iswprint
realloc
abort
_exit
program_invocation_name
__ctype_get_mb_cur_max
calloc
strlen
memset
strstr
__errno_location
```

Рисунок 6.2 — Результат использования утилиты `strings`

2. Подсчитаем общее количество файлов (каталогов) в каталоге /var с помощью утилиты find с ключами wc -l, как показано на рисунке 6.3.

```
student@student-VirtualBox:~$ find . /var | wc -l
find: '/var/cache/apparmor/26b63962.0': Отказано в доступе
find: '/var/cache/ldconfig': Отказано в доступе
find: '/var/cache/system-tools-backends/backup': Отказано в доступе
find: '/var/cache/apt/archives/partial': Отказано в доступе
find: '/var/cache/private': Отказано в доступе
find: '/var/cache/cups': Отказано в доступе
find: '/var/lib/NetworkManager': Отказано в доступе
find: '/var/lib/AccountsService/users': Отказано в доступе
find: '/var/lib/fwupd/gnupg': Отказано в доступе
find: '/var/lib/snapd/void': Отказано в доступе
find: '/var/lib/snapd/cookie': Отказано в доступе
find: '/var/lib/colord/.cache': Отказано в доступе
find: '/var/lib/gdm3/.cache/libgweather': Отказано в доступе
find: '/var/lib/gdm3/.local/share/keyrings': Отказано в доступе
find: '/var/lib/gdm3/.local/share/gnome-shell': Отказано в доступе
find: '/var/lib/gdm3/.local/share/gvfs-metadata': Отказано в доступе
```

Рисунок 6.3 – Результат подсчёта общего количества файлов (каталогов) в каталоге /var

3. Найдём общее количество процессов, выполняющихся в системе в данный момент с помощью команды ps, выводящей отчёт о работающих процессах, где ключ a показывает процессы для всех пользователей, ключ u показывает владельца процесса, ключ u показывает процессы, не подключенные к терминалу, как показано рисунке 6.4.

```
student@student-VirtualBox:~$ ps aux | grep bash | wc -l
2
student@student-VirtualBox:~$ ps aux | grep bash
student      2189  0.0  0.2 19240  5080 pts/0    Ss   13:56   0:00  bash
student      3330  0.0  0.0 17688   724 pts/0    S+   14:03   0:00  grep --color=auto  bash
student@student-VirtualBox:~$
```

Рисунок 6.4 — Результат подсчёта общего количества процессов, выполняющихся в системе в данный момент

4. Выведем список выполняющихся процессов, в именах которых присутствует слово manager и отсутствует слово grep, как показано рисунке 6.5.

```
student@student-VirtualBox:~$ ps aux | grep bash | grep manager | grep -v grep
```

Рисунок 6.5 — Вывод списка выполняющихся процессов, в именах которых присутствует слово manager и отсутствует слово grep

5. Создадим текстовый файл, содержащий набор строк вида:

```
123
178
176
755
713
873
```

Создание и содержание текстового файла с помощью утилиты printf, как показано рисунке 6.6, 6.7.

```
student@student-VirtualBox:~$ printf "123
> 178
> 176
> 755
> 713
> 873
> " > 5.txt
student@student-VirtualBox:~$
```

Рисунок 6.6 — Создание текстового файла



The screenshot shows a text editor window titled "5.txt". The content of the file is as follows:

```
1 123
2 178
3 176
4 755
5 713
6 873
```

Рисунок 6.7 — Содержание текстового файла

С помощью утилиты grep найдём строки, в которых есть цифра 7, после которой находится одна из цифр — 1, 3 или 5, как показано рисунке 6.8.

```
student@student-VirtualBox:~$ cat /home/student/5.txt | grep [7][135]
755
713
873
student@student-VirtualBox:~$
```

Рисунок 6.8 — Вывод строк с помощью утилиты `grep`, в которых есть цифра 7, после которой находится одна из цифр — 1, 3 или 5

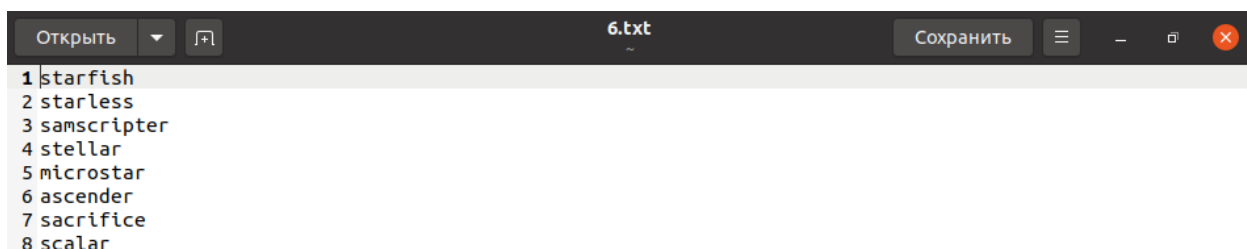
6. Создадим текстовый файл, содержащий набор строк вида:

```
Starfish
Starless
Samscripтер
Stellar
Microsrar
Ascender
sacrificescalar
```

Создание и содержание текстового файла с помощью утилиты описанной выше в пункте 5, как показано рисунке 6.9 – 6.10.

```
student@student-VirtualBox:~$ printf "starfish
> starless
> samscripтер
> stellar
> microstar
> ascender
> sacrifice
> scalar
> " > 6.txt
student@student-VirtualBox:~$
```

Рисунок 6.9 — Создание текстового файла



```
1 starfish
2 starless
3 samscripтер
4 stellar
5 microstar
6 ascender
7 sacrifice
8 scalar
```

Рисунок 6.10 — Содержание текстового файла

С помощью команды `sort` отсортируем строки заданного файла при помощи утилиты `grep`, с её помощью найдём строки, начинающиеся на букву `s` и заканчивающиеся на букву `r`, как показано рисунке 6.11.

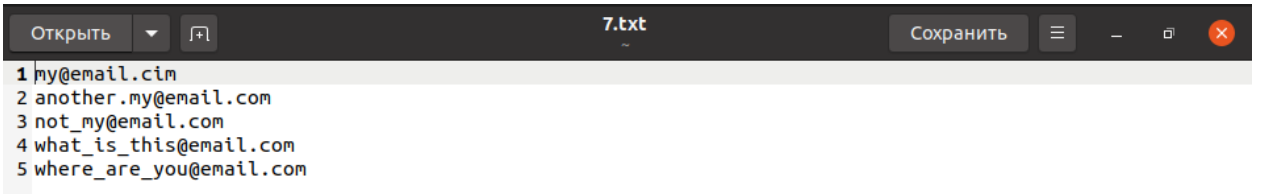
```
student@student-VirtualBox:~$ sort /home/student/6.txt | grep '\b[s]\w*[r]\b'
samscripтер
scalar
stellar
student@student-VirtualBox:~$
```

Рисунок 6.11 — Нахождение с помощью утилиты `grep` строк, начинающиеся на букву `s` и заканчивающихся на букву `r`

7. Создадим текстовый файл, содержащий простейшие адреса электронной почты вида `username@website.com`, описанным в пункте 5 методом, как показано рисунках 6.12, 6.13.

```
student@student-VirtualBox:~$ printf 'my@email.cim
> another.my@email.com
> not_my@email.com
> what_is_this@email.com
> where_are_you@email.com
> ' > 7.txt
student@student-VirtualBox:~$
```

Рисунок 6.12 — Создание текстового файла



```
1 my@email.cim
2 another.my@email.com
3 not_my@email.com
4 what_is_this@email.com
5 where_are_you@email.com
```

Рисунок 6.13 — Содержание текстового файла

С помощью утилиты `grep` найдём строки, содержащие правильные простейшие адреса. Проверим возможность использования более сложного регулярного выражения для распознавания адресов, содержащих другие допустимые символы рисунок 6.14.

```
student@student-VirtualBox:~$ grep -E "(\w+\.)*\w+@(\w+\.)+[A-Za-z]+" 7.txt
my@email.cim
another.my@email.com
not_my@email.com
what_is_this@email.com
where_are_you@email.com
student@student-VirtualBox:~$
```

Рисунок 6.14 — Нахождение правильных простейших адресов

8 Для замены одних символов другими предназначена утилита `tr`. Продемонстрируем её работу на произвольном примере, как показано рисунке 6.15.

```
student@student-VirtualBox:~$ printf '01234
' > 10.txt
student@student-VirtualBox:~$ cat /home/student/10.txt | tr 01234 56789 | head -4
56789
student@student-VirtualBox:~$
```

Рисунок 6.15 — Утилита `tr`

9. Создадим текстовый файл, содержащий допустимые и недопустимые IP-адреса, например

```
127.0.0.1
255.255.255.255
12.34.56
123.256.0.0
1.23.099.255
0.79.378.111
```

Создание и содержание текстового файла, как показано на рисунке 6.16.

```
student@student-VirtualBox:~$ printf '127.0.0.1
> 255.255.255.0
> 22.33.44
> 1.22.081.32
> 0.79.333.222
> ' > 8.txt
student@student-VirtualBox:~$
```

Рисунок 6.16 — Создание текстового файла

С помощью утилиты `grep` и ключей `-E -o` и руководства `man` найдём строки, содержащие допустимые четырехбайтовые IP-адреса, как показано рисунке 6.17.

```
student@student-VirtualBox:~$ grep -E -o "(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)" 8.txt
127.0.0.1
255.255.255.0
1.22.081.32
student@student-VirtualBox:~$
```

Рисунок 6.17 — Нахождение правильных IP-адресов

10 Создадим текстовый файл, содержащий корректные и некорректные номера телефонов ведомственной АТС объемом 399 номеров, номера с 000 до 399 – корректные, 0, 400, 900 – некорректные, как показано рисунке 6.18.

```
390
391
392
393
394
395
396
396
397
398
399
400
900
' > 9.txt
```

Рисунок 6.18 — Создание текстового файла

С помощью утилиты `grep` и руководства `man` найдём строки, содержащие допустимые номера телефонов, как показано рисунке 6.19.

```
student@student-VirtualBox:~$ grep -E "[0-3][0-9][0-9]" 9.txt
397
398
399
student@student-VirtualBox:~$
```

Рисунок 6.19 — Нахождение правильных номеров телефонов

Контрольные вопросы

1. Вывод на экран содержимого нетекстового файла с помощью утилит `hexdump` и `strings`.

На терминал Linux может быть выведен нетекстовый файл, однако пользы в этом будет мало. Во-первых, раз уж файл содержит не текст, то не предполагается, что пользователь сможет что-либо понять из его содержимого. Во-вторых, если в нетекстовых данных, выводимых на терминал, случайно встретится управляющая последовательность, терминал ее выполнит.

Если содержимое нетекстового файла все-таки желательно просмотреть (то есть превратить в текст), можно воспользоваться утилитой `hexdump` с ключом `-C`, которая выдает содержимое файла в виде шестнадцатеричных ASCII-кодов, или с помощью утилиты `strings`, показывающей только те части файла, которые могут быть представлены в виде текста.

2. Стандартный ввод, вывод, стандартный вывод ошибок.

Каждый процесс Linux получает при старте три дескриптора, открытых для него системой. Первый из них (дескриптор 0) открыт на чтение, это стандартный ввод процесса. Именно со стандартным вводом работают все операции чтения, если в них не указан дескриптор файла. Второй (дескриптор 1) – открыт на запись, это стандартный вывод процесса. С ним работают все операции записи, если дескриптор файла не указан в них явно. Наконец, третий поток данных (дескриптор 2) предназначается для вывода диагностических сообщений, он называется стандартный вывод ошибок.

3. Конвейер и канал.

Нередко возникают ситуации, когда нужно обработать вывод одной программы какой-то другой программой. Для решения подобной задачи в `bash` предусмотрена возможность перенаправления вывода можно не только в

файл, но и непосредственно на стандартный ввод другой программе. В Linux такой способ передачи данных называется конвейер.

В `bash` для перенаправления стандартного вывода на стандартный ввод другой программе служит символ «`|`». Можно последовательно обработать данные несколькими разными программами, перенаправляя вывод на ввод следующей программе и организовав сколь угодно длинный конвейер для обработки данных. В результате получаются командные строки вида «`cmd1 | cmd2 | ... | cmdN`».

Организация конвейера устроена в `shell` по той же схеме, что и перенаправление в файл, но с использованием особого объекта системы - канала. Можно представить трубу, немедленно доставляющую данные от входа к выходу (английский термин – «`pipe`»). Каналом пользуются сразу два процесса: один пишет туда, другой читает. Связывая две команды конвейером, `shell` открывает канал (заводится два дескриптора – входной и выходной), подменяет стандартный вывод первого процесса на входной дескриптор канала, а стандартный ввод второго процесса - на выходной дескриптор канала. После чего остается запустить по команде в этих процессах, и стандартный вывод первой попадет на стандартный ввод второй.

Канал (`pipe`) – неделимая пара дескрипторов (входной и выходной), связанных друг с другом таким образом, что данные, записанные во входной дескриптор, будут немедленно доступны на чтение с выходного дескриптора.

4. Фильтры.

Если программа и вводит данные, и выводит, то ее можно рассматривать как трубу, в которую что-то входит и из которой что-то выходит. Обычно смысл работы таких программ заключается в том, чтобы определенным образом обработать поступившие данные. В Linux такие программы называют фильтрами: данные проходят через них, причем что-то «застревает» в фильтре и не появляется на выходе, а что-то изменяется, что-то проходит сквозь фильтр неизменным. Фильтры в Linux обычно по умолчанию читают

данные со стандартного ввода, а выводят на стандартный вывод. Простейший фильтр - программа `cat`.

5. Структурные единицы текста. Подсчет количества единиц текста.

Работая с текстом в Linux, нужно принимать во внимание, что текстовые данные, передаваемые в системе, структурированы. Большинство утилит обрабатывает не непрерывный поток текста, а последовательность единиц. В текстовых данных в Linux выделяются следующие структурные единицы:

- Строки
- Поля
- Символы

6. Элементарные регулярные выражения.

Регулярными выражениями называются особым образом составленные наборы символов, выделяющие из текста нужное сочетание слов или символов, которое соответствует признакам, отраженным в регулярном выражении. Иными словами, регулярное выражение – это фильтр для текста.

В Linux регулярные выражения используются командой `grep`, которая позволяет искать файлы с определенным содержанием либо выделять из файлов строки с необходимым содержимым (например, номера телефонов, даты и т. д.).

7. Знакозаменяющие метасимволы.

Не все символы можно использовать прямо по назначению.

Посмотрите, например, на конструкцию, которая описывалась в предыдущем разделе. Допустим, требуется найти в каком-то файле строки, содержащие следующий набор символов: `abc[def`. Можно предположить, что регулярное выражение будет составлено по принципам, описанным выше, но это неверно. Открывающая квадратная скобка — это один из символов, который несет для программы, работающей с регулярными выражениями, особый смысл (который был рассмотрен ранее). Такие символы называются метасимволами.

В Linux к знакозаменяющим метасимволам относятся:

- `.` – Предполагает, что в конечном выражении на ее месте будет стоять любой символ.
- `\w` – Замещает любые символы, которые относятся к буквам, цифрам и знаку подчеркивания.
- `\W` – Замещает все символы, кроме букв, цифр и знака подчеркивания (то есть является обратным метасимволу `\w`).
- `\d` – Замещает все цифры.
- `\D` – Замещает все символы, кроме цифр.
- `\b` – Замещает символ перевода курсора на один влево (возврат курсора).
- `\r` – Замещает символ перевода курсора в начало строки.
- `\n` – Замещает символ переноса курсора на новую строку.
- `\t` – Замещает символ горизонтальной табуляции.
- `\v` – Замещает символ вертикальной табуляции.
- `\f` – Замещает символ перехода на новую страницу.
- `\s` – Равнозначен использованию пяти последних метасимволов, то есть вместо метасимвола `\s` можно написать `[\r\n\t\v\f]`.
- `\S` – Является обратным метасимволу `\s`.

8. Метасимволы количества повторений в регулярных выражениях.

В Linux к метасимволам количества повторений в регулярных выражениях относятся:

- `?` – Указывает обработчику регулярных выражений на то, что предыдущий символ, метасимвол или конструкция могут вообще не существовать в конечном тексте либо присутствовать, но иметь не более одного вхождения.
- `*` – Означает, что впереди стоящие символ, метасимвол либо конструкция могут как отсутствовать, так и быть в конечном выражении, причем количество вхождений неограниченно.

· + – Действие схоже с действием предыдущего символа с тем отличием, что впередистоящие метасимвол, символ или конструкция должны повторяться как минимум один раз.

· {min, max} – Указывает минимальное и максимальное количество вхождений.

9. Группировка выражений в регулярных выражениях.

Группировка в регулярных выражениях применяется для соединения нескольких его составляющих в одну единицу.

10. Использование зарезервированных символов в регулярных выражениях.

Некоторые символы имеют для программы, работающей с регулярными выражениями, особый смысл. Это, например, косая черта, точка, круглая, фигурная и квадратная скобки, звездочка и т. д. Однако не исключено, что в целевом выражении также могут быть эти символы, и их наличие нужно будет определить в регулярном. В данном случае эти символы нужно указать особым образом. Для этого перед нужным символом ставят косую черту, то есть, чтобы указать наличие в конечном тексте символа звездочки, в регулярном выражении на соответствующем месте следует написать *.

11. Подсчет количества элементов текстового файла.

Часто бывает необходимо подсчитать количество определенных элементов текстового файла. Для подсчета строк, слов и символов служит стандартная утилита – wc (от англ. «word count» – «подсчет слов»).

12. Назначение утилит head, tail, cut.

Иногда пользователя интересует только часть из тех данных, которые собирается выводить программа. Утилита head нужна, чтобы вывести только первые несколько строк файла. Не менее полезна утилита tail (англ. «хвост»), выводящая только последние строки файла. Если же пользователя интересует только определенная часть каждой строки файла - поможет утилита cut.

13. Назначение и использование утилиты grep.

Зачастую пользователю нужно найти только упоминания чего-то конкретного среди данных, выводимых утилитой. Обычно эта задача сводится к поиску строк, в которых встречается определенное слово или комбинация символов. Для этого подходит стандартная утилита `grep`, которая может искать строку в файлах, а может работать как фильтр: получив строки со стандартного ввода, она выведет на стандартный вывод только те строки, где встретилось искомое сочетание символов.

14. Выполнение транслитерации.

Удобство работы с потоком не в последнюю очередь состоит в том, что можно не только выборочно передавать результаты работы программ, но и автоматически заменять один текст другим прямо в потоке. Для замены одних символов другими предназначена утилита `tr` (сокращение от англ. «translate» - «преобразовывать, переводить»), работающая как фильтр.

15. Назначение потокового редактора `sed`.

Помимо простой замены отдельных символов, возможна замена последовательностей (слов). Специально для этого предназначен потоковый редактор `sed` (сокращение от англ. «stream editor»). Он работает как фильтр и выполняет редактирование поступающих строк: замену одних последовательностей символов другими, причем можно заменять и регулярные выражения