ИУ-10
Системное
Программное
Обеспечение
Администрирование Linux

Утилиты для жизни в linux и работы текстовыми файлами

Москва, 2022

На этом уроке

1. Разберем утилиты, с помощью которых можно читать текстовые файлы
2. Познакомимся с текстовыми редакторами vim и nano
3. Разберем регулярные выражения и их применение
4. Познакомимся с утилитами для работы с текстовыми файлами

Содержание

На этом уроке **1**

Знакомство с текстовыми редакторами **2**
Утилиты для чтения текстовых файлов 2
Текстовый редактор vi/Vim 2
Текстовый редактор nano 5

Перенаправление потоков ввода и вывода **6**

Утилиты для работы с текстом **8**
Регулярные выражения 8
grep 10
SED 11
AWK 12
cut,sort,tr 13

Практическое задание **15**

Глоссарий **15**

Дополнительные материалы **16**

Используемые источники **16**

Знакомство с текстовыми редакторами

Утилиты для чтения текстовых файлов

Для просмотра содержимого файлов существует несколько полезных утилит:

1. Команда **cat (concatenate)** позволяет быстро прочитать содержимое файла, а также склеить несколько файлов в один. Например, `cat file` выведет на экран содержимое файла с именем file. Иногда полезно запускать с опцией -A, чтобы увидеть непечатаемые символы

2. Программы постраничного просмотра текста **less** и **more**. Основное их различие заключается в том, что **less** позволяет просмотр в обе стороны (вверх и вниз) за счёт создаваемого буфера.

3. Команда **tail** позволит вывести на экран заданное количество строк от конца файла или содержимое файла в режиме интерактивного просмотра. Например, `tail -20 file` - покажет последние 20 строк файла file. `tail -f /var/log/syslog` непрерывно выводит на экран содержимое файла syslog по мере его обновления. Для прерывания работы tail -f, используйте комбинацию Ctrl-C

4. По аналогии с tail утилита head будет показывать первые линии файла

5. Текстовые редакторы.

Текстовый редактор vi/Vim

Внимание! *В силу схожести управления редакторами vi и Vim далее в тексте рассматриваемработу с Vim.*

Vim — текстовый редактор, обратно совместимый с vi. Он может быть использован для правки всех видов простого текста. По сравнению с vi, Vim имеет много усовершенствований: многократная отмена операций, множественность окон и буферов, подсветка синтаксиса, правка командной строки, автодополнение имён файлов, встроенная справка, визуальное выделение и т. п. Vim имеет большое количество плагинов, используя которые можно превратить этот редактор в довольно удобную IDE. Также с редактором устанавливается программа-тренажер **vimtutor**, которая поможет почувствовать себя более уверенно в работе с данным редактором.

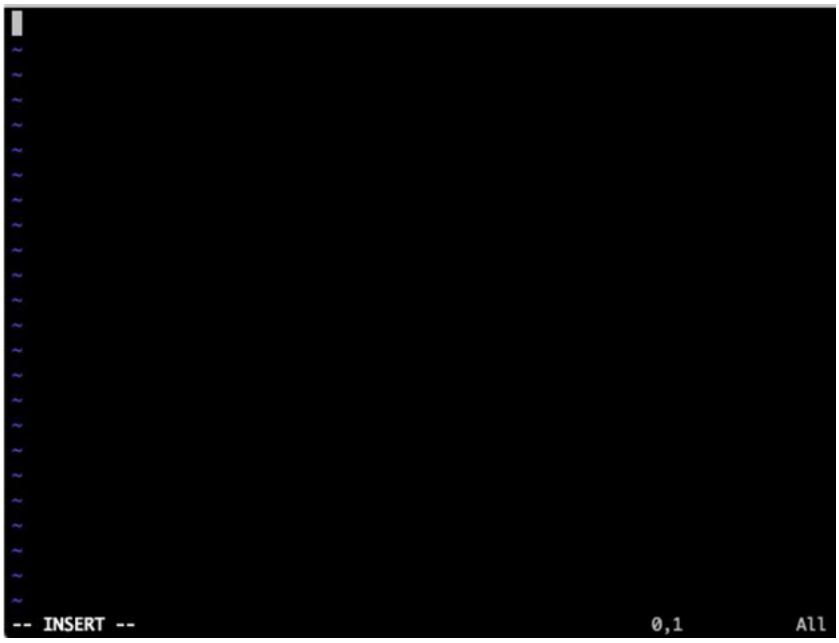
Разберем базовые элементы управления этим редактором.

Запускаем редактор командой **vim myfile**:

Vim/vi имеет три режима:

1. **Командный режим**. В этом режиме осуществляется навигация по файлу, выполняются редактирующие действия с файлом (удаление символа, копирование, вставка и т. д). **Важно**: команды редактирования вызываются обычными латинскими буквами. Навигация по строкам осуществляется либо при помощи клавиш «Вверх», «Вниз», «Влево», «Вправо», либо используя буквы: **k** — вверх, **j** — вниз, **h** — влево, **l** — вправо. Удаление символа под курсором — клавиша **x**. Удаление строки: **dd** (дважды нажать клавишу d). Пролистывание страниц: клавиши **PgUp**, **PgDn** или комбинации **ctrl** + **F**, **ctrl** + **B**.

2. **Режим редактирования**. В этом режиме мы можем писать в файл. Для перехода в режим редактирования используется одна из команд: клавиша **i** начнёт редактирование строки с текущего положения курсора, клавиша **a** начнёт редактирование строки со следующего после курсора символа, клавиша **o** начнёт редактирование текста со следующей строки. Вернёмся к нашему примеру и переведём редактор в режим редактирования, нажав клавишу **i**:



Курсор начнёт мигать, в левом нижнем углу появится сообщение `- - ВСТАВКА - -` или `- - INSERT - -` , что говорит нам о том, что редактор находится в режиме редактирования и мы можем писать свой текст.

3. **Режим последней строки** — специальный режим редактора, в котором мы можем передавать ему сложные команды, например на сохранение файла, выход из редактора. Для перехода в этот режим необходимо проделать следующее: после окончания работы с текстом нажимаем клавишу **Esc** (выходим из режима редактирования), далее нажимаем клавишу «:» (двоеточие) и получаем в последней строке редактора приглашение к вводу команд:

После двоеточия мы можем использовать следующие команды: **q** — выйти из редактора, **q!** — выйти из редактора без сохранения изменений, **w** — сохранить файл, **wq** — сохранить и выйти из редактора. После ввода нужной команды нажимаем клавишу **enter**. Более подробно ознакомиться с командами редактора Vim, как и говорилось выше, поможет программа **vimtutor**.

Текстовый редактор nano

nano — консольный текстовый редактор для Unix и Unix-подобных операционных систем, основанный на библиотеке curses и распространяемый под лицензией GNU GPL. В настоящее время включён в дистрибутивы Ubuntu по умолчанию и в установке не нуждается. Более прост и понятен для начинающего пользователя Linux в сравнении с редакторами vi и Vim. Запускаем редактор следующей командой: **nano myfile**.

Сразу после запуска открытый файл становится доступным для редактирования. В отличие от редактора Vim, у nano нет каких-то особых режимов работы. Ниже области редактирования располагается панель с наиболее популярными комбинациями клавиш управления редактором. ^ означает клавишу Ctrl. Сохранение файла — комбинация клавиш **Ctrl + o**, выход из редактора — **Ctrl + x**.

Перенаправление потоков ввода и вывода

Продолжая знакомиться с утилитами для жизни в оболочке Bash, нельзя пропустить стандартные потоки ввода-вывода. Давайте разберем, что это такое и как мы можем их использовать при администрировании системы.

Одним из принципов UNIX-Way является идея, что каждая программа хорошо отлажена и решает только одну задачу, но сами программы-утилиты можно комбинировать, что очень часто применяется. Сейчас мы разберем, каким образом это работает.

При запуске каждая программа автоматически открывает 3 специальных файла, которые ещё называют потоками ввода-вывода. Их различают по номерам файловых дескрипторов (указателей на файл):

- **0** — стандартный поток ввода (**STDIN**). Файл, из которого осуществляется чтение данных.
- **1** — стандартный поток вывода (**STDOUT**). Файл, в который осуществляется запись данных.
- **2** — стандартный поток ошибок (**STDERR**). Файл, в который осуществляется запись об ошибках или сообщения, которые не могут быть записаны в стандартный поток вывода.

Информацию из потоков можно перенаправить в любой другой файл, используя следующие символы:

1. < — перенаправление стандартного ввода, например, <**file**. Процесс будет использовать файл **file** как источник данных.

```
user@server:~$ cat some_text.txt

hello World!

user@server:~$ tr [:lower:] [:upper:] < some_text.txt

HELLO WORLD!
```

2. >, >> — перенаправление стандартного вывода, например, >**file**. Сообщения от процесса будут направлены в файл с именем **file**. Если файл не существует, он будет создан, в противном случае перезаписан. Для записи сообщений в конец файла используется символ >>, например, >>**file**. Сообщения от процесса будут перенаправлены в файл с

именем file. Если файл не существует, он будет создан. В противном случае сообщения будут дописаны в конец файла. > и **1**> - синонимы, то есть если не указан номер потока, то по умолчанию используется первый поток (**STDOUT**).

```
user@server:~$ echo "hello world!"

hello world!

user@server:~$ echo "hello World!" > text_file.txt

user@server:~$ cat text_file.txt

hello world!

user@server:~$ echo "Another line!" >> text_file.txt

user@server:~$ cat text_file.txt

hello world!

Another line!
```

3. **2**>, **2**>> — перенаправление стандартного потока ошибок, например, **2**>**file**. Сообщения об ошибках процесса будут направлены в файл с именем **file**. Если файл не существует, он будет создан, в противном случае перезаписан. Для записи сообщений в конец файла используется символ **2**>>, например, **2**>>**file**. Сообщения об ошибках процесса будут перенаправлены в файл с именем file. Если файл не существует, он будет создан, в противном случае сообщения будут дописаны в конец файла.

```
user@server:~$ some_text.txt

some_text.txt: command not found

user@server:~$ some_text.txt 2> err_text.txt

user@server:~$ cat err_text.txt

some_text.txt: command not found
```

4. **2**>**&1** — позволит объединить поток ошибок (stderr) и стандартный вывод (stdout) и перенаправить данные в другой файл.

Стандартные потоки можно перенаправлять не только в файлы, но и на ввод другим процессам. Такое перенаправление называют конвейер (pipeline),

в нём используется специальный символ pipe «|» (вертикальная черта). Например, `command-1 | command-2|...|command-n` перенаправит результат работы команды `command-1` на ввод другой команде — `command-2`, которая, в свою очередь, перенаправит результат своей работы на ввод следующей команде. Такое перенаправление очень часто используется в работе с командной строкой Linux. Например, используя такое перенаправление, мы можем подсчитать количество объектов в текущем каталоге.

```
user@server:~$ ls -l | wc -l

30
```

Утилиты для работы с текстом

Регулярные выражения

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;
- $[\hat{}az]$ — любой символ, кроме тех, что указаны в скобках;
- $[a-z]$ — любой символ из указанного диапазона;
- $[\hat{}a-z]$ — 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 можно применять изученные ранее утилиты: grep, SED и AWK.

grep

Grep находит на вводе целые строки, отвечающие заданному текстовому шаблону, 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;

```
user@server:~$ grep -ir error
/var/log/*
                                                                    16:28:1
URL:http://ftpmaster.internal/ubuntu/pool/main/libg/libgpg-error/libgpg-e
1rror0_1.37-1_amd64.deb                 [58004/58004]                    ->
"/build/chroot//var/cache/apt/archives/partial/libgpg-error0_1.37-1_amd64.
deb"
[1]

/var/log/bootstrap.log:Selecting      previousl     unselecte     packag
libgpg-error0:amd64.

/var/log/bootstrap.log:Preparing to unpack .../libgpg-error0_1.37-1_amd64.
deb ...

/var/log/bootstrap.log:Unpacking libgpg-error0:amd64 (1.37-1) ...
```

```
/var/log/bootstrap.log:Setting up libgpg-error0:amd64 (1.37-1) ...

...
```

- **grep -P** будет использовать расширенные возможности регулярных выражений;

```
user@server:~$ grep -P 'Jan\s\d{2}\s20' /var/log/syslog

Jan 10 20:14:59 server snapd[614]: storehelpers.go:551: cannot refresh:
snap has no updates available: "core18", "lxd"

Jan 10 20:15:09 server systemd[1]: Reloading.

Jan 10 20:15:09 server systemd[1]: /lib/systemd/system/dbus.socket:5:
ListenStream= references a path below legacy directory /var/run/, updating
/var/run/dbus/system_bus_socket -> /run/dbus/system_bus_socket; please
update the unit file accordingly.

Jan 10 20:15:09 server systemd[1]: Reloading.

...
```

- **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**.

```
user@server:~$ cat /var/log/syslog |grep -v named

Jan 10 14:17:33 server systemd[1]: logrotate.service: Succeeded.

Jan 10 14:17:33 server systemd[1]: Finished Rotate log files.

Jan 10 14:17:33 server systemd[1]: fstrim.service: Succeeded.

Jan 10 14:17:33 server systemd[1]: Finished Discard unused blocks on
filesystems from /etc/fstab.

Jan 10 14:17:33 server systemd[1]: man-db.service: Succeeded.

Jan 10 14:17:33 server systemd[1]: Finished Daily man-db regeneration.

...
```

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.

```
user@server:~$ cat file

test

test something

another tesssssst

test

test hello

user@server:~$ sed 's/test/text/g' file

text

text something

another tesssssst

text

text hello
```

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.

```
@server:~$ ls -la | awk '{print $1 }'

total

drwxr-xrx

drwxr

-rw-r--r-

-drwxr

xr-x

-rw

-rw-r--r--

-rw-r--r--
```

cut,sort,tr

Последние команды, которые хотелось бы упомянуть (и продемонстрировать несколько примеров использования конвейера), - это cut, sort и tr. **cut** предназначен для фильтрации вывода. **sort** предназначен для сортировки вывода. А **tr** - для перевода.

Команде cut необходимо передать делимитер (опция d) для разделения различных элементов в строке и номер элемента (опция f) для отображения.

```
root@server:~# cut -f 3 -d : /etc/passwd

0

1
```

```
2

...

41

65534

100
```

Воспользуемся pipe и утилитой sort.

```
root@server:~# cut -f 3 -d : /etc/passwd | sort

0

1

10

100

1000

1001

1002

...

root@server:~# cut -f 3 -d : /etc/passwd | sort -n

0

1

2

3
```

Сама по себе команда sort не понимает, что перед ней числа, поэтому мы получили не совсем ожидаемый результат. Ключ -n привел все в порядок.

И последняя утилита tr поможет с преобразованием текста, для этого возьмем первое поле с логином.

```
root@server:~# cut -f 1 -d : /etc/passwd | tr [:lower:] [:upper:]

ROOT

DAEMO

N BIN
```

```
SYS

SYNC

GAMES

MAN
```

Практическое задание

1. Создать файл с наполнением, используя несколько способов. Использовать изученные на занятии текстовые редакторы для наполнения файлов произвольными данными.

2. Попробовать вывести с помощью cat содержимое всех файлов в директории **/etc** (cat /etc/*). Направить ошибки в отдельный файл в вашей домашней директории. Сколько объектов **не** удалось прочесть?

3. Использовать команду cut 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. Потом с помощью wc подсчитать различные типы разрешений в этом каталоге. ***Попробовать*** убрать из подсчета строку **total**.

4. * В ОС Linux скрытыми файлами считаются те, имена которых начинаются с точки. Сколько скрытых файлов в вашем домашнем каталоге?

5. * Используя дополнительный материал, настроить авторизацию по SSH с использованием ключей.

Глоссарий

Консольные текстовые редакторы — редакторы текста для командной строки в ОС Linux. Небольшой обзор редакторов.

Стандартные потоки — специальный тип потоков, имеющих свой номер (дескриптор) и предназначенных для выполнения стандартных функций: ожидание команд пользователя, вывод данных на экран, вывод сообщений об ошибках на экран.

Конвейер (pipeline) — набор процессов, для которых реализована следующая схема: вывод результата работы одного процесса передаётся на

ввод другому процессу.

Аутентификация — процедура проверки подлинности, например сравнением введённого пароля пользователя с паролем, сохранённым в базе данных паролей.

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.

Дополнительные материалы

1. *Регулярные выражения*
2. *SED*
3. *AWK*
4. *Потоки ввода/вывода*
5. *Авторизация по ключу*
6. *Настройка авторизации по ключу, используя программу PuTTY*
7. *Небольшой обзор редакторов*

Используемые источники

Регулярные выражения в Linux

Костромин В. Linux для пользователя

Статья, посвящённая текстовым редакторам vi/Vim