Регулярные выражения

Механизм **регулярных выражений** являются очень мощным инструментом системы Linux. Регулярные выражения могут использоваться при работе с множеством программ, таких, как `bash, vi, rename, grep, sed` и других.

В данной главе представлены базовые сведения о регулярных выражениях.

Версии синтаксисов регулярных выражений

Существуют три различных версии синтаксисов регулярных выражений:

`BRE: Basic Regular Expressions (Базовый синтаксис регулярных выражений)`

`ERE: Extended Regular Expressions (Расширенный синтаксис регулярных выражений)`

`PCRE: Perl Regular Expressions (Синтаксис регулярных выражений языка программирования Perl)`

В зависимости от используемого инструмента может использоваться один или несколько упомянутых синтаксисов.

К примеру, инструмент **`grep`** поддерживает параметр **`-E`**, позволяющий принудительно использовать расширенный синтаксис регулярных выражений (ERE) при разборе регулярного выражения, в то в время, как параметр **`-G`** позволяет принудительно использовать базовый синтаксис регулярных выражений (BRE), а параметр **`-P`** - синтаксис регулярных выражений языка программирования Perl (PCRE).

Учтите и то, что инструмент **grep** также поддерживает параметр **-F**, позволяющий прочитать регулярное выражение без обработки.

Инструмент **sed** также поддерживает параметры, позволяющие выбирать синтаксис регулярных выражений.

Всегда читайте страницы руководств используемых инструментов!

Утилита grep

Вывод строк, совпадающих с шаблоном

Утилита grep является популярным инструментом систем Linux, предназначенным для поиска строк, которые совпадают с определенным шаблоном. Ниже приведены примеры простейших **регулярных выражений**, которые могут использоваться при работе с ним.

Это содержимое используемого в примерах тестового файла. Данный файл содержит три строки (или три символа **новой строки**).

```
paul@rhel65:~$ cat names
Tania
Laura
Valentina
```

При **поиске** отдельного символа будут выводиться только те строки, которые содержат заданный символ.

```
paul@rhel65:~$ grep u names
Laura
paul@rhel65:~$ grep e names
Valentina
paul@rhel65:~$ grep i names
Tania
Valentina
```

Сравнение с шаблоном, использованным в данном примере, осуществляется очевидным образом; в том случае, если заданный символ встречается в строке, утилита **grep** выведет эту строку.

Объединение символов

Для поиска сочетаний символов в строках символы регулярного выражения должны объединяться аналогичным образом.

В данном примере демонстрируется принцип работы утилиты grep, в соответствии с которым регулярному выражению **ia** будет со-

ответствовать строка Tan**ia**, но не строка V**a**lentina, а регулярному выражению **in** - строка Valent**in**a, но не строка Ta**n**ia.

```
paul@rhel65:~$ grep a names
Tania
Laura
Valentina
paul@rhel65:~$ grep ia names
Tania
paul@rhel65:~$ grep in names
Valentina
paul@rhel65:~$
```

Один или другой символ

Как в синтаксисе PCRE, так и в синтаксисе ERE может использоваться символ создания программного канала, который в данном случае будет представлять логическую операцию "ИЛИ". В данном примере мы будем искать с помощью утилиты grep строки, в которых встречается символ **i** или символ **a**.

```
paul@debian7:~$ cat list
Tania
Laura
paul@debian7:~$ grep -E 'i|a' list
Tania
Laura
```

Обратите внимание на то, что мы используем параметр **-E** утилиты grep для принудительной интерпретации нашего регулярного выражения как выражения, использующего расширенный синтаксис регулярных выражений (ERE).

Нам придется **экранировать** символ создания программного канала в регулярном выражении, использующем базовый синтаксис регулярных выражений (BRE) для аналогичной интерпретации этого символа в качестве логической операции "ИЛИ".

```
paul@debian7:~$ grep -G 'i|a' list
paul@debian7:~$ grep -G 'i\|a' list
Tania
Laura
```

Одно или большее количество совпадений

Символ * соответствует нулю, одному или большему количеству вхождений предыдущего символа, а символ + - последующего символа.

```
paul@debian7:~$ cat list2
ll
lol
lool
loool
paul@debian7:~$ grep -E 'o*' list2
ll
lol
lool
loool
paul@debian7:~$ grep -E 'o+' list2
lol
lool
loool
paul@debian7:~$
```

Совпадение в конце строки

В следующих примерах мы будем использовать данный файл:

```
paul@debian7:~$ cat names
Tania
Laura
Valentina
Fleur
Floor
```

В двух примерах ниже показана методика использования **символа** доллара для поиска совпадения в конце строки.

```
paul@debian7:~$ grep a$ names
Tania
Laura
Valentina
paul@debian7:~$ grep r$ names
```

```
Fleur
Floor
```

Совпадение в начале строки

Символ вставки (**^**) позволяет осуществлять поиск совпадения в начале (или с первых символов) строки.

В данных примерах используется рассмотренный выше файл.

```
paul@debian7:~$ grep ^Val names
Valentina
paul@debian7:~$ grep ^F names
Fleur
Floor
```

Символы доллара и вставки, используемые в регулярных выражениях, называются **якорями** (anchors).

Разделение слов

Последовательность символов **\b** может использоваться в регулярных выражениях в качестве разделителя слов. Рассмотрим в качестве примера следующий файл:

```
paul@debian7:~$ cat text
The governer is governing.
The winter is over.
Can you get over there?
```

При простом поиске строки **over** будет выведено слишком много результирующих строк.

```
paul@debian7:~$ grep over text
The governer is governing.
The winter is over.
Can you get over there?
```

Экранирование разыскиваемых слов с помощью символов пробелов не является удачным решением (так как другие символы также могут использоваться в качестве разделителей слов). В примере ниже показана методика использования последовательности символов **\b** для поиска строк с заданным словом, а не последовательностью символов:

```
paul@debian7:~$ grep '\bover\b' text
The winter is over.
```

```
Can you get over there?
paul@debian7:~$
```

Обратите внимание на то, что утилита **grep** также поддерживает параметр **-w**, предназначенный для осуществления поиска по словам.

```
paul@debian7:~$ cat text
The governer is governing.
The winter is over.
Can you get over there?
paul@debian7:~$ grep -w over text
The winter is over.
Can you get over there?
paul@debian7:~$
```

Параметры утилиты grep

Иногда оказывается проще скомбинировать простое регулярное выражение с параметрами утилиты **grep**, нежели создать более сложное регулярное выражение. Эти параметры обсуждались ранее:

```
grep -i
grep -v
grep -w
grep -A5
grep -B5
grep -C5
```

Предотвращение раскрытия регулярного выражения командной оболочкой

Символ доллара является специальным символом как для регулярного выражения, так и для командной оболочки (вспомните о переменных командной оболочки и встраиваемых командных оболочках). Исходя из этого, рекомендуется при любых обстоятельствах экранировать регулярные выражения, так как экранирование регулярного выражения позволяет предотвратить раскрытие этого выражения командной оболочкой.

```
paul@debian7:~$ grep 'r$' names
Fleur
Floor
```

Утилита rename

Реализации утилиты rename

В дистрибутиве Debain Linux по пути **/usr/bin/rename** расположена ссылка на сценарий **/usr/bin/prename**, устанавливаемый из пакета **perl**.

```
paul@pi ~ $ dpkg -S $(readlink -f $(which rename))
perl: /usr/bin/prename
```

В дистрибутивах, основанных на дистрибутиве Red Hat, не создается аналогичной символьной ссылки для указания на описанный сценарий (конечно же, за исключением тех случаев, когда создается символьная ссылка на сценарий, установленный вручную), поэтому в данном разделе не будет описываться реализация утилиты **rename** из дистрибутива Red Hat.

В дискуссиях об утилите rename в сети Интернет обычно происходит путаница из-за того, что решения, которые отлично работают в дистрибутиве Debian (а также Ubuntu, xubuntu, Mint, ...), не могут использоваться в дистрибутиве Red Hat (а также CentOS, Fedora, ...).

Пакет perl

Команда **rename** на самом деле реализована в форме сценария, использующего **регулярные выражения языка программирования perl**. С полным руководством по использованию данного сценария можно ознакомиться после ввода команды **perldoc perlrequick** (после установки пакета **perldoc**).

```
root@pi:~# aptitude install perl-doc
Следующие НОВЫЕ пакеты будут установлены:
  perl-doc
0 пакетов обновлено, 1 установлено новых, 0 пакетов отме-
чено для удаления, и 0 пакетов не обновлено.
Необходимо получить 8,170 kB архивов. После распаковки
13.2 MB будет занято.
Получить: 1 http://mirrordirector.raspbian.org/raspbian/
wheezy/main perl-do...
Получено 8,170 kБ в 19c (412 kБ/c)
Выбор ранее не выбранного пакета perl-doc.
```

```
(Чтение базы данных ... на данный момент установлено
67121 файл и каталог.)
Распаковывается perl-doc (из .../perl-doc_5.14.2-
21+rpi2_all.deb) ...
Adding 'diversion of /usr/bin/perldoc to
/usr/bin/perldoc.stub by perl-doc'
Обрабатываются триггеры для man-db ...
Настраивается пакет perl-doc (5.14.2-21+rpi2) ...

root@pi:~# perldoc perlrequick
```

Хорошо известный синтаксис

Чаще всего утилита **rename** используется для поиска файлов с именами, соответствующими определенному шаблону в форме **строки**, и замены данной строки на **другую строку**.

Обычно данное действие описывается с помощью регулярного выражения **s/строка/другая строка/**, как показано в примере:

```
paul@pi ~ $ ls
abc       allfiles.TXT  bllfiles.TXT  Scratch
tennis2.TXT
abc.conf  backup        cllfiles.TXT  temp.TXT
tennis.TXT
paul@pi ~ $ rename 's/TXT/text/' *
paul@pi ~ $ ls
abc       allfiles.text  bllfiles.text  Scratch
tennis2.text
abc.conf  backup         cllfiles.text  temp.text
tennis.text
```

А ниже приведен другой пример, в котором используется хорошо известный синтаксис утилиты **rename** для повторного изменения расширений тех же файлов:

```
paul@pi ~ $ ls
abc       allfiles.text  bllfiles.text  Scratch
tennis2.text
abc.conf  backup         cllfiles.text  temp.text
tennis.text
paul@pi ~ $ rename 's/text/txt/' *.text
paul@pi ~ $ ls
```

```
abc       allfiles.txt  bllfiles.txt  Scratch
tennis2.txt
abc.conf  backup        cllfiles.txt  temp.txt
tennis.txt
paul@pi ~ $
```

Эти два примера являются работоспособными по той причине, что используемые нами строки встречаются исключительно в расширениях файлов. Не забывайте о том, что расширения файлов не имеют значения при работе с командной оболочкой bash.

В следующем примере продемонстрирована проблема, с которой можно столкнуться при использовании данного синтаксиса.

```
paul@pi ~ $ touch atxt.txt
paul@pi ~ $ rename 's/txt/problem/' atxt.txt
paul@pi ~ $ ls
abc       allfiles.txt  backup        cllfiles.txt
temp.txt     tennis.txt
abc.conf  aproblem.txt  bllfiles.txt  Scratch
tennis2.txt
paul@pi ~ $
```

При исполнении рассматриваемой команды осуществляется замена исключительно первого вхождения разыскиваемой строки.

Глобальная замена

Синтаксис, использованный в предыдущем примере, может быть описан следующим образом: **s/регулярное выражение/строка для замены/**. Это описание является простым и очевидным, так как вам придется всего лишь разместить **регулярное выражение** между двумя первыми слэшами и **строку для замены** между двумя последними слэшами.

В следующем примере данный синтаксис немного расширяется благодаря добавлению **модификатора**.

```
paul@pi ~ $ rename -n 's/TXT/txt/g' aTXT.TXT
aTXT.TXT renamed as atxt.txt
paul@pi ~ $
```

Теперь используемый нами синтаксис может быть описан как **s/регулярное выражение/строка для замены/g**, где модификатор **s** обозначает операцию замены (**switch**), а модификатор **g** - сообщает о необходимости осуществления глобальной замены (**global**).

Обратите внимание на то, что в данном примере был использован параметр **-n** для вывода информации о выполняемой операции (вместо выполнения самой операции, заключающейся в непосредственном переименовании файла).

Замена без учета регистра

Другим **модификатором**, который может оказаться полезным, является модификатор **i**. В примере ниже показана методика замены строки на другую строку без учета регистра.

```
paul@debian7:~/files$ ls
file1.text  file2.TEXT  file3.txt
paul@debian7:~/files$ rename 's/.text/.txt/i' *
paul@debian7:~/files$ ls
file1.txt  file2.txt  file3.txt
paul@debian7:~/files$
```

Изменение расширений

Интерфейс командной строки Linux не имеет представления о расширениях файлов, аналогичных применяемым в операционной системе MS-DOS, но многие пользователи и приложения с графическим интерфейсом используют их.

В данном разделе приведен пример использования утилиты **rename** для изменения исключительно расширений файлов. В примере используется символ доллара для указания на то, что точкой отсчета для замены является окончание имени файла.

```
paul@pi ~ $ ls *.txt
allfiles.txt  bllfiles.txt  cllfiles.txt  really.txt.txt
temp.txt  tennis.txt
paul@pi ~ $ rename 's/.txt$/.TXT/' *.txt
paul@pi ~ $ ls *.TXT
allfiles.TXT  bllfiles.TXT    cllfiles.TXT
really.txt.TXT
temp.TXT      tennis.TXT
paul@pi ~ $
```

Обратите внимание на то, что **символ доллара** в рамках регулярного выражения обозначает **окончание строки**. Без символа доллара исполнение данной команды должно завершиться неудачей в момент обработки имени файла `really.txt.txt`.

Утилита sed

Редактор потока данных

Редактор потока данных (stream editor) или, для краткости, утилита **sed**, использует **регулярные выражения** для модификации потока данных.

В данном примере утилита **sed** используется для замены строки.

```
echo Понедельник | sed 's/Понедель/Втор/'
Вторник
```

Слэши могут быть заменены на некоторые другие символы, которые могут оказаться более удобными и повысить читаемость команды в ряде случаев.

```
echo Понедельник | sed 's:Понедель:Втор:'
Вторник
echo Понедельник | sed 's_Понедель_Втор_'
Вторник
echo Понедельник | sed 's|Понедель|Втор|'
Вторник
```

Интерактивный редактор

Несмотря на то, что утилита **sed** предназначена для обработки потоков данных, она также может использоваться для интерактивной обработки файлов.

```
paul@debian7:~/files$ echo Понедельник > today
paul@debian7:~/files$ cat today
Понедельник
paul@debian7:~/files$ sed -i 's/Понедель/Втор/' today
paul@debian7:~/files$ cat today
Вторник
```

Простые обратные ссылки

Символ **амперсанда** может использоваться для ссылки на искомую (и найденную) строку.

В данном примере **амперсанд** используется для удвоения количества найденных строк.

```
echo Понедельник | sed 's/Понедель/&&/'
ПонедельПонедельник
```

```
echo Понедельник | sed 's/ник/&&/'
Понедельникник
```

Обратные ссылки

Круглые скобки используются для группировки частей регулярного выражения, на которые впоследствии могут быть установлены ссылки.

Рассмотрите следующий пример:

```
paul@debian7:~$ echo Sunday | sed 's_\(Sun\)_\1ny_'
Sunnyday
paul@debian7:~$ echo Sunday | sed 's_\(Sun\)_\1ny \1_'
Sunny Sunday
```

Точка для обозначения любого символа

В **регулярном выражении** простой символ точки может обозначать любой символ.

```
paul@debian7:~$ echo 2014-04-01 | sed 's/....-..-../YYYY-MM-DD/'
YYYY-MM-DD
paul@debian7:~$ echo abcd-ef-gh | sed 's/....-..-../YYYY-MM-DD/'
YYYY-MM-DD
```

Множественные обратные ссылки

В случае использования более чем одной пары **круглых скобок**, ссылка на каждую из них может быть осуществлена путем использования последовательных числовых значений.

```
paul@debian7:~$ echo 2014-04-01 | sed 's/\(....\)-\
(..\)-\(..\)/\1+\2+\3/'
2014+04+01
paul@debian7:~$ echo 2014-04-01 | sed 's/\(....\)-\
(..\)-\(..\)/\3:\2:\1/'
01:04:2014
```

Данная возможность называется **группировкой** (grouping).

Пробел

Последовательность символов `\s` может использоваться для ссылки на такой символ, как символ пробела или табуляции.

В данном примере осуществляется глобальный поиск последовательностей символов пробелов (\s), которые заменяются на 1 символ пробела.

```
paul@debian7:~$ echo -e 'сегодня\tтеплый\tдень'
сегодня теплый  день
paul@debian7:~$ echo -e 'сегодня\tтеплый\tдень' | sed 's_\s_ _g'
сегодня теплый день
```

Необязательные вхождения

Символ знака вопроса указывает на то, что предыдущий символ является **необязательным**.

В примере ниже осуществляется поиск последовательности из трех символов o, причем третий символ o является необязательным.

```
paul@debian7:~$ cat list2
ll
lol
lool
loool
paul@debian7:~$ grep -E 'ooo?' list2
lool
loool
paul@debian7:~$ cat list2 | sed 's/ooo\?/A/'
ll
lol
lAl
lAl
```

Ровно n повторений

Вы можете указать точное количество повторений предыдущего символа.

В данном примере осуществляется поиск строк с ровно тремя символами o.

```
paul@debian7:~$ cat list2
ll
lol
lool
```

```
loool
paul@debian7:~$ grep -E 'o{3}' list2
loool
paul@debian7:~$ cat list2 | sed 's/o\{3\}/A/'
ll
lol
lool
lAl
paul@debian7:~$
```

От n до m повторений

А в данном примере мы четко указываем, что символ должен повторяться от минимального (2) до максимального (3) количества раз.

```
paul@debian7:~$ cat list2
ll
lol
lool
loool
paul@debian7:~$ grep -E 'o{2,3}' list2
lool
loool
paul@debian7:~$ grep 'o\{2,3\}' list2
lool
loool
paul@debian7:~$ cat list2 | sed 's/o\{2,3\}/A/'
ll
lol
lAl
lAl
paul@debian7:~$
```

История командной оболочки bash

Командная **оболочка bash** также может интерпретировать некоторые регулярные выражения.

В данном примере показана методика манипуляций с символом восклицательного знака в рамках маски поиска в истории командной оболочки bash.

```
paul@debian7:~$ mkdir hist
paul@debian7:~$ cd hist/
paul@debian7:~/hist$ touch file1 file2 file3
paul@debian7:~/hist$ ls -l file1
-rw-r--r-- 1 paul paul 0 апр 15 22:07 file1
paul@debian7:~/hist$ !l
ls -l file1
-rw-r--r-- 1 paul paul 0 апр 15 22:07 file1
paul@debian7:~/hist$ !l:s/1/3
ls -l file3
-rw-r--r-- 1 paul paul 0 Апр 15 22:07 file3
paul@debian7:~/hist$
```

Данная методика также работает в случае использования чисел при чтении истории команд командной оболочки bash.

```
paul@debian7:~/hist$ history 6
 2089  mkdir hist
 2090  cd hist/
 2091  touch file1 file2 file3
 2092  ls -l file1
 2093  ls -l file3
 2094  history 6
paul@debian7:~/hist$ !2092
ls -l file1
-rw-r--r-- 1 paul paul 0 апр 15 22:07 file1
paul@debian7:~/hist$ !2092:s/1/2
ls -l file2
-rw-r--r-- 1 paul paul 0 апр 15 22:07 file2
paul@debian7:~/hist$
```