Регулярные выражения Bash: полный гайд

timeweb.cloud/tutorials/linux/regulyarnye-vyrazheniya-bash-gajd

Команда Timeweb Cloud 27 июля 2022 г.

Одним из принципов Unix-систем является широкое использование текстовых данных: конфигурационные файлы, входные и выходные данные программ в *nix часто организованы в виде обычного текста. Регулярные выражения — это мощный инструмент для манипуляции текстовой информацией. В этом гайде разберем тонкости работы с регулярными выражениями Bash, которые помогут вам реализовать весь потенциал командной строки и скриптов в Linux.

Что такое регулярные выражения?

Регулярные выражения — это специальным образом записанные строки, используемые для поиска символьных шаблонов в тексте. Чем-то они похожи на групповые символы в оболочке, но их возможности куда шире. Многие утилиты для работы с текстом в Linux и языки программирования включают в себя механизм регулярных выражений. Здесь возникают проблемы: разные программы и языки оперируют различными диалектами регулярных выражений. В этой статье рассмотрим стандарт POSIX, которому соответствуют большинство утилит в Linux.

Утилита grep

Программа `grep` — это основной инструмент для работы с регулярными выражениями. Grep анализирует данные со стандартного ввода, ищет совпадения с указанным шаблоном и выводит все подходящие строки. Обычно grep предустановлен в большинстве дистрибутивов. Если хотите потренироваться в использовании регулярных выражений, можете повторять описанные команды в виртуальной машине, либо на арендованном сервере.

Grep имеет следующий синтаксис:

```
grep [параметры] регулярное_выражение [файл…]
```

Самый простой случай использования grep — поиск строк, содержащих фиксированную подстроку. В следующем примере grep вывел все строки, содержащие последовательность nologin:

```
grep nologin /etc/passwd
```

```
Output:
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
games:x:5:60:games:/usr/games:/usr/sbin/nologin
...
```

У grep имеются множество параметров. Подробно с ними ознакомиться можно в документации. Приведем ключи, полезные при работе с регулярными выражениями:

Ключ `-v` — инвертировать критерий. В этом случае grep выводит строки, не содержащие совпадений:

```
ls /bin | grep -v zip
```

```
Output:
[
411toppm
7z
7za
7zr
…
```

Ключ `-i` — игнорировать регистр символов.

Ключ `-o` — выводить не строки, а только совпадения с шаблоном:

```
  ls /bin | grep -o zip

Output:
zip
zip
zip
zip
…
```

Ключ `-w` — искать только строки, содержащие все слово, которое составляет шаблон.

```
  ls /bin | grep -w zip

Output:
gpg-zip
zip
```

Для сравнения та же команда без опции. В вывод также попали строки, содержащие шаблон в качестве подслова в слове.

```
  ls /bin | grep zip

Output
bunzip2
bzip2
bzip2recover
funzip
...
```

Basic Regular Expressions

Ранее упоминалось, что существует множество диалектов регулярных выражений. В стандарте POSIX рассматриваются два вида реализаций. Первый — Basic Regular Expressions (BRE). Ему соответствуют практически все POSIX-совместимые программы. Второй — Extended Regular Expression (ERE). Этот вид позволяет создавать более сложные регулярные выражения, но поддерживается не всеми утилитами. Для начала рассмотрим особенности BRE.

Метасимволы и литералы

В тексте выше мы уже столкнулись с простыми регулярными выражениями. К примеру, выражение «zip» обозначает строку, соответствующую следующим критериям: в строке не меньше трех символов; в строке присутствуют символы «z», «i», «p», причем именно в таком порядке; между ними нет других символов. Символы, соответствующие сами себе (как «z», «i», «p») называются литералами. Кроме того, существуют другая категория символов, называемая метасимволами. Они применяются для составления различных критериев поиска. К метасимволам в BRE относятся:

```
^ $ . [ ] * \ -
```

Чтобы использовать метасимвол в качестве литерала, его нужно экранировать с помощью обратного слэша (\). Обратите внимание, что некоторые метасимволы имеют специальное значение в оболочке. Поэтому при передаче регулярного выражения в аргумент команды его следует заключать в кавычки.

Любой символ

Метасимвол «точка» (.) соответствует любому символу в данной позиции. Например:

```
  ls /bin | grep '.zip'

Output:
bunzip2
bzip2
bzip2recover
funzip
gpg-zip
gunzip
gzip
mzip
p7zip
pbzip2
preunzip
prezip
prezip-bin
streamzip
unzip
unzipsfx
```

Здесь имеется один важный момент. Программа zip не вошла в вывод, так как метасимвол «точка» увеличил длину обязательного совпадения до четырех символов.

Якорные символы

Символ «карет» (^) и «доллар» ($) в регулярных выражениях играют роль якорей. Это означает, что в их присутствии совпадение с шаблоном возможно, только если оно будет найдено в начале строки (^) или в ее конце ($).

```
  ls /bin | grep '^zip'
Output:
zip
zipcloak
zipdetails
zipgrep
…
```

```
  ls /bin | grep ‘zip$’
Output:
funzip
gpg-zip
gunzip
…
```

```
  ls /bin | grep ‘^zip$’
Output
zip
```

Регулярное выражение `^$` будет соответствовать пустым строкам.

Множества символов

Кроме описания совпадения с любым символом в заданной позиции (`.`) в регулярных выражениях имеется возможность описать символ из определенного множества. Делается это с помощью квадратных скобок. В следующем примере ищутся соответствия со строками bzip и gzip:

```
  ls /bin | grep '[bg]zip'
Output:
bzip2
bzip2recover
gzip
```

Все метасимволы, кроме двух, теряют свое специальное значение внутри скобок.

Если сразу после открывающей квадратной скобки стоит символ карет (`^`), остальные символы множества интерпретируются как недопустимые в данной позиции. Например:

```
  ls /bin | grep '[^bg]zip'
Output:
bunzip2
funzip
gpg-zip
gunzip
mzip
p7zip
preunzip
prezip
prezip-bin
streamzip
unzip
unzipsfx
```

Включив отрицание, мы получили список имен файлов, содержащих последовательность zip, которой предшествуют любой символ, кроме b или g. Обратите внимание, что имя zip не было найдено. Символ отрицания не отменяет необходимости присутствия символа в заданной позиции. Кроме того символ карет является отрицанием, только если стоит сразу же после открывающей скобки; в противном случае он теряет свое специальное значение.

С помощью дефиса (`-`) можно определять диапазоны символов. Так можно выразить любой диапазон символов и даже нескольких таких диапазонов. К примеру нам нужно найти все имена файлов, начинающиеся с буквы или цифры. Делается это так:

```
  ls ~ | grep '^[A-Za-z0-9]'
Output:
backup
bin
Books
Desktop
docker
Documents
Downloads
GNS3
…
```

Классы символов POSIX

При использовании диапазонов символов существует одна проблема. Диапазоны трактуются по-разному в зависимости от настроек локали. Например, в некоторых ситуациях диапазон [A-Z] интерпретируется в лексикографическом порядке, то есть он включает все алфавитные символы, кроме символа a в нижнем регистре. Для решения этой проблемы в стандарте POSIX придумали несколько классов, описывающих разные множества символов. Некоторые из них:

- `[:alnum:]` — Алфавитно-цифровые символы; эквивалент диапазона [A-Za-z0-9] в ASCII.
- `[:alpha:]` — Алфавитные символы; эквивалент диапазона [A-Za-z] в ASCII.
- `[:digit:]` — Цифры от 0 до 9.
- `[:lower:]` и `[:upper:]` — Символы нижнего и верхнего регистра соответственно.
- `[:space:]` — Пробельные символы, включая пробел, табуляцию, возврат каретки, перевод строки, вертикальную табуляцию и перевод формата.

Наличие классов символов не дает удобного способа выражения частичных диапазонов, таких как [A-M]. Пример использования:

```
  ls ~ | grep '[[:upper:]].*'
Output:
Books
Desktop
Documents
Downloads
GNS3
GOG Games
Learning
Music
...
```

Extended Regular Expressions

Особенности, рассматривавшиеся выше, поддерживаются практически всем POSIX-совместимыми приложениями и приложениями, реализующими BRE (например grep и потоковый редактор sed). Стандарт POSIX ERE позволяет создавать более выразительные регулярные выражения, однако не все программы умеют с ним работать. Диалект ERE поддерживался программой egrep, но GNU-версия grep также поддерживает расширенные регулярные выражения при вызове с ключом `-E`.

В ERE множество метасимволов расширяются следующими:

```
( ) { } ? + |
```

Чередование

Чередование позволяет выбирать совпадение с одним из нескольких выражений. Так же как выражения в квадратных скобках позволяют одному символу соответствовать множеству указанных символов, чередование позволяет находить совпадение с множеством строки или других регулярных выражений. Чередование обозначается метасимволом вертикальной черты:

```
  echo "AAA" | grep -E 'AAA|BBB'
Output:
AAA

echo "BBB" | grep -E 'AAA|BBB'
Output:
BBB

echo "CCC" | grep -E 'AAA|BBB'
```

Группировка

Элементы регулярных выражений можно объединять и ссылаться на них как на один элемент. Делается это с помощью круглых скобок.

Следующее выражение будет соответствовать именам файлов, начинающихся с bz, gz или zip. Если отбросить круглые скобки, смысл регулярного выражения изменится, и ему будут соответствовать имена, начинающиеся с bz или содержащие gz, или zip.

```
  ls /bin | grep -E '^(bz|gz|zip)'
Output:
bzcat
bzgrep
bzip2
bzip2recover
bzless
bzmore
gzexe
gzip
zip
zipdetails
zipgrep
zipinfo
zipsplit
```

Квантификаторы

Квантификаторы позволяют определить число совпадений с элементом. BRE поддерживают несколько способов.

Квантификатор ? означает совпадение с элементом ноль или один раз. Иными словами совпадение с предыдущим элементом необязательно:

```
  echo "tet" | grep -E 'tes?t'
Output:
tet

echo "test" | grep -E 'tes?t'
Output:
test

echo "tesst" | grep -E 'tes?t'
Output:
```

В последнем случае совпадения не найдены, так как буква «s» встретилась дважды.

Подобно метасимволу `?`, звездочка (`*`) обозначает необязательный элемент; однако, в отличие от знака вопроса, этот элемент может встречаться любое число раз, а не только единожды. Рассмотрим предыдущий пример, используя вместо знака вопроса звездочку:

```
  echo "tet" | grep -E 'tes*t'
Output:
tet

echo "test" | grep -E 'tes*t'
Output:
test

echo "tesst" | grep -E 'tes*t'
Output:
tesst
```

На этот раз все три строки совпали с шаблоном.

Метасимвол `+` действует почти так же, как `*`, но требует совпадения с предыдущим элементом не менее одного раза:

```
  echo "tet" | grep -E 'tes+t'
Output:

echo "test" | grep -E 'tes+t'
Output:
test

echo "tesst" | grep -E 'tes+t'
Output:
tesst
```

Теперь с шаблоном не совпала первая строка, так как метасимвол `+` требует хотя бы одного совпадения с предыдущим элементом.

В BRE существуют специальные метасимволы `{` и `}`, которые, в отличие от предыдущих квантификаторов, позволяют выразить минимальное и максимальное число обязательных совпадений. Всего существует четыре возможных способа задания числа совпадений:

- `{n}` — Совпадение, если предыдущий элемент встречается точно n раз.
- `{n,m}` — Совпадение, если предыдущий элемент встречается не менее n и не более m раз.
- `{n,}` — Совпадение, если предыдущий элемент встречается n или более раз.
- `{,m}` — Совпадение, если предыдущий элемент встречается не более m раз.

Пример:

```
  echo "tet" | grep -E "tes{1,3}t"
Output:

echo "test" | grep -E "tes{1,3}t"
Output:
test

echo "tesst" | grep -E "tes{1,3}t"
Output:
tesst

echo "tessst" | grep -E "tes{1,3}t"
Output:
tessst

echo "tesssst" | grep -E "tes{1,3}t"
Output:
```

С шаблоном совпали только те строки, где буква s встречается один, два или три раза.

Регулярные выражения на практике

В качестве заключения рассмотрим пару примеров, как регулярные выражения могут использоваться на практике.

Проверка номеров телефонов

Допустим, у нас имеется список номеров телефонов. Корректный формат номера: (nnn) nnn-nnnn. В списке 10 номеров, три номера имеют некорректный формат.

```
  cat phonenumbers.txt
Output:
(185) 136-1035
(95) 213-1874
(37) 207-2639
(285) 227-1602
(275) 298-1043
(107) 204-2197
(799) 240-1839
(218) 750-7390
(114) 776-2276
(7012) 219-3089
```

Задача — найти неправильные номера. Для этого можно использовать следующую команду:

```
  grep -Ev '^\([0-9]{3}\) [0-9]{3}-[0-9]{4}$' phonenumbers.txt
Output:
(95) 213-1874
(37) 207-2639
(7012) 219-3089
```

Здесь мы использовали параметр `-v`, чтобы обратить сопоставление и вывести только строки, не соответствующие указанному выражению. Поскольку круглые скобки считаются метасимволами в ERE, мы экранировали их обратными слешами, чтобы они интерпретировались как литералы.

Поиск некорректных имен файлов

Команда find поддерживает проверку, основанную на регулярном выражении. Тут важно помнить одну деталь: если grep выводит строку, содержащую совпадение с регулярным выражением, то `find` требует точного совпадения пути. Допустим нам нужно найти в директории имена файлов и каталогов, содержащие пробелы и другие, потенциально вредные символы:

```
find . -regex '.*[^-_./0-9a-zA-Z].*'
```

Последовательность `.*` в начале и конце означает любое количество любых символов. Такой прием необходим, так как `find` требует совпадения всего пути. В квадратных скобках находится отрицание множества допустимых символов в именах файлов. Таким образом любое имя файла или каталога, содержащее хотя бы один символ, не являющиеся дефисом, подчеркиванием, цифрой или латинской буквой, попадет в вывод.

Публичный IP

Заключение

В этой статье мы рассмотрели лишь несколько примеров практического применения регулярных выражений. В начале вам будет сложно придумывать длинные регулярки. Но со временем вы получите опыт, используя регулярные выражения для поиска в разных приложениях, поддерживающих такую возможность.