

Программирование на Bash: Регехр и их использование для синтаксического анализа. Полезные утилиты.



Артем
Палецкий



Артем Палецкий

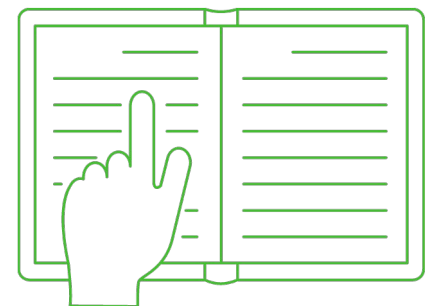
Ведущий инженер, InfoWatch

Предисловие

На этом занятии мы поговорим о:

- регулярных выражениях;
- таких утилитах как `sed`, `awk`, `grep`;
- работе с логами в Linux.

По итогу занятия вы получите представление о работе regex и научитесь использовать их для анализа информации.



План занятия

1. [Предисловие](#)
2. [Регулярные выражения](#)
3. [grep, sed](#)
4. [awk](#)
5. [Итоги](#)
6. [Домашнее задание](#)



Регулярные выражения

Обзор

Regex (Regular expression) – формальный язык поиска и осуществления манипуляций с подстроками в тексте.

Может использоваться для поиска и для замены данных в тексте.

Утилиты, поддерживающие регулярные выражения:

bash

grep

sed

awk

...

Пример регулярного выражения:

`[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}`

Синтаксис

В регулярных выражениях используются свои метасимволы.

Вот основной их список.

.	любой символ;
[]	диапазон символов;
[^]	все символы, кроме указанных в фигурных скобках;
*	любое количество символов перед звездочкой, в том числе ноль;
+	одно или несколько стоящее перед ним выражение;
?	ноль или одно стоящее перед ним выражение;

Синтаксис

В регулярных выражениях используются свои **метасимволы**.

Вот основной их список.

Использование регулярных выражений в `grep`.

`grep -e 'regexp' filename`

.	любой символ;
[]	диапазон символов;
[^]	все символы, кроме указанных в фигурных скобках;
*	любое количество символов перед звездочкой, в том числе ноль;
+	одно или несколько стоящее перед ним выражение;
?	ноль или одно стоящее перед ним выражение;

Синтаксис

. (точка) соответствует любому символу.

Пример:

tes.

Соответствие:

test testtttt 1tes1 ratesk

Несоответствие:

tes vest

.	любой символ;
[]	диапазон символов;
[^]	все символы, кроме указанных в фигурных скобках;
*	любое количество символов перед звездочкой, в том числе ноль;
+	одно или несколько стоящее перед ним выражение;
?	ноль или одно стоящее перед ним выражение;

Синтаксис

[] в скобках перечисляются символы которые необходимо искать.

Пример:

tes[tl]

Соответствие:

test testtttt tesla

Несоответствие:

tes vest tesk

.	любой символ;
[]	диапазон символов;
[^]	все символы, кроме указанных в фигурных скобках;
*	любое количество символов перед звездочкой, в том числе ноль;
+	одно или несколько стоящее перед ним выражение;
?	ноль или одно стоящее перед ним выражение;

Синтаксис

Возможно также использовать диапазон. Заглавные и строчные символы указываются отдельными диапазонами.

Пример:

`tes[a-z]`

Соответствие:

`test testtttt tesla tesz`

Несоответствие:

`tes vest tesA 0tes0`

.	любой символ;
[]	диапазон символов;
[^]	все символы, кроме указанных в фигурных скобках;
*	любое количество символов перед звездочкой, в том числе ноль;
+	одно или несколько стоящее перед ним выражение;
?	ноль или одно стоящее перед ним выражение;

Синтаксис

Аналогично предыдущему пункту но с инверсией.

Пример:

`tes[^t]`

Соответствие:

`tesl tesa tesla tesz`

Несоответствие:

`tester test`

.	любой символ;
[]	диапазон символов;
[^]	все символы, кроме указанных в фигурных скобках;
*	любое количество символов перед звездочкой, в том числе ноль;
+	одно или несколько стоящее перед ним выражение;
?	ноль или одно стоящее перед ним выражение;

Синтаксис

Звездочка – любое количество символов.

Пример:

test*

Соответствие:

test tesa testtttta tesz

Несоответствие:

est

.	любой символ;
[]	диапазон символов;
[^]	все символы, кроме указанных в фигурных скобках;
*	любое количество символов перед звездочкой, в том числе ноль;
+	одно или несколько стоящее перед ним выражение;
?	ноль или одно стоящее перед ним выражение;

Синтаксис

Можно комбинировать с другими выражениями.

Пример:

.*

Соответствие:

Все

Несоответствие:

-

.	любой символ;
[]	диапазон символов;
[^]	все символы, кроме указанных в фигурных скобках;
*	любое количество символов перед звездочкой, в том числе ноль;
+	одно или несколько стоящее перед ним выражение;
?	ноль или одно стоящее перед ним выражение;

Синтаксис

Можно комбинировать с другими выражениями.

Пример:

`ab[cd]*`

Соответствие:

`abcccdddcddcd ab`

Несоответствие:

`af`

.	любой символ;
[]	диапазон символов;
[^]	все символы, кроме указанных в фигурных скобках;
*	любое количество символов перед звездочкой, в том числе ноль;
+	одно или несколько стоящее перед ним выражение;
?	ноль или одно стоящее перед ним выражение;

Синтаксис

Один или более символов или выражений.

Пример:

ab1+

Соответствие:

ab111 ab1

Несоответствие:

ab

.	любой символ;
[]	диапазон символов;
[^]	все символы, кроме указанных в фигурных скобках;
*	любое количество символов перед звездочкой, в том числе ноль;
+	одно или несколько стоящее перед ним выражение;
?	ноль или одно стоящее перед ним выражение;

Синтаксис

Ноль или один символ или выражение.

Пример:

`ab1?$`

Соответствие:

`ab1 ab`

Несоответствие:

`ab11 ab2 abc`

.	любой символ;
[]	диапазон символов;
[^]	все символы, кроме указанных в фигурных скобках;
*	любое количество символов перед звездочкой, в том числе ноль;
+	одно или несколько стоящее перед ним выражение;
?	ноль или одно стоящее перед ним выражение;

Квантификаторы

Квантификатор – указатель количества предшествующих выражений или символов.

Символы $*$, $+$ и $?$ также являются частными случаями квантификаторов.

$$* = \{0, \}$$

$$+ = \{1, \}$$

$$? = \{0, 1\}$$

$\{n\}$	непосредственное количество;
$\{n, \}$	n или больше;
$\{n, m\}$	не меньше чем n и не больше чем m;
$\{, m\}$	не больше чем m;

Квантификаторы

Примеры:

`wa|{2}
| |`

<code>{n}</code>	непосредственное количество;
<code>{n,}</code>	n или больше;
<code>{n,m}</code>	не меньше чем n и не больше чем m;
<code>{,m}</code>	не больше чем m;

Квантификаторы

Примеры:

wa|{2} = watt
| |

tre{1,}

{n}	непосредственное количество;
{n,}	n или больше;
{n,m}	не меньше чем n и не больше чем m;
{,m}	не больше чем m;

Квантификаторы

Примеры:

$\text{wa}[\text{tr}]\{2\} = \text{watt}$

$\text{tre}\{1,\} = \text{tree}$

$\text{gab}\{0,1\}[\text{em}]\{1,3\}$

$\{n\}$	непосредственное количество;
$\{n,\}$	n или больше;
$\{n,m\}$	не меньше чем n и не больше чем m;
$\{,m\}$	не больше чем m;

Квантификаторы

Примеры:

wa[tr]{2} = watt

tre{1,} = tree

gab{0,1}[em]{1,3} = game

tol{,3}

{n}	непосредственное количество;
{n,}	n или больше;
{n,m}	не меньше чем n и не больше чем m;
{,m}	не больше чем m;

Квантификаторы

Примеры:

wa[tr]{2} = watt

tre{1,} = tree

gab{0,1}[em]{1,3} = game

tol{,3} = toll

{n}	непосредственное количество;
{n,}	n или больше;
{n,m}	не меньше чем n и не больше чем m;
{,m}	не больше чем m;

Экранирование и группировка

Для использования в регулярном выражении спецсимвола используется \

Сам \ экранируется как \\

Применить квантификатор к нескольким символам сразу можно используя группировку: ()

В качестве логического ИЛИ можно использовать |

Примеры:

(www.)?netology\[a-z]{2,4}

(Moscow|Paris|London)+



grep, sed

grep

grep – утилита для поиска строк соответствующих регулярному выражению.

Типичное использование:

```
grep -e 'regexp' file
```

Из важных ключей:

- **-v** инвертирует вывод;
- **-c** подсчет количества выводимых строк вместо их вывода;
- **-n** вывод номера строки;
- **-i** нечувствительность к регистру;
- **-r** рекурсивный поиск по всем файлам.

Sed

sed (stream editor) – потоковый текстовый редактор.

Получает входящий поток (в виде файла) и редактирует его.

Типичное использование:

```
sed -e 's/abc/xyz/g' file > file2
```

Заменяет все строки **abc** на **xyz** в файле **file** и перенаправляет получившийся результат в **file2**

Sed

sed (stream editor) – потоковый текстовый редактор.

Получает входящий поток (в виде файла) и редактирует его.

Типичное использование:

```
sed -e 's/abc/xyz/g' file > file2
```

Заменяет все строки **abc** на **xyz** в файле **file** и перенаправляет получившийся результат в **file2**

Внимание, впереди опасность:

ключ **-i, --in-place**.

Производит замену в изначальном файле **file**.

Sed

Вместо файла можно
использовать поток, например,
из другой команды:

```
echo '123asd123' | sed 's/1/3/g'
```

```
ps aux | sed 's/.*root.*//g'
```



awk

Awk

Обладает практически
безграничными возможностями
по **обработке данных**
благодаря встроенному языку
программирования.

Данные поступающие в awk
можно представить в виде
таблицы, где строки – это
строки, а в качестве делителя
столбцов выступает пробел.

Awk

Обладает практически безграничными возможностями по **обработке данных** благодаря встроенному языку программирования.

Данные поступающие в awk можно представить в виде **таблицы**, где строки – это строки, а в качестве делителя столбцов выступает пробел.

Все столбцы заносятся в переменные:

\$0 – вся строка целиком;

\$1 – первый столбец;

\$2 – второй столбец.

...

Чтобы использовать другой разделитель столбцов можно применить ключ **-F**

Например: -F:

Здесь стандартный разделитель заменен на **:**

Awk: синтаксис

Вывод чего либо на экран:

```
print "
```

Объявление переменных:

```
var='value'
```

Условный оператор:

```
if-then-else
```

Цикл while:

```
while (i > 0){}
```

Awk: синтаксис

Вывод чего либо на экран:

```
print "
```

Объявление переменных:

```
var='value'
```

Условный оператор:

```
if-then-else
```

Цикл while:

```
while (i > 0){}
```

Встроенные переменные:

- FS = разделитель столбцов;
- RS = разделитель строк;
- OFS = выходной разделитель столбцов;
- ORS = выходной разделитель строк.

Когда цикл перестает помещаться в одну строку то можно поместить его в файл и указать его имя ключом **-f**:

```
awk -f /tmp/awkscript
```

Awk: примеры

Вывод только нужных нам столбцов:

```
ps aux | awk '{print $2 $8}'
```

Вывод с условием:

```
ps aux | awk '{if ($2 > 200) print $2}'
```

Циклы внутри скрипта:

```
cat /proc/loadavg | awk '{i=1;  
sum=0; while (i<4){ sum += $i;  
i++ } print 'sum' }'
```

Работа с логами с помощью awk

1. Посмотрим сводную статистику по кодам ответа Nginx.

```
cat /var/log/nginx/access.log | awk '{print $9}' | sort -n | uniq -c
```

2. Запишем некоторый фрагмент сетевых логов в помощью tcpdump.

```
tcpdump -n > tcpdump.txt
```

3. Посмотрим статистику по отправленным и принятым пакетам.

```
cat tcpdump.txt | awk '{a=gensub("\.[a-z0-9]+$","", "g", $3);  
b=gensub("\.[a-z0-9:]+$","", "g", $5); print a $4 b}' | sort -n | uniq -c
```



Итоги

Итоги

Сегодня мы рассмотрели регулярные выражения и утилиты для работы с ними и теперь:

- умеем составлять регулярные выражения;
- можем применять их на практике.





Домашнее задание

Домашнее задание

Давайте посмотрим ваше [домашнее задание](#).

- Вопросы по домашней работе задавайте **в чате** мессенджера Slack.
- Задачи можно сдавать **по частям**.
- Зачёт по домашней работе проставляется после того, как **приняты все задачи**.

**Задавайте вопросы и
пишите отзыв о лекции!**

Артем Палецкий