Регулярные выражения

Введение в регулярные выражения

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дополнительная инфа:

<https://habr.com/ru/company/ruvds/blog/327530/>

https://habr.com/ru/company/ruvds/blog/327896/

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Регулярные выражения позволяют осуществлять нечеткий поиск и замену, необходимы для анализа на корректность, могут использоваться для интеллектуальных замен и обработки данных.

Используются в скриптах, при поиске данных, анализе логов, написании конфигурационных файлов. Также используются в программировании, в проверке данных на валидность (проверка на корректность URL, email, дополнительная проверка данных перед отправкой запросов в MySQL и т.д.). Могут использоваться при анализе текстов на естественных и искусственных языках, для фильтрации спама и т.д.

Для освоения можно использовать непосредственно реализацию регулярных выражений (regexp) в языках программирования/утилитах, либо работать с онлайн-ресурсами для проверки регулярных выражений. Например,<https://regex101.com/> или<http://www.regexpal.com/>. Существуют несколько диалектов регулярных выражений: POSIX-совместимые, Perl-совместимые (PCRE — Perl compatibly regular expression), JavaScript.

PCRE-регулярные выражения

Формат шаблона регулярного выражения:

|  |
| --- |
| /шаблон/модификатор |

Самый простой пример:

Шаблон:

|  |
| --- |
| /abc/g |

Тестовая строка:

|  |
| --- |
| abc abc abc aaa |

Будут подсвечены все abc. Попробуйте убрать модификатор g.

Модификаторы:

* g – глобальный поиск;
* i – не учитывать регистр;
* m – многострочный.

Вариант — искать не все вхождения в строке, а проверять всю строку.

Шаблон:

|  |
| --- |
| /^abc$/g |

Тестовая строка:

|  |
| --- |
| Abc |

Совпадает. Тестовая строка:

|  |
| --- |
| Abc  abc |

2 строки – не совпадает.

Но если использовать это выражение, будут подсвечены вхождения на каждой строке.

|  |
| --- |
| /^abc$/gm |

Как вы уже догадались:

* ^ в начале означает начало строки;
* $ в конце шаблона  означает конец строки.

|  |
| --- |
| /^abc/g |

Подсветит только первое вхождение abc abc abc.

|  |
| --- |
| /abc$/g |

Только последнее не найдет соответствий:

|  |
| --- |
| /^abc$/g |

Для этой строки найдет:

|  |
| --- |
| abc |

Для Abc не найдет. Для Abc и abc уже нужно:

|  |
| --- |
| /^abc$/gi |

В шаблоне можно использовать, непосредственно строгие вхождения (за исключением спецсимволов, их надо экранировать: \$ —знак доллара, \. — точка).

Есть и другие варианты:

* /./g — один любой символ;
* /\w/g — один буквенно-цифровой символ и \_;
* /\d/g — одна цифра;
* /\s/g — пробел или его эквивалент (табуляция, перевод строки);
* /\D/g — то, что не является цифрой;
* /\W/g— то, что не является буквенно-цифровым символом;
* /\S/g — то, что не является пробелом;
* \b используется, чтобы обозначить начало или конец слова (т.е. символ стоит перед или после пробела);
* /a\b/g — подсветит все конечные буквы последовательно.

|  |
| --- |
| na dra nva sta |

И даже в:

|  |
| --- |
| na dra nva sta |

* /\bn/g подсветит все начальные буквы n;
* \B позволяет находить не начальные и не конечные символы.

Повторы:

* Если 0 и более раз, используем \*.
* Если 1 и более раз, используем +.
* Если 0 или 1 раз, используем ?.
* /sa+/g — подойдут «sa saa saaa sa saa saaa».
* /sa\*/g — подойдут «s sa saaa» .
* /sa?/g — подойдут «s sa».

Также нежадный поиск доступен:

|  |
| --- |
| /sa+?/g |

|  |
| --- |
| /sa\*?/g |

Жадный берет максимально возможное вхождение, нежадный — минимально. Сравните:

* /.\*\.txt/ для строки «text.txt.txt» выделит всю строку;
* /.\*?\.txt/ выделит «text.txt».

Варианты с произвольным повторением:

* /\w{3}/g строго три алфавитно-цифровых символа;
* /\w{1,3}/g от 1 до 3 символа;
* /\w{0,3}/g от 0 до 3;
* /\w{3,}/g три и более.

Сравните на txt c doc docx. Вариант из нескольких выборов:

|  |
| --- |
| /.+\.(doc|docx)/g |

Имя файла состоит из нескольких символов, точки и расширений — doc или docx.

Вариант выбора нескольких символов:

* /[ab]\*/g — только символы a и b;
* /[a|b]\*/g — только символы a,b и | (здесь он простой символ);
* /[ab-]\*/g — символы a,b и тире;
* /[a-e]\*/g — символы от a до e;
* /[a-e-]\*/g — символы от a до e и тире.

Проверить подстроку на соответствие и определить её второе вхождение. Выделяем подстроку в (подстрока), а затем ее следующее вхождение определяем как \1. Если есть второе вхождение, то следующее как \2. Такие скобки (http|https) тоже считаются:

|  |
| --- |
| /<(a|strong).\*>.\*<\/\1>/g |

Косая черта экранируется, например, http:// будет как http:\/\/.

Сравните:

|  |
| --- |
| <a href=ya.ru>test</a> is  not <strong>google</strong> |

Этот вариант уже не сработает:

|  |
| --- |
| <a href=ya.ru>test</b> |

На месте \1 должно быть «a», так как в шаблоне, а там «b». Поиск назад и вперед. Только для PCRE.

**Просмотр назад (позитивный поиск назад)**

|  |
| --- |
| (?<=шаблон1)шаблон2 |

Шаблон2 будет соответствовать, если ему предшествует шаблон1. Пример:

|  |
| --- |
| (?<=ip=)127\.0\.0\.1 |

ip=**127.0.0.1** — соответствует, а просто 127.0.0.1 — нет. Выделено будет 127.0.0.1

**Просмотр назад с отрицанием (негативный поиск назад)**

|  |
| --- |
| *(?<!шаблон1)шаблон2* |

Шаблон2 будет соответствовать, если ему не предшествует шаблон1.

Пример:

|  |
| --- |
| (?<!ip=)127\.0\.0\.1 |

ip=127.0.0.1 не соответствует, а addr=**127.0.0.1** соответствует, как и **127.0.0.1**. Выделено будет 127.0.0.1  Это просмотр назад.

**Просмотр вперед (позитивный поиск вперед)**

|  |
| --- |
| Шаблон1(?=шаблон2) |

Будет искать шаблон1, если после него следует шаблон2.

Под этот шаблон подойдут **ipaddr**=127.0.0.1 и **ip**=127.0.0.1, а просто ip или ip=1.2.3.4 — нет.

|  |
| --- |
| \w+=(?=127\.0\.0\.1) |

Выделено будет ipaddr и ip соответственно.

**Просмотр вперед с отрицанием (негативный поиск вперед):**

|  |
| --- |
| Шаблон1(?!шаблон2) |

Будет искать шаблон1, если после него не следует шаблон2.

Под этот шаблон подойдут **ipaddr**=1.0.0.1 и **ip**=1.0.0.1, а просто ipaddr=127.0.0.1 или ip=127.0.0.1 — нет.

|  |
| --- |
| \w+=(?!127\.0\.0\.1) |

Выделено будет ipaddr и ip соответственно. Обратите внимание, просмотр назад и вперед может распространяться и на шаблон.

Шаблон:

|  |
| --- |
| /^\w{2,4}(?<!ip)=.{1,15}$/g |

* ipx=1234  — пойдет;
* test=1234 — пойдет;
* ip=1234 — не подойдет.

Обратите внимание, что sip=1234 тоже не подойдет.

POSIX-совместимые регулярные выражения

PCRE-совместимые регулярные выражения более современные и удобные, поддерживаются в Linux наравне с POSIX-выражениями. Иногда можно столкнуться с POSIX-регулярными выражениями. К примеру, grep по умолчанию работает с POSIX-совместимыми выражениями, хотя и с PCRE тоже умеет работать.

Многие особенности POSIX и PCRE похожи, но среди отличий можно отметить классы и категории символов.

|  |  |  |
| --- | --- | --- |
| **POSIX-класс** | **Эквивалент** | **Значение** |
| [:upper:] | [A-Z] | символы верхнего регистра |
| [:lower:] | [a-z] | символы нижнего регистра |
| [:alpha:] | [A-Za-z] | символы верхнего и нижнего регистра |
| [:alnum:] | [A-Za-z0-9] | цифры, символы верхнего и нижнего регистра |
| [:digit:] | [0-9] | цифры |
| [:xdigit:] | [0-9A-Fa-f] | шестнадцатеричные цифры |
| [:punct:] | [.,!?:…] | знаки пунктуации |
| [:blank:] | [ \t] | пробел и табуляция |
| [:space:] | [ \t\n\r\f\v] | пробельные символы (пробелы + переводы строки, табуляция и т.д.) |
| [:cntrl:] |  | символы управления |
| [:graph:] | [^ \t\n\r\f\v] | печатные символы |
| [:print:] | [^\t\n\r\f\v] | печатные символы и пробелы |

# Использование регулярных выражений в grep

Утилита grep позволяет вывести подстроки с нужными вхождениями и подсветить их из указанного файла. Кроме того, с помощью конвейера можно обрабатывать результаты вывода другой команды.

Вывести только файлы, содержащие .sh в данной директории:

|  |
| --- |
| $ ls | grep .sh |

Но такой вариант выведет и файлы, которые просто содержат строку .sh. Например test.sh test попадет под раздачу. Для более точного поиска подойдут регулярные выражения. grep умеет использовать регулярные выражения. Например, если мы хотим, чтобы в вышеприведенном примере действительно были файлы с расширением .sh:

|  |
| --- |
| $ ls | grep \.sh$ |

Найти файлы, в которых есть записи, начинающиеся с root:

|  |
| --- |
| $ grep ^root /etc/\* 2>/dev/null |

Перенаправление потока 2>/dev/null означает, что мы не будем видеть ошибки в результате попытки прочитать директорию как файл.

Если мы хотим использовать PCRE выражения, поможет ключ P:

|  |
| --- |
| $ ls | grep -P '^[\w\d\.]\*\.(sh|conf)$' |

## Ещё возможности grep

Ключ -v позволяет исключить нужные подстроки. grep позволяет осуществлять выборку не только из входящего потока, но и из файла. Вывести все записи, не содержащие root в /etc/passwd:

|  |
| --- |
| $ grep -v root /etc/passwd |

Рекурсивный поиск. Найти все строчки с root в /etc:

|  |
| --- |
| $ grep -R root /etc |

-l — отображать список файлов, содержащих указанное вхождение.

|  |
| --- |
| $ grep -R -l root/etc |

Есть и другие интересные ключи, например:

* -i — без учета регистра;
* -n — отображать номера строк;
* -w — искать слова целиком.

Улучшите вышеуказанные примеры самостоятельно. Например, чтобы действительно находился только root, а не chroot.

## Поиск

find позволяет осуществлять поиск файлов с широкими возможностями.

Вывести все файлы в текущей директории:

|  |
| --- |
| find . |

Вывести все директории в текущей директории. Обратите внимание, что ищет и в поддиректориях:

|  |
| --- |
| find . -type d |

Можно для найденного списка выполнить какую-нибудь команду:

|  |
| --- |
| find .  -type d -exec 'ls' {} \; |

Для каждого файла будет выполнено ls, вместо {} будет подставлено имя файла, экранированное \; после команды обязательно. Умеет find работать и с комбинацией условий и с регулярными выражениями. Подробнее про find <http://pingvinus.ru/note/command-find>.

## sed

sed является потоковым редактором и умеет делать с файлами многое, как это делается в обычных текстовых редакторах, но по заданной команде и в скриптах. sed  берет данные из потока или из файла, делает замены, и выводит данные в поток. Ключ -i позволяет перезаписать исходный файл.

Например, если нам нужно сделать только замену, можно использовать:

|  |
| --- |
| sed 's/<? /<?php /g' -i $file |

Форма заменить что-то на что-то очень часто встречается. Кстати, /g означает то же самое, что и в регулярных выражениях:

|  |
| --- |
| sed 's/Ivanov/Petrov/g' -i $file |

Везде заменить Ivanov нa Petrov. Иногда бывает нужно сделать замену только в определенных строках.

Заменить Ivanov на Petrov только в строках, которые начинаются на user:

|  |
| --- |
| sed '/^user/s/Ivanov/Petrov/g' file |

Как вы поняли, sed умеет работать с регулярными выражениями, и получается у него это очень хорошо. Правда, sed не умеет делать поиск назад и вперед, для этого надо применять, например, find и perl. Найти два слова и поменять их местами:

|  |
| --- |
| sed 's/\([a-z]\*\) \([a-z]\*\)/\2 \1/' |

Удвоить некоторое значение:

|  |
| --- |
| sed 's/root/& &/g' /etc/passwd |