Регулярные выражения grep, egrep, sed

- Вступление
- Маски поиска
- Регулярные выражения
- GREP
- EGREP
- SED

Вступление

Любой текст, таблицы с данными, списки файлов, код программы, содержат в себе определенный набор символов.
Часто возникает задача поиска слов или выражений, принадлежащих к одному типу, но с возможными вариациями в написании, такие как даты, имена файлов с определенным расширением и стандартным названием, e-mail адреса. С другой стороны, есть задачи по нахождению вполне определенных слов, которые могут иметь различное написание, либо поиск, исключающий отдельные символы или классы символов.
Для этих целей были созданы определенные системы, основанные на описании текста при помощи шаблонов. К таким системам относятся и регулярные выражения.

Маски поиска

Любой поиск основан на поиске по некоторому образцу — шаблону или маске поиска.
Шаблон поиска задается при помощи специальных символов: символы-джокеры или метасимволы — символы, которые используются для замены других символов или их последовательностей, приводя таким образом к символьным шаблонам.
Основные такие символы известны и часто используются при поиске файлов на диске или информации в поисковиках.

Маски поиска

Поиск cpp файлов в текущем каталоге
$ find . -iname '*.cpp'
./example2.cpp
./example.cpp
* - заменяет любую последовательность символов, в том числе и отсутствие символов.

Маски поиска

Поиск cpp файлов в текущем каталоге
$ find . -iname '*.cpp'
./example2.cpp
./example.cpp
* - заменяет любую последовательность символов, в том числе и отсутствие символов.

Поиск файлов .cpp, .hpp и т.д. в текущем каталоге
$ find . -iname '*.?pp'
./example2.cpp
./2.hpp
./example.cpp
? - замена одного любого символа.

Маски поиска

Поиск cpp файлов в текущем каталоге
$ find . -iname '*.cpp'
./example2.cpp
./example.cpp
* - заменяет любую последовательность символов, в том числе и отсутствие символов.

Поиск файлов .cpp, .hpp и т.д. в текущем каталоге
$ find . -iname '*.?pp'
./example2.cpp
./2.hpp
./example.cpp
? - замена одного любого символа.

Поиск cpp или hpp файлов в текущем каталоге
$ find . -iname '*.[ch]pp'
./example2.cpp
./2.hpp
./example.cpp
[ ] - поиск символов, содержащихся в скобках.

Регулярные выражения

Регулярные выражения (англ. regular expressions, сокр. RegExp, RegEx, жарг. регэкспы или регексы) — система синтаксического разбора текстовых фрагментов по формализованному шаблону, основанная на системе записи образцов для поиска.
Образец (англ. pattern) задает правило поиска, по-русски также иногда называется «шаблоном», «маской». Регулярные выражения произвели прорыв в электронной обработке текста в конце XX века. Они являются развитием символов-джокеров (англ. wildcard characters).
Сейчас регулярные выражения используются многими текстовыми редакторами и утилитами для поиска и изменения текста на основе выбранных правил. Многие языки программирования поддерживают регулярные выражения для работы со строками. Например, Java, .NET Framework, Perl, PHP, JavaScript, Python и др. имеют встроенную поддержку регулярных выражений.
Набор утилит (включая редактор sed и фильтр grep), поставляемых в дистрибутивах UNIX, одним из первых способствовал популяризации понятия регулярных выражений.

GREP

grep — утилита командной строки, которая находит на вводе строки, отвечающие заданному регулярному выражению, и выводит их.

Название представляет собой акроним английской фразы «search **G**lobally for lines matching the **R**egular **E**xpression, and **P**rint them» — «искать везде строки, соответствующие регулярному выражению, и выводить их».

Команда grep сопоставляет строки исходных файлов с шаблоном, заданным базовым регулярным выражением. Если файлы не указаны, используется стандартный ввод. Обычно каждая успешно сопоставленная строка копируется на стандартный вывод; если исходных файлов несколько, перед найденной строкой выдается имя файла. В качестве шаблонов воспринимаются базовые регулярные выражения (выражения, имеющие своими значениями цепочки символов, и использующие ограниченный набор алфавитно-цифровых и специальных символов).

Использование grep

grep использует в качестве шаблона базовые регулярные выражения.

Базовые регулярные выражения – BRE (**B**asic **R**egular **E**xpressions)
Синтаксис базовых регулярных выражений на данный момент определён как устаревший, но он до сих пор широко распространён из соображений обратной совместимости. Многие UNIX-утилиты используют такие регулярные выражения по умолчанию.

BRE определяются некоторым набором стандартных метасимволов и правилами их написания.

Рассмотрим в качестве примера для поиска cpp файл и увидим, какие возможности предоставляет для работы с ним grep.

Поиск простых слов

(Слово — набор символов ограниченный с обоих сторон символами пунктуации или спец символами)

Grep может просто искать конкретное слово:

```
$ grep Hello ./example.cpp
printf("Hello world!\n"); //Standard string
```

Поиск простых слов

(Слово — набор символов ограниченный с обоих сторон символами пунктуации или спец символами)

Grep может просто искать конкретное слово:
$ grep Hello ./example.cpp
printf("Hello world!\n"); //Standard string

Или строку, но в таком случае её нужно заключать в кавычки:
$ grep 'Hello world' ./example.cpp
printf("Hello world!\n"); //Standard string

Поиск простых слов

(Слово — набор символов ограниченный с обоих сторон символами пунктуации или спец символами)

Grep может просто искать конкретное слово:
$ grep Hello ./example.cpp
printf("Hello world!\n"); //Standard string

Или строку, но в таком случае её нужно заключать в кавычки:
$ grep 'Hello world' ./example.cpp
printf("Hello world!\n"); //Standard string

Часто слово может быть записано в другом регистре — в таком случае можно использовать ключ, игнорирующий регистр:
$ grep -i sum ./example.cpp
/*returns sum of two numbers*/
int Sum(int a, int b)
printf("Num1 + Num2 = %i\n", Sum(number1, number2)); //Call of function Sum and print result

[ ] - перечисление символов

Часто возникает ситуация, когда точное написание искомого слова неизвестно. В этом случае можно использовать квадратные скобки:

$ grep -i N[ua]m1 ./example.cpp

```
int Nam1 = 3; //varible 3

printf("Num1 = % i, Num2 = %i\n", number1, number2); //Print variables

printf("Num1 + Num2 = %i\n", Sum(number1, number2)); //Call of function Sum and print result
```

Grep ищет совпадения по шаблону, где на месте второго символа может быть как u, так и a, т.е. Совпадения типа:

Num1, Nam1, num1, nam1.

. - точка
(замена любого печатного символа)

В некоторых случаях может быть неизвестен какой-либо символ в искомом выражении:

```
$ grep -i swa. ./example.cpp
void swap(int *a, int *b)
void swa()
swap(&number1, &number2); //Call of function
```

Поиск слова swap, после которого идет любой печатный символ.

. - точка
(замена любого печатного символа)

В некоторых случаях может быть неизвестен какой-либо символ в искомом выражении:
$ grep -i swa. ./example.cpp
void swap(int *a, int *b)
void swa()
swap(&number1, &number2); //Call of function

Поиск слова swap, после которого идет любой печатный символ.

Таким образом запрос
$ grep -i 'swa().' ./example.cpp
Не найдет ничего, поскольку после swa() идет перевод строки.

* - отсутствие или повторение символа

Иногда символ может появляться в тексте более одного раза подряд или не появляться вообще.

$ grep -i Hel*o ./example.cpp

```
long heo = 123;

printf("Hello world!\n"); //Standard string
```

Так в переменной heo вообще нет символа l, в то время как в слове Hello их 2.

Спец символы
(^ - начало строки, $ - конец строки)

$ grep -i ^#include ./example.cpp
#include <cstdio>
Поиск шаблона который стоит вначале строки.
Можно использовать такой запрос для поиска всех подключенных файлов.

Спец символы
(^ - начало строки, $ - конец строки)

$ grep -i ^#include ./example.cpp
#include <cstdio>
Поиск шаблона который стоит вначале строки.
Можно использовать такой запрос для поиска всех подключенных файлов.

$ grep ';$' ./example.cpp
*a = *a xor *b;
*b = *a xor *b;
*a = *a xor *b;
return (a + b);
long heo = 123;
Поиск всех строчек, которые заканчиваются ;

Некоторые ключи

Если файл довольно большой, то оказывается удобным знать еще и номер строки:
grep -i -n hello ./example.cpp
27: printf("Hello world!\n"); //Standard string

Некоторые ключи

Если файл довольно большой, то оказывается удобным знать еще и номер строки:
grep -i -n hello ./example.cpp
27: printf("Hello world!\n"); //Standard string

А следующий запрос выводит весь код, исключая строки, содержащие только комментарии:
egrep -v ^/[/*] ./example.cpp

При использовании ключа -v grep выводит не совпадающие с шаблоном строки.
^ - мета символ начала строки
[] позволяют находить комментарии обоих типов, т.е. в файле примера:
//change values of to numbers
/*returns sum of two numbers*/

EGREP

egrep — это короткий вызов grep с ключом -E

Отличие от grep заключается в возможности использовать расширенные регулярные выражения с использованием символьных классов POSIX.

POSIX® (англ. **P**ortable **O**perating **S**ystem **I**nterface for Uni**X** — Переносимый интерфейс операционных систем Unix) — набор стандартов, описывающих интерфейсы между операционной системой и прикладной программой. Стандарт создан для обеспечения совместимости различных UNIX-подобных операционных систем и переносимости прикладных программ на уровне исходного кода, но может быть использован и для не-Unix систем.

Расширенные регулярные выражения или ERE (**E**xtended **R**egular **E**expressions):
1. Отменено использование обратной косой черты для метасимволов { } и ( ).
2. Обратная косая черта перед метасимволом отменяет его специальное значение.
3. Добавлены метасимволы +, ?, |.
4. Возможность использования символьных классов POSIX

Квантификаторы

?, *, +, {n,m}

```
$ egrep swap? ./example.cpp
void swap(int *a, int *b)
void swa()
swap(&number1, &number2); //Call of function
```

Сопоставляет шаблон, где после слова swa может стоять или нет символ p.

? - является упрощенным вызовом {0,1}, т.е. проверкой на наличие или отсутствие символа, стоящего перед квантификатором.

Запрос $ egrep 'swap{0,1}' ./example.cpp выдает аналогичный результат.

Квантификаторы

?, *, +, {n,m}

$ egrep -i 'Hel*' ./example.cpp
long heo = 123;
printf("Hello world!\n"); //Standard string

* - является упрощенным вызовом {0,}, т.е. проверкой на отсутствие или наличие любого количества повторяющегося символа, стоящего перед квантификатором.

$ egrep -i 'Hel+' ./example.cpp
printf("Hello world!\n"); //Standard string

+ - является упрощенным вызовом {1,}, т.е. проверкой на наличие любого количества (один и более) повторяющегося символа, стоящего перед квантификатором.

Квантификаторы

?, *, +, {n,m}

В примерах выше в качестве выражения перед квантификаторами выступал символ, но вообще может и некоторое более сложное выражение.

$ egrep -i '[lu]{2,2}' ./example.cpp

```
#include <cstdio>
//change values of to numbers
printf("Hello world!\n"); //Standard string
swap(&number1, &number2); //Call of function
printf("Num1 + Num2 = %i\n", Sum(number1, number2)); //Call of function Sum and print result
```

Выражение [lu]{2,2} аналогично [lu][lu], т.е. ищутся стоящие подряд символы из [], последовательности: uu, ll, ul, lu.

Общим выражением для квантификаторов является {n,m}, означающее, что в искомом тексте может стоять не менее n (или ноль, если n не задано), но и не более m (или любое кол-во, если m не задано) выражений подряд, определяющихся вырожением перед квантификатором.

Символьные классы

$ egrep 'number1 = [0-9]' ./example.cpp
Позволяет найти строчки, где переменной присваивается определенное значение
int number1 = 1; //varible 1
В скобках указывается диапазон символов от начального до конечного. [0-9] эквивалентно [0123456789].

$ egrep [a-z]+[0-9]+ ./example.cpp
Позволяет найти строки с нумерованными переменными:

```
int number1 = 1; //varible 1
int number2 = 3; //varible 2
int Nam1 = 3; //varible 3
swap(&number1, &number2); //Call of function
printf("Num1 = % i, Num2 = %i\n", number1, number2); //Print variables
printf("Num1 + Num2 = %i\n", Sum(number1, number2)); //Call of function Sum and print result
```

Переменные могут состоять только из символов нижнего регистра от a до z.

Символьные классы

Символьные классы:
[0-9] — цифры
[A-Z] — заглавные буквы
[a-z] — буквы нижнего регистра
Они могут находится в одних скобках в любом порядке.
По сути — это сокращенная запись перечисления всех символов в определенном промежутке.

$ egrep [A-Za-z][a-z]*[0-9]+ ./example.cpp
Позволяет найти строки с нумерованными переменными:

```
int number1 = 1; //varible 1
int number2 = 3; //varible 2
int Nam1 = 3; //varible 3
swap(&number1, &number2); //Call of function
printf("Num1 = % i, Num2 = %i\n", number1, number2); //Print variables
printf("Num1 + Num2 = %i\n", Sum(number1, number2)); //Call of function Sum and print result
```

Переменные могут начинаться с загловной, но остальное может состоять только из символов нижнего регистра от a до z.

Поиск по файлам

Существует возможность поиска по нескольким файлам и в таком случае перед строкой выводится имя файла.

```
$ egrep -i Hello ./example.cpp ./example2.cpp
./example.cpp:    printf("Hello world!\n"); //Standard string
./example2.cpp: printf("Hello world!\n"); //Standard string
```

SED

sed (от англ. Stream EDitor) — потоковый текстовый редактор (а также язык программирования), применяющий различные предопределённые текстовые преобразования к последовательному потоку текстовых данных.

Первоначально был написан как UNIX-утилита Ли Макмахоном (Lee E. McMahon) из Bell Labs в 1973—74 годах. Сейчас sed доступен фактически для любой операционной системы, поддерживающей работу с командной строкой.

Использование SED

Sed можно использовать как grep, выводя строки по шаблону базового регулярного выражения:
$ sed -n /Hello/p ./example.cpp
printf("Hello world!\n"); //Standard string

Можно использовать его для удаления строк (удаление всех пустых строк):
$ sed /^$/d ./example.cpp

После чего отредактированный текст выведется на стандартный вывод. При этом исходный файл **не** изменится.

Sed имеет возможность сразу заменять редактируемый файл результатом при помощи флага -i

$ sed -i /^$/d ./example.cpp

Использование s//

Основным инструментом работы с sed является выражение типа:
$ sed s/искомое_выражение/чем_заменить/имя_файла

Так, например, если выполнить команду:
$ sed s/int/long/ ./example.cpp

То все найденные последовательности int будут заменены на long.

Использование s//

Основным инструментом работы с sed является выражение типа:
$ sed s/искомое_выражение/чем_заменить/ имя_файла

Так, например, если выполнить команду:
$ sed s/int/long/ ./example.cpp

То все найденные последовательности int будут заменены на long.

Но есть проблема заключающаяся в том, что появится и такая строка:
pr**long**f("Hello world!\n"); //Standart string

Для того, чтобы найти верный запрос рассмотрим 2 других примера:

Использование ERE в s//
(расширенных регулярных выражений)

Ключ -r позволяет использовать расширенные регулярные выражения.

```
$ sed -r 's/\/\/.*$//g' ./example2.cpp
```

Этот запрос позволяет убрать комментарии из файлов.

\/ - экранированный символ /
.* - любая последовательность символов
$ - конец строки

Использование s// и \n (n=0,1...10)

Заменим функцию foo с параметрами на функцию bar, а функцию foo без параметров не изменять.

```
$ sed -r 's/foo(\([^\)]+\))/bar\1/' ./example2.cpp
```

foo(int *a,int *b) → bar(int *a,int *b)
foo(&number1, &number2); → bar(&number1, &number2);

\(и \) - экранированные скобки
(\([^\)]+\)) - скобки, внутри которых есть какое-то выражение
\1 - подстановка выражения в скобках

Использование s//

Теперь можем поправить изначальный запрос
$ sed s/int/long/ ./example.cpp

Его правильно записать так:
$ sed -r 's/([^a-zA-Z0-9_])int([^a-zA-Z0-9_])/\1long\2/g' ./example.cpp

([^a-zA-Z0-9_]) - скобки, внутри которых выражение 1 (перед int)
([^a-zA-Z0-9_]) - скобки, внутри которых выражение 2 (после int)
Где \1 и \2 означает подстановку найденного соответствия выражению в скобках.

Sed скрипты

Существует возможность использовать sed скрипты, содержащие очень сложные выражения для замены. Но их расматривать мы не будем.

example.cpp

- Текст файла в примечании

example2.cpp

- Текст файла в примечании