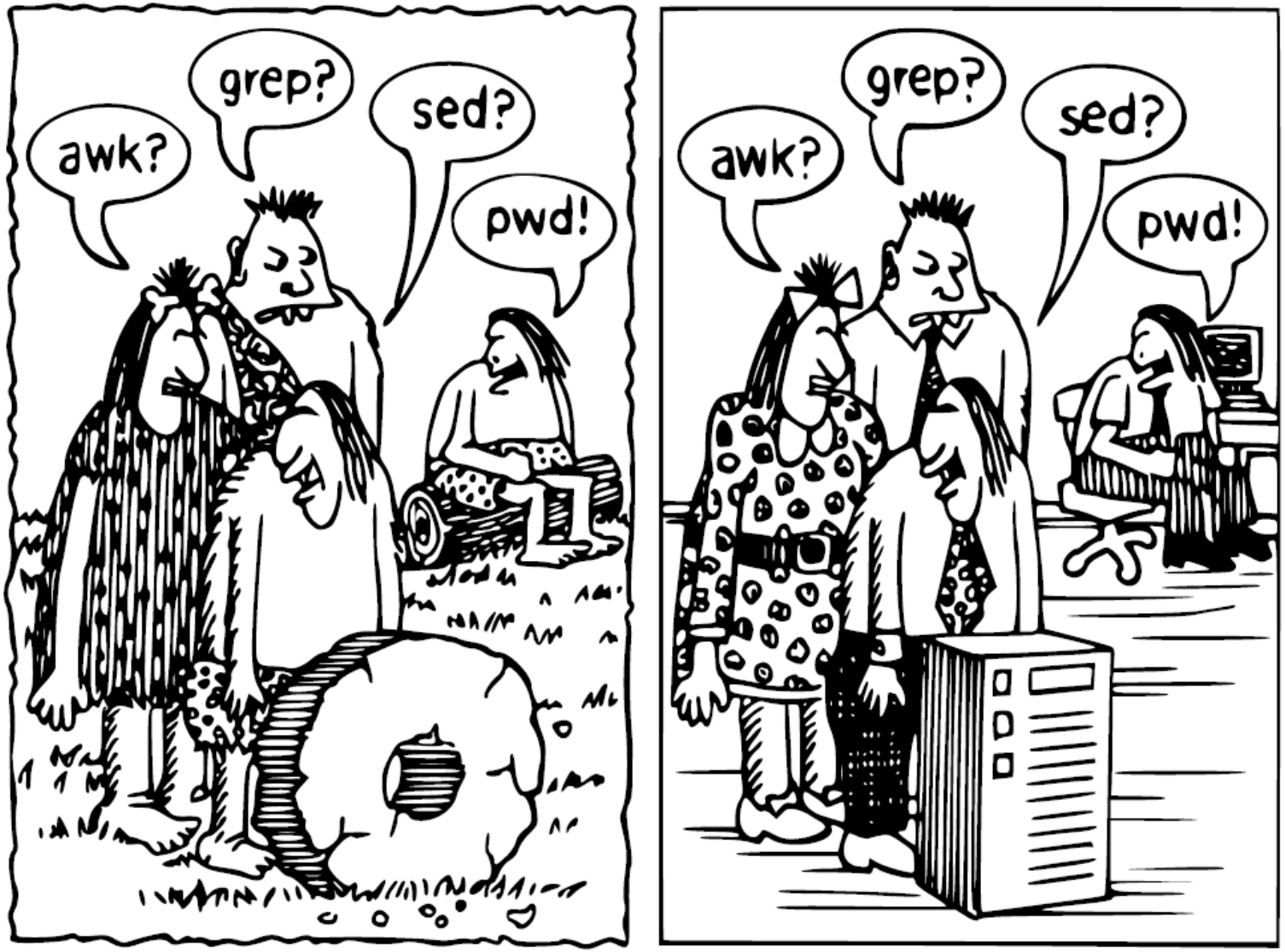
РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ


awk?
grep?
sed?
pwd!
awk?
grep?
sed?
pwd!

ИСТОРИЯ

- Кен Томпсон добавил поиск по регулярному выражению в редактор QED в конце 1960-х, позаимствовав нотацию из теоретической статьи Клини
- Из QED регулярные выражения перекочевали в ed – стандартный текстовый редактор системы

СТАНДАРТЫ

- Стандарт POSIX:
  - Basic Regular Expressions (BRE)
  - Extended Regular Expressions (ERE)

Поддерживаются в утилитах Unix

- Perl Compatible Regular Expressions

Из Perl синтаксис заимствован в Java, .NET, Python, Ruby, JavaScript, и т.д.

ПРИМЕР

- ls *.txt

РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ

- Формальный язык поиска и осуществления манипуляций с подстроками в тексте, основанный на использовании метасимволов (wildcard)
- По сути это «шаблон», состоящий из символов и метасимволов и задающий правило поиска.

ШАБЛОНЫ

Символы в шаблонах делятся на два типа:

- Литералы – обычные символы
- Метасимволы – символы, которые используются для замены других символов или их последовательностей

ЛИТЕРАЛЫ

Литералы:

- все символы за исключением специальных
  - [ ] \ ^ $. | ? * + ( ) { }
- специальные символы предваренные \
  - например: \[или \$

МЕТАСИМВОЛ .

Обозначает один любой символ

Пример:

– st..d – регулярное выражение

– под его описание подходит:

standard

stand

astddd

СИМВОЛЬНЫЕ КЛАССЫ

Позволяет указать, что на данном месте в строке может стоять один из перечисленных символов.

- [A-Z] – любая заглавная латинская буква
- [a-d] – строчная буква от a до d
- [A-Za-z0-9] – Латинская буква или цифра
- [А-Яа-яЁё] – любая русская буква

СИМВОЛЬНЫЕ КЛАССЫ

- Спецсимвол отрицания в символьных классах:

  ^ (крышка)

- [^abc] - все символы (не буквы, а именно символы) кроме букв латинского алфавита a, b, c.

ПЕРЕЧИСЛЕНИЕ

- Вертикальная черта разделяет допустимые варианты. Например, gray|grey соответствует gray или grey

- gr(a|e)y описывают строку gray или grey

ПОЗИЦИЯ ВНУТРИ СТРОКИ

Представление	Позиция	Пример	Соответствие
^	Начало строки	^a	aaa aaa
$	Конец строки	a$	aaa aaa
\b	Граница слова	a\b	aaa aaa
		\ba	aaa aaa
\B	Не граница слова	\Ba\B	aaa aaa

КВАНТИФИКАЦИЯ

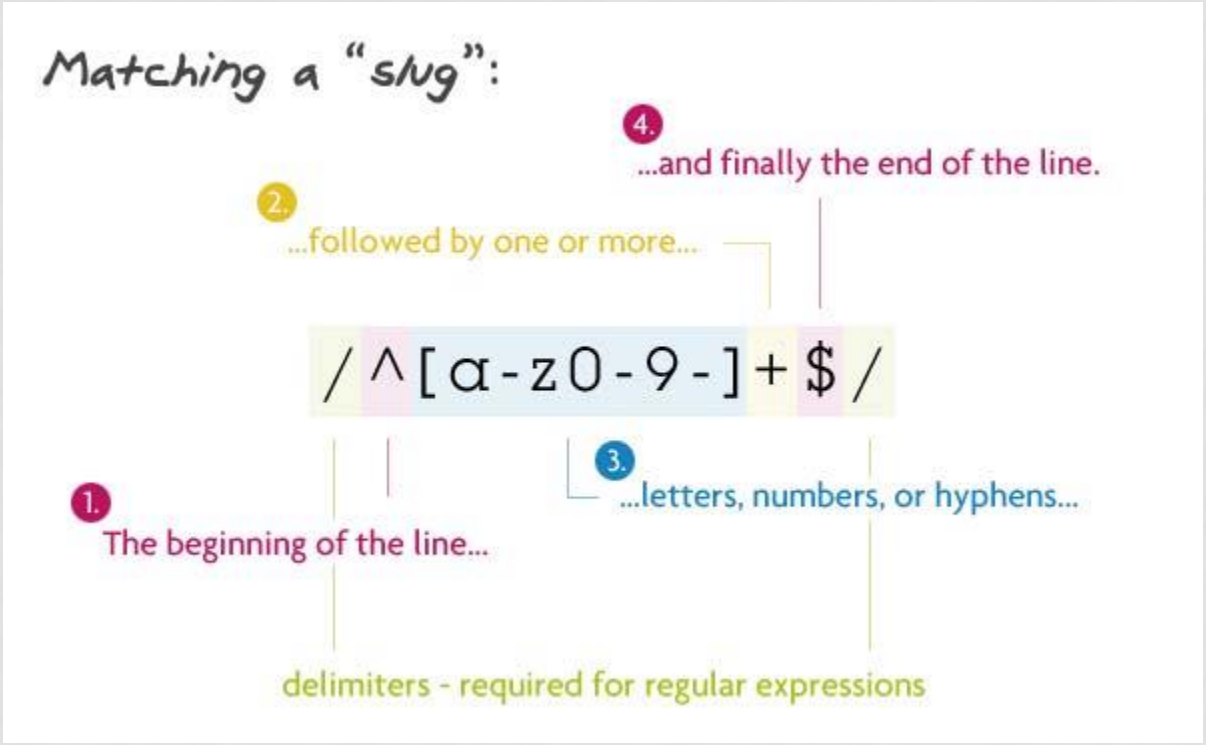
Представление	Число повторений	Пример	Соответствие
{n}	Ровно n раз	colou{3}r	colouuur
{m,n}	От m до n включительно	colou{2,4}r	colouur, colouuur, colouuuur
{m,}	Не менее m	colou{2,}r	colouur, colouuur, colouuuur и т. д.
{,n}	Не более n	colou{,3}r	color, colour, colouur, colouuur
* {0,}	Ноль или более	colou*r	color, colour, colouur и т. д.
+ {1,}	Одно или более	colou+r	colour, colouur и т. д. (но не color)
? {0,1}	Ноль или одно	colou?r	color, colour

ПРИМЕР

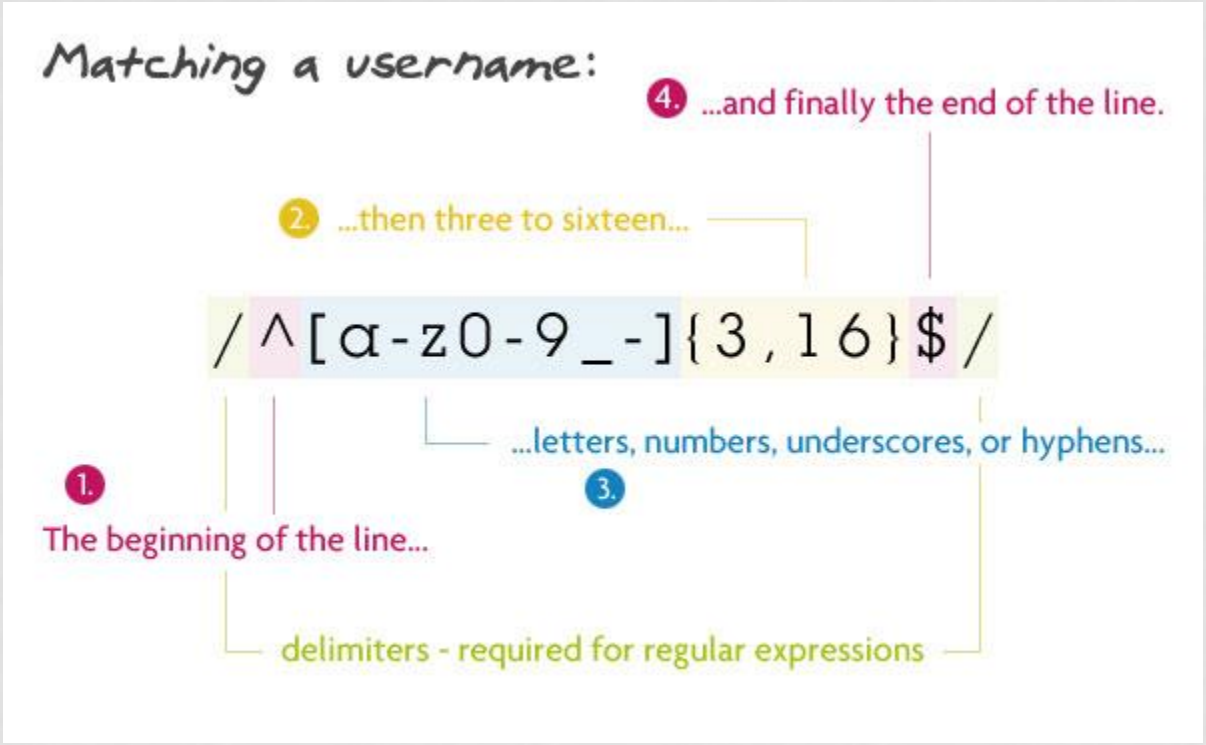
Проверка MAC-адреса

- /^(?[0-9A-Fa-f]{2}:){5}[0-9A-Fa-f]{2}$/

ПРИМЕР


Matching a "slug":
4.
...and finally the end of the line.
2.
...followed by one or more...
/^[a-z0-9-]+$/
3.
...letters, numbers, or hyphens...
1.
The beginning of the line...
delimiters - required for regular expressions

ПРИМЕР


Matching a username:
4. ...and finally the end of the line.
2. ...then three to sixteen...
/^[a-z0-9_-]{3,16}$/
...letters, numbers, underscores, or hyphens...
3.
1.
The beginning of the line...
delimiters - required for regular expressions

КВАНТИФИКАЦИЯ

- «Ленивые» выражения
- «Жадные» выражения
- «Ревнивые»(сверхжадные) выражения

Жадный	Ленивый	Ревнивый
*	*?	*+
?	??	?+
+	+?	++
{n,}	{n,}?	{n,}+

ЖАДНАЯ КВАНТИФИКАЦИЯ

- Выражение (<.*>) соответствует строке, содержащей несколько тегов HTML-разметки, целиком.
- Википедия — свободная энциклопедия, в которой каждый может изменить или дополнить любую статью

ЛЕНИВАЯ КВАНТИФИКАЦИЯ

- Чтобы выделить отдельные теги, можно применить ленивую версию этого выражения: (<.*?>) Ей соответствует не вся показанная выше строка, а отдельные теги (выделены цветом):
- Википедия — свободная энциклопедия, в которой каждый может изменить или дополнить любую статью .

РЕВНИВАЯ (СВЕРХЖАДНАЯ) КВАНТИФИКАЦИЯ

- Захватывает самое большое вхождение. В каком-то смысле, ещё «жаднее» жадных и идет дальше них: *один раз что-то «схватив», они никогда не откатываются назад, они не «отдают» кусочки схваченного ими следующим частям регекспа.*

ПРИМЕР

"one" "two" "three" "test" "me"

- ".*"
- ".*? "
- ".*+"

ПРИМЕР

"one" "two" "three" "test" "me"

- ".*"

  "one" "two" "three" "test" "me"

- ".*? "

  "one"

- ".*+"

  Ничего!

ГРУППИРОВКА

- Круглые скобки используются для определения области действия и приоритета операций.
- Например, выражение (тр[ау]м-?)* найдёт последовательность вида трам-трам-трумтрам-трум-трамтрум.

ГРУППИРОВКА С ОБРАТНОЙ СВЯЗЬЮ

- При обработке выражения подстро́ки, найденные по шаблону внутри группы, сохраняются в отдельной области памяти и получают номер начиная с единицы.
- Каждой подстроке соответствует пара скобок в регулярном выражении.
- Квантификация группы не влияет на сохранённый результат, то есть сохраняется лишь первое вхождение.

ПРИМЕР

- Пример:

(та|ту)-\1

- Найдет:

та-та или ту-ту, но не та-ту

ГРУППИРОВКА БЕЗ ОБРАТНОЙ СВЯЗИ

(?:*шаблон*)

- Под результат такой группировки не выделяется отдельная область памяти и, соответственно, ей не назначается номер.
- Это положительно влияет на скорость выполнения выражения .

АТОМАРНАЯ ГРУППИРОВКА

(?>*шаблон*)

- Не создает обратных связей.
- Такая группировка запрещает возвращаться назад по строке, если часть шаблона уже найдена.

a(?>bc|b|x)cc	abccaxcc, но не abccaxcc
a(?>x*)xa	не найдётся axxxa

НАПОМИНАНИЕ

- Существуют три типа регулярных выражений:
  - BRE
  - ERE
  - PCRE

BASIC REGULAR EXPRESSIONS

- Все символы трактуются буквально, исключая перечень в таблице
- \(.*[.,]\)*
  - Точка в [ ] и вне трактуется по-разному
  - ( ) и { } в качестве синтаксического элемента необходимо предварять “\”
- ([0-9]*\.[0-9]*\$)
  - Чтобы искать собственно точку, доллар и пр. метасимволы, их нужно предварять “\”
  - ( ) и { } без “\” – ищет сами символы скобок!

EXTENDED REGULAR EXPRESSIONS

- **Добавлено:**
  - ? + |
- **Исключено:**
- \n – из-за высокой вычислительной стоимости
- **Изменено:**
- Символы скобок ( ) { } как синтаксические элементы не требуют “\” перед собой, для поиска самих этих символов “\” теперь нужен.

ВЫРАЖЕНИЯ В СТИЛЕ PERL

- Ленивые квантификаторы: *?, +?, ??
- Сверхжадные квантификаторы: *+, ++, ?+
- Сокращенные записи символьных классов: \w, \W, \s,\S, …
- Lookaheads и lookbehinds – подсказки алгоритму поиска
- Именованные группы связывания(named capture groups)
- Рекурсивные шаблоны.

GREP

- В ed для любой правки нужно ввести команду
- Одной командой пользовались часто:
- g/регулярное выражение/p – найти и напечатать строки, соответствующие выражению
- Для этой задачи сделали отдельную программу – grep.

GREP

- grep [options] PATTERN [FILE...]
- grep 'регулярное выражение' 'файл‘
  - grep -E '^(bat|Bat|cat|Cat)' heroes.txt
  - grep -i -E '^(bat|cat)' heroes.txt
- cat 'файл' | grep 'регулярное выражение‘
  - cat heroes.txt | grep -E '^[bcBC]at‘

GREP. ПРИМЕРЫ

Вывод имен файлов, содержащих строки, соответствующие шаблону:

- $ grep **–l** –E ‘^conf' /etc/*

То же самое, но включая подкаталоги:

- $ grep **–l –r** –E ‘^conf' /etc/*

SED

- sed [OPTION]... {script} [input-file]..
- sed – “Stream EDitor”
- Читает входной поток строка за строкой, на лету изменяя его в соответствии со скриптом.
- Язык sed имеет всего около дюжины команд, но хитрости их применения достойны целой книги

ЗАПУСК SED

- Для работы sed необходим скрипт. Его можно передать тремя способами:
  - sed -e *script* [input-file]
  - sed -f *script-file* [input-file]
  - sed [options] *script* [input-file]
- В последнем случае скриптом считается первый аргумент, не являющийся параметром ключа

РАБОТА SED

- Sed построчно прочитывает весь вход один раз.
- К каждой строке поочередно применяется одна и та же последовательность команд, записанная в скрипте
- Результат направляется в stdout, если sed был запущен с ключом -i, то записывается поверх исходного файла

РАБОТА SED

- Sed имеет два буфера для данных:
  - Pattern space – основной
  - Hold space – дополнительный
- Команды оперируют их содержимым.
- Каждая вновь прочитанная строка входа автоматически записывается в *pattern space*. На вывод подается то, что в нем оказалось в конце работы скрипта.

ПРИМЕР

- Команда “=” добавляет номер в начало строки

```
$ sed -e '=' helloworld.cpp
1 #include <cstdio>
2 void main() {
3     printf("Hello, world\n");
4 }
```

ПРИМЕР

Команда “d” очищает *pattern space* и заставляет прочитать следующую строку входа

- $ sed -e 'd' helloworld.cpp

(вывод пуст – pattern space каждый раз очищается)

ПРИМЕР

Можно указать номер строки, к которой применяется команда:

$ sed -e '3d' helloworld.cpp

```
#include <cstdio>
void main() {
}
```

ПРИМЕР

Диапазон строк:

- *$ sed -e '2,4d' helloworld.cpp*
- #include <cstdio>

В файле осталась только первая строка.

ПРИМЕР

Адресация по регулярному выражению:

- *$ sed -e '/{/,/}/d' helloworld.cpp*
- #include <cstdio>

Регулярное выражение должно быть окружено косыми чертами: “*/ regexp /*”

- Sed по умолчанию ожидает регулярные выражения в синтаксисе BRE. Если вызвать с ключом -r, sed будет интерпретировать их как ERE.

ЗАМЕНА ТЕКСТА

- *$ sed -e 's/@/ at /' emails.txt*

Было: john.doe@example.com
Стало: john.doe at example.com

Важно! Команда “s” в таком виде применяется к строке только один раз в том месте, где нашлось первое соответствие выражению.

Чтобы заменить все соответствия в строке, нужно добавить “g” (global):

- *$ sed -e 's/@/ at /g' emails.txt*

ЗАМЕНА ТЕКСТА &

$ sed -r -e 's/[0-9]+:[0-9]+:[0-9]+/& UTC/' times.txt

Было: 21:16:15

Стало: 21:16:15 UTC

“&” заменяется найденной подстрокой

ЗАМЕНА ТЕКСТА \1..\9

$ sed -r -e 's/([0-9]+):([0-9]+):([0-9]+)/\1 hours \2 minutes \3 seconds/' times.txt

Было: 21:16:15

Стало: 21 hours 16 minutes 15 seconds

\n заменяется n-ой группой связывания

ВСТАВКА СТРОК

a \text

Вставляет text ниже текущей строки (append)

i \text

Вставляет text выше текущей строки (insert)

c \text

Вставляет text вместо текущей строки

Пример:$ sed -e 'a \ \n' readme.txt

Команда вставляет дополнительный перенос строки в конце каждой из строк

ПРО HOLD SPACE

$ sed -e '1!G;h;$!d' forward.txt > backward.txt

Команда переставляет строки файла в обратном порядке

1!G – для каждой строки, кроме первой, дописывает содержимое hold space в конец patten space

h – копирует содержимое pattern space в hold space

$!d – применяет “d” ко всем строкам, кроме последней

Итого, по завершении работы, в pattern space содержится “перевернутый” текст

ЗАДАНИЯ НА ДОМ

1. Распознавать MAC-адрес короче, чем было написано в презентации.
2. Написать regexp для разбора ip-адреса. Написать надо именно команду (cat file | grep...)
3. Имя, Фамилия, Телефон -- телефонная книжка в csv. Преобразовать в html, который запустится в браузере. Только с помощью sed!

ЗАДАНИЯ НА ДОМ

5. Файл file.c, вывести все хедеры (только имена самих библиотек).
6. Утилита /sbin/ifconfig, выводит названия интерфейсов и их параметры. Все IP-адреса всех интерфейсов заменить на xxx.x.x.x, каждый икс соответствует одной цифре в IP-адресе. Разделить интерфейсы строкой дефисов.

ЗАДАНИЯ НА СЕЙЧАС

1. Из файла написанного, на C вытащить все строковые константы.
2. См. выше, но строки не должны входить в комментарии