Perl. Строки.

Введение.

Строка — последовательность символов.
В строки возможно включать любые символы.

Самая короткая строка — пустая (нет символов).
Длина самой длинной строки ограничена лишь доступной памятью.

Литеральные представления строк:

- в одинарных кавычках,
- в двойных кавычках.

Perl. Строки.

Литеральные представления. Одинарная кавычка.

Строки в одинарных кавычках.

Примеры:

```
'hello'
''
'O\'Brien'
'abc\\'
'no new line here\n'
'line one
line two'
```

Замечания:

Сами кавычки — не часть строки.

Все символы между ними действительны.

Символ косой черты имеет спецзначение только перед кавычкой и другой косой чертой.

Perl. Строки.

Литеральные представления. Двойная кавычка.

Строки в двойных кавычках.
Примеры:

```
"hello"
""
"quotation:\"text here\""
"abc\\"
"new line here\n"
"field1\tfield2\n"
```

Замечания:
Существует большое количество управляющих последовательностей.
Строка без управляющих последовательностей идентична в обоих литеральных представлениях.

Perl. Строки.

Управляющие последовательности.

`\r`	Возврат к началу строки
`\f`	Переход к новой странице
`\b`	Backspace
`\004`	Восьмеричное ASCII-значение
`\x0a`	Шестнадцатеричное ASCII-значение
`\l`	Вывод следующей буквы в нижнем регистре
`\L`	Вывод всех следующих букв в нижнем регистре
`\u`	Вывод следующей буквы в верхнем регистре
`\U`	Вывод всех следующих букв в верхнем регистре
`\Q`	Экранировать последующие символы
`\E`	Прекратить действие \L, \U и \Q

Замечание:
Есть некоторые другие. См. документацию.

Perl. Строки.

Простейшие операции.

Сравнение строк.
Операторы: eq, ne, lt, gt, le, ge.
Замечание:
Осторожно, в BASH эти операторы сравнивают числа!

Конкатенация.
Производится оператором «точка».

```
'hello'.' '."world\n" #"hello world\n"
```

Повторение строки.
Производится оператором «x».

```
"hi" x (2 + 2)          # "hihihihi"
7 x 3                   # "777"
```

Perl. Строки.

Преобразование строка <> число.

Perl выполняет преобразование автоматически.

```
a + b                  # число
a.b                    # строка
```

При преобразовании из строки в число игнорируются ведущие пробелы и нечисловые окончания строк

```
1 + "  17.5abc"   # 18.5
100 * "hello"     # 0
```

Для преобразования строк, содержащих числа в других системах счисления предназначены функции hex и oct.

В строку — конкатенацией.

```
"".125                 # "125"
```

Perl. Строки.

Интерполяция скаляров в строках.

Для строковых литералов в двойных кавычках производится интерполяция переменных.
Интерполяция — подстановка значения переменной в строку вместо её имени.

Синтаксис имени переменной в строке, как в BASH.

```
$v = 'w';
$v2 = "h, $v!\n";       # теперь v2 = "h, w!\n"
print "h, ${v}!\n";     # другой синтаксис
print 'h,'.$v."!\n";    # без интерполяции
print $v;# интерполяции нет за её ненадобностью
```

Замечание:
Переменная с undef значением заменяется пустой строкой.

Perl. Строки.

Интерполяция скаляров в строках.

Интерполяция производится до применения управляющих последовательностей.

```
$v = 'c\n';
$v2 = "\Ua${v}b";    # v2 = "AC\nB"
$v3 = "a\Q${v}\Eb"; # v3 = "ac\\nb"
```

Интерполяция производится однопроходно.

```
$v = '$a';
$v2 = "I am $v";     # v2 = "I am $a"
$v3 = "\$v";         # v3 = "$v"
```

Замечание:
Интерпретатор считает именем переменной наибольшую возможную последовательность символов. Используем {}.

Perl. Строки.

Отступление. Переменная по умолчанию.

В Perl существует переменная по умолчанию $_, которая автоматически используется во многих случаях, когда явно не задано применение другой переменной или значения.

```
chomp;          # взятие значения переменной $_ и
                  запись результата в $_
print;          # вывод значения $_
```

Замечание:
Есть и другие переменные по умолчанию, некоторые из них будут рассмотрены далее.

Perl. Строки.

Ввод. Оператор <STDIN> стандартного ввода.

Построчное чтение со стандартного ввода.

```
print while <STDIN>;
while (<STDIN>) {
   print;
}
while(defined($_ = <STDIN>)) {
   print $_;
}
```

Замечание:
Короткие формы записи можно использовать, если в условии цикла нет ничего, кроме оператора ввода <STDIN>

Perl. Строки.

Ввод. Оператор <> «ромб».

Построчное чтение данных из файлов, указанных как параметры командной строки при запуске скрипта.

```
$ ./myscript.pl file1 file2
```

Синтаксис:

```
while (<>) {
   chomp && print;# вывод без переводов строк
}
```

Замечания:

При отсутствии файлов чтение идет со стандартного ввода.

При достижении конца данных в потоке ввода оператор <> возвратит undef.

В случае ошибки чтения файла выводится предупреждение и начинается обработка следующего файла.

Perl. Строки.

Отступление. Chop и chomp.

Встроенная функция chop удаляет из строкового значения переменной последний символ, возвращаемое значение — этот самый отброшенный символ:

```
$x = "abc";
$y = chop($x);     # $x = "ab", $y = "c"
$x = "";
chop($x);          # $x = "", ошибки нет
```

Функция chomp удаляет символ перевода строки в конце:

```
$x = "abc\n";
chomp($x);         # $x = "abc"
chomp($x);         # $x = "abc"
chomp($a = <STDIN>);
```

Perl. Строки.

Ввод. Дескрипторы файлов.

Дескриптор — именованный канал ввода/вывода.
Зарезервированные имена:

```
STDIN, STDOUT, STDERR, DATA, ARGV, ARGVOUT
```

Открытие файлов, связь дескриптора с файлом:

```
open CONFIG, "my.conf";     # read
open CONFIG, "<my.conf";    # read
open RESULT, ">$resfile";   # write
open LOG, ">> logfile";     # append
```

Закрытие дескриптора файла:

```
close RESULT;
```

Perl. Строки.

Ввод. Дескрипторы файлов. Использование.

Открытие файла и выход в случае ошибки:

```
open CONFIG, "my.conf" or die "Error: $!";
while (<CONFIG>) {
   chomp;
   do_some_work();
}
```

Повторное открытие дескриптора файла:

```
open STDERR, ">logfile";
```

Замечание:
При повторном открытии дескриптора файла Perl сначала автоматически его закроет, если надо.

Perl. Строки.

Вывод. Оператор print.

Вывод данных по имени дескриптора, открытого на запись:

```
print DESCRIPTORNAME "text here";
```

Изменение дескриптора на вывод по умолчанию:

```
select DESCRIPTORNAME;
print "text here";# вывод в DESCRIPTORNAME
```

Замечания:

В Perl есть встроенные функции семейства printf, позволяющие производить форматированный вывод в стиле языка C.

Также в Perl существуют шаблоны написания отчетов (форматы), задающиеся специальными конструкциями.

Perl. Строки.

Шаблоны. Сопоставление с шаблоном.

Синтаксис шаблона:

`/temlate/modifiers` - сокращенная запись

```
m/other template/modifiers
```

В некоторых случаях удобно сменить символ-разделитель:

```
/^\/usr\/bin\/perl\b/
m@^/usr/bin/perl\b@
m{^usr/bin/perl\b}
```

Модификаторы:

`/hello/i` — без учета регистра

`/^a.*z$/s` — считать перевод строки обычным символом

Perl. Строки.

Шаблоны. Связывание шаблона с переменной.

По умолчанию шаблон сопоставляется с переменной $_

```
chomp($_=<STDIN>);
$number = $_ if /^[0-9]+$/;
```

Чтобы сравнить другую переменную с шаблоном необходимо произвести их связывание:

```
my $var = "long long text";
print "Match found" if $var =~ /long/;
```

Интерполяция переменных в шаблонах происходит аналогично интерполяции переменных в строковых литералах:

```
my $what = shift @ARGV;
/\Q$what\E/ and print while <>;
```

Perl. Строки.

Шаблоны. Замена по шаблону.

Синтаксис шаблона:

```
s/template/substitute/modifiers
```

В некоторых случаях удобно сменить символ-разделитель:

```
s/^https:\/\//http:\/\//
s@^https://@http://@
s{^https://}#http://#
```

Модификаторы:

`s/^a.*z$/found/is` — так же, как и при сопоставлении

`s/one/two/g` — глобальная замена (всех совпадений)

`s/one/two/e` — считать two Perl-выражением (eval)

Оператор связывания такой же. Интерполяция так же.

Perl. Строки.

Шаблоны. Переменные для совпадающих с шаблоном данных.

Внутри регулярного выражения есть обратные ссылки:

```
s/^(foo|bar)xyz(.*)$/\1:\2/i
```

После завершения сопоставления с шаблоном в переменных вида $n хранятся строки-совпадения.

```
$_ = "one, two, three";
print $1 if /\s(\w+),/;     # two
```

Замечание:
Переменные не изменяют своих значений до следующего успешного совпадения с каким-либо шаблоном.
Следующий синтаксис (?:expression) позволяет выделять часть регулярного выражения в шаблоне без запоминания.

Perl. Строки.

Шаблоны. Автоматически создаваемые переменные сравнения.

Часть строки, совпавшая с шаблоном, сохраняется в $&, предшествующая совпавшему фрагменту — в $`, следующая за ним — в $', последняя запомненная подстрока — в $+:

```
if ('one, two four' =~ /,\s(\w+)\s/) {
   print $1;                  # two
   print "($`)($&)($')"; # (one)(, two )(four)
}
```

Еще пример, с заменой по шаблону:

```
$_  = "that is it";
s/(\w+)/<$1>/g; # $_ = "<that><is><it>"
```

Perl. Строки.

Шаблоны. Дополнительно.

Модификатор /o указывает Perl, что данный шаблон следует компилировать только один раз:

```
s/(\w+)/$word/o
```

Символьные классы, сокращенные обозначения:

```
\d = [0-9], \D = [^\d],
\w = [A-Za-z0-9_], \W = [^\w],
\s = [\f\t\n\r ], \S = [^\s]
```

Фиксирующие директивы:

`^`, `$` — начало, конец строки,

`\b`, `\B` — граница слова/нет границы слова.

Perl. Строки.

Шаблоны. Транслитерация.

Синтаксис: tr/old symbols/new symbols/modifiers

```
tr/ /_/;
$count = tr/a-zA-Z//;# число букв
```

Модификаторы:
/d — удалять символы, для которых нет замены
/c — заменять символы не входящие в старый набор
/s — удалять повторяющиеся символы при замене
/U, /C — перекодировать в юникод/однобайтовую кодировку

Автоматической интерполяции переменных нет.

```
eval "tr/$oldlist/$newlist/";
```

Perl. Строки.

Шаблоны. Транслитерация. Гениальный скрипт.

```
#!/usr/bin/perl
tr/a-zA-Z/A-Za-z/ and print while <>;
```

Perl. Строки.

Поиск подстроки по индексу.

Для поиска экземпляра подстроки в строке в Perl можно использовать функции index и rindex:

```
$x = index($str, $substr);          # первый
$y = index($str, $substr, $x+1);# следующий
$i = rindex($str,$substr);          # последний
$j = rindex($str,$substr, $i-1);# предпоследний
```

Замечания:
Возвращаемый индекс начала подстроки отсчитывается от 0; если подстрока не найдена, возвращается -1.
Третий необязательный параметр функций указывает номер позиции, начиная с которой ведется поиск (для index) или вплоть до которой ведется поиск (для rindex).

Perl. Строки.

Манипулирование подстрокой.

Оператор substr работает с частью строки традиционно:

```
$part = substr($str, $pos, $len);
$part = substr $str, $pos;# до конца строки
$other = substr($str, $pos, $len, $subst);
```

И нетрадиционно.
Замена выбранной части для строки-переменной:

```
substr($str, 0, 5) = "new symbols";
```

Обращение к символу строки по индексу:

```
substr($str, $index, 1) = $char;
```

Связывание только с частью строки:

```
substr($str, -50) =~ s/\bone\b/two/g;
```

Perl. Строки.

Дополнительно.

Оператор split разбивает строку по шаблону-разделителю:

```
@fields = split /\s+/, "This is  a\ttest";
```

Оператор join, напротив, делает из массива строку:

```
$result = join ":", @fields;
```