Perl - SYNOPSIS

Version: 1.00 build 0005
Date: 02.04.2007
Author: Sergey Lepenkov (Serż Minus) <minus@mail333.com>

Содержание

Типы данных. Переменные

«Символы» типов. Префиксы

<префикс>ИМЯ_ПЕРЕМЕННОЙ

префиксы:

**$** — «scalar» ($a). Скаляр (число/строка/ссылка). Переменная в единственном числе.

@ — «array» (@a). Группа скаляров с числовым ключом. Именованный список скаляров.

% — «hash» (%a). Ассоциативные массивы с ключом в виде строкового литерала (строки). Хэши

& — «subprogram» (&a). Именованный блок кода, который может быть вызван. При вызове подпрограммы можно опускать этот префикс, но следует указывать явные аргументы, даже в случае их отсутствия, например: postprint($a,$b) или postprintf()

***** — «typeglob» (*a). Полный набор имен с именем ИМЯ_ПЕРЕМЕННОЙ (в нашем случае *a содержит в себе ссылку на список, содержащий переменные: $a, @a, %a, &a и a). Используется для передачи ссылок на заголовок файла «handle»

«handle» (a). Без префикса лексема понимается как имя заголовка файла

Область видимости (namespace). Пакеты

package NAMESPACE – Объявление того, что с данного места начинается область видимости с именем NAMESPACE. Доступ к пакетам текущего модуля можно получить, используя в качестве префикса конструкцию:

"<префикс_типа>NAMESPACE::<имя>"

Например: $main::VERSION

Код вашей программы имеет предопределенное имя (пакет) **main**. Но конструкцию *package main* можно не использовать, т.к. она подразумевается по умолчанию.

Более подробно описано в разделе **«Модули. Прагмы»**.

my EXPR – лексическая переменная. Область действия распространяется только на уровень пакета или процедуры (подпрограммы).

local EXPR – динамическая, локальная. Область действия аналогична **my**, но переменная сохраняет свои значения между вызовами. Предпочтительней использовать **my**.

our EXPR – условно-глобальная. Действие распространено на уровне модуля. Для передачи скалярных констант в виде простых строковых или числовых литералов, можно использовать переменные окружения в глобальном массиве **@ENV**.

```
my $a;
my ($a,$b);
my @a;
my %a;
my (@a,@b,$c,%e)
```

Внутренние переменные

$_ — переменная по умолчанию. Эта переменная принимает результаты вызова процедур и функций, участвует во всех процедурах как буфер значений по умолчанию.

@_ — массив значений по умолчанию. Как правило, содержит список аргументов вызова процедур.

$1…n — содержимое скобок **( )** в регулярных выражениях.
n – порядковый номер скобок, учитывая вложенность.
См. раздел «Регулярные выражения»

$&**, **$+, **$`**, **$'** – обратные ссылки в регулярных выражениях.
$& — найденная подстрока по последнему шаблону;
$` — подстрока до $&;
$' — подстрока после $&;
$+ — подстрока найденная с помощью шаблона «или».
См. раздел **«Регулярные выражения»**

$* — признак многострочности. **0** – однострочная; **1** – многострочная

$. — номер прочитанной строки файла. **0** – конец файла

$/ — признак завершения строки (перевод строки "**\n**")

$| — флаг сброса буфера вывода при каждой операции вывода. Значение по умолчанию: **FALSE**

$, — символ разделения элементов массивов при обращении к нему с помощью print **@a**

$" — символ разделения элементов массивов при обращении к нему с помощью *print "@a"*

$ — символ, который вставляется в конец результата *print @a* или *print "@a"*. Следует использовать с осторожностью.

$; — символ разделения ключей от значений хэша (**\034**)

$# — формат вывода чисел по умолчанию

$% — формат вывода номеров страниц по умолчанию

$= — длина одной страницы (количество строк 60)
$- — количество оставшихся строк на странице

$~ — имя формата вывода текущей страницы. Имя указателя
$^ — имя текущего формата для вывода заголовка страницы

$^L — символ смены листа. По умолчанию **\f**
$^A — аккумулятор для команды **formline()**

$: — символ переноса строки для многострочной строки

$? — статус завершения дочернего процесса **system()**, **wait()** и др.

$! — имя возникшей ошибки. Результат команды **die()**

$@ — ошибка выполнения в команде **eval()**

$$ — номер текущего процесса

$0 — имя файла текущего процесса

$<**, **$> — реальный и эффективный UID процесса (user)

$(**, **$) — реальный и эффективный GID процесса (group)

$[— номер первого индекса массива. По умолчанию: 0

$] – версия текущего интерпретатора Perl (*usr/bin/perl -v*)

$^T — время старта текущей программы в секундах, считая с начала 1970 года.

$^W — **TRUE**, если Perl запущен с флагом **–w**, иначе **FALSE**

$^X — команда запуска Perl-интерпретатора

**$^I** — текущее значение inplace-edit возможности. Для отключения используйте присвоение *$^I =undef()*

$^P — внутренний флаг отладки. Применяется для того, чтобы отладчик не отслеживал самого себя.

$^F — номер максимального системного описателя файлов (system file descriptor). Обычно равен **2**

$^D — TRUE, если Perl запущен с флагом **–d**, иначе **FALSE**

@ARGV — список параметров вызова сценария

@INC — список директорий диска которые просматривает Perl для выполнения команд **do()**, **require()** или **use()**.

%INC — этот хэш содержит имена директорий для имен использованных файлов командами **do()** или **require()**. Ключ — имя файла, а значение — директория.

@ISA — список подпрограмм пакетов

%ENV — список переменных окружения

%SIG — список обработчиков системных сигналов

Кавычки и интерполяция

‘ ’, **q( )** — ограниченная интерполяция (только **\'** и \\)
“ ”, **qq( )** — стандартная интерполяция

@a = **qw()** – использовать разделитель “пробелы” для перевода слов в список.

qr() – запись регулярного выражения в переменную

` ` (грависы), **qx()** – запись команды для исполнения.
Грависы определяют команду относительно ОС, когда как **qx()** определяет команды относительно интерпретатора Perl.

```
$rex = qr/my.STRING/is;
s/$rex/foo/;

$output = `cmd 3>&1 1>&2 2>&3 3>&-`;
$output = qx/cmd 3>&1 1>&2 2>&3 3>&-/;
```

q( ) и другие **q-операторы** (generic) могут вместо **( )** иметь аналогичные последовательности: **! !**, / /, **[ ]**, **{ }**, # # и др.

Интерполяция

Интерполируются только переменные с префиксом **$** и **@**, а также ряд управляющих метасимволов (escape)

```
my $a = “world!”;
print “Hello $a”;
```

Интерполяция массивов приводит к выводу всего массива с разделителем. Разделители приведены в разделе **«Внутренние переменные»**: **$.** и **$”**.

Интерполяция осуществляется главным образом в термах **“ ”** и их аналогах. См. таблицу:

Customary	Generic	Meaning	Interpolates
' '	**q{}**	Literal	No
" "	**qq{}**	Literal	Yes

` `	**qx{}**	Command	Yes*
	qw{}	Word list	No
/ /	**m{}**	Pattern match	Yes*
	qr{}	Pattern	Yes*
	s{}{}	Substitution	Yes*
	tr{}{}	Transliteration	No(см. ниже)
<<EOF		here-doc	Yes*

* за исключением разделителя ' '

Управляющие метасимволы. Escape

Глобально интерполируемые метасимволы (и в **'** и в **"**):

\' – одинарная кавычка
\\ — знак интерполяции «backslash»

Метасимволы стандартной интерполяции (только в **"**)

\n – новая строка LF (0A)
\r – возврат каретки CR (0D)
\t – табуляция (09)
\s – пробельный символ «Spacebar»(20)
\b – нажатие на «Backspace» (0B)
\f – конец страницы печати «Form Feed» (0C)
\a – сигнал «Beep» (07)
\e – следующий символ является escape-символом (1B)

\033 – восьмеричный символ
\x1B – шестнадцатеричный символ
\x{263a} — шестнадцатеричный символ (смайлик)
\N{name} — именованный символ

\" – двойная кавычка
\$ — знак «dollar» (**$**)
\@ — знак «commercial at» (**@**)
\% — знак «percent sign» (**%**)

\u – перевод последующего символа в верхний регистр

\U – перевод последующей подстроки до **\E** в верхний регистр
\l – перевод последующего символа в нижний регистр
\L – перевод последующей подстроки до **\E** в нижний регистр
\Q – все последующие, вплоть до символа **\E**, не буквенно-цифровые символы экранировать обратной косой чертой “\”
\E – отмена действия **\Q**,**\U** или **\L**

Литералы V-строк

```
$a=v13.10; # 2 символа: CRLF
$a=v1.20.40.50; # chr(1).chr(20).chr(40).chr(50)
$a=1.20.40.50; # v1.20.40.50. т.к. много десятичных
разделителей
$ip=127.0.0.1; # IP
```

Ссылки

Жесткие ссылки – это скаляры, содержащие ссылки на переменные

Символические ссылки – это скаляры, содержащие имена других переменных

Создание жестких ссылок

\$NAME – ссылка на скалярную переменную
\@NAME – ссылка на списковую переменную
\(LIST) – ссылка на список
\&NAME – ссылка на подпрограмму
***NAME** – ссылка на typeglob/handle
[LIST] – ссылка на анонимный массив
{LIST} – ссылка на анонимный хэш

```
$a=[1,2,['a','b','c']]
$b = $a->[2]->[2]; # возвращается значение 'c'

$a = sub {print "hello",$_[0]?$_[0]:''}; # ссылки на
подпрограммы
$b=&$a; # выводит hello
$b=$a->(' world!'); # выводит hello world

$a = bless \$b,$c; # ссылка на объект
$b = $a->method(); # доступ к методам объекта

$a = *foo{SCALAR}; # аналогично \$foo
```

```
$a = *foo{ARRAY}; # аналогично \@foo
$a = *foo{HASH}; # аналогично \%foo
$a = *foo{CODE}; # аналогично \&foo
$a = *foo{GLOB}; # аналогично \*foo
$a = *foo{IO}; # аналогично ссылке на STDIN и STDOUT
$a = *foo{FILEHANDLE}; # аналогично *foo{IO}
```

Разыменовывание

```
$a = \@a; # создание ссылки
@b=@$a; # разыменовывание ссылки
@$a{$b,$c}=($d,$e); # добавление еще двух пар значений
push(@{$a},$b); # добавление $b к массиву с именем $a
```

Символические ссылки

```
$name=”bam”
$$name = 1; # $bam = 1
$name->[0] = 4; # $bam[0] = 4
$name->{X} = ‘A’; # $bam{X} = ‘A’
@$name = (); # @bam =()
&$name; # &bam;
```

Дескрипторы

Обращение ко всему содержимому, на которое указывает дескриптор, осуществляется используя оператор **< >**. Например: **<STDIN>**, где STDIN – дескриптор.

Предварительно открытые дескрипторы:

STDOUT – стандартный канал вывода. Либо «экран», либо HTTP-вывод, либо что-то иное.

STDIN – стандартный канал входа. Либо клавиатура, либо переменная окружения, устанавливаемая сервером Apache, либо что-то иное.

STDERR – стандартный канал вывода ошибок. Создан для возможности добавления записей в log-файл пользовательской информации

Передача дескрипторов посредством ссылок:

```
my $fhlink = *STDOUT;
my $fhlink = \*STDOUT;
```

использование ссылок на дескрипторы:

```
sub newopen {
       local *FH;
       open FH, $_ or return undef;
       return *FH
}
my $fh= newopen('filename');
```

Внутренние литеральные лексемы.

Не интерполируются лексемы:

__LINE__ — текущий номер строки
__FILE__ — имя файла в данной точке кода
__PACKAGE__ — имя текущего пакета package
__END__ — любой символ следующий за этим литералом игнорируется вплоть до конца файла программы. Но это содержимое можно прочесть с помощью заголовка **DATA**. Например: *@a=<DATA>*
__DATA__ — аналог **__END__**, но действует в рамках текущего пакета.

Контекст

```
$var1 = <>;  # Прочитать одну строку файла
@var1 = <>;  # Прочитать все строки файла в массив @var1
$var1 = (1,2,3); # $var = 3 - количество элементов
@var1 = (1,2,3); # Создание массива @var1 с элементами 1,2,3
```

Скалярный

```
$a = <>;
```

Списочный

```
@a = <>;
%a = <>;
($a,$b) = <>;
```

Булев (логический)

```
if ($a == $b) BLOCK
while (@a) BLOCK
```

и т.д.

Пустой контекст
Пустой контекст приводит к ошибке!

Интерполирующий
```
$a="$b@a"; # интерполирующий контекст.
```

Отличие между списковым и интерполирующим контекстом видно из примера использования внутренних переменных **$.** и **$"**. См. раздел **«Встроенные переменные»**

Операторы

Унарные операторы: префиксные операторы. Кроме пост-инкрементирования и пост-декрементирования. Все остальные – инфиксные (по обе стороны пре-пост).

! @a – унарный оператор **!**
$a * $b – бинарный оператор ***
$a?$b:$c – тернарный оператор **?:**
print $a, $b, $c – списковый оператор **print**

Приоритеты и ассоциативность

Таблица ассоциативность. Операторы перечислены в порядке уменьшения приоритета.

Ассоциативность	Оператор
left	Термы и списковые операторы (leftward)
left	->
nonassoc	++ --
right	**
right	! ~ \ and unary + and -
left	=~ !~
left	* / % x
left	+ - .
left	<< >>
nonassoc	Встроенные именованные (унарные)
nonassoc	< > <= >= lt gt le ge
nonassoc	== != <=> eq ne cmp
left	&
left	| ^
left	&&
left	||
nonassoc	.. ...
right	?:
right	= += -= *= и т.д.
left	, =>

nonassoc	списковые операторы (rightward)
right	not
left	and
left	or xor

2 + 3 * 4 – приоритет ***** больше приоритета **+**

$a * $b * $c # ($a * $b) * $c – левая ассоциативность (left)
$a ** $b ** $c # $a ** ($b ** $c) – правая ассоциативность (right)
2 != 3 != 4 – ошибка. Не ассоциативный оператор

Термы и списковые операторы

Списковые операторы “leftward”

print (1, 3, sort 4, 2) - вывод 1324
print $foo, exit – **exit** выполняется раньше **print**
print($foo), exit – **print** выполняется раньше **exit**

аналогично ведут себя **do{}**, **sub{}** и другие термы, такие как анонимные хэши **{ }**, массивы **[ ]**, подпрограммы **sub{}**.

Списковые операторы и термы стремятся проглотить все аргументы после самого оператора.

Списковые операторы “rightward”

К таким операторам относятся операторы **and**, **or**, **not** и **xor**. См. раздел **«Логические операторы»**.

Итеративные операторы ввода

< > - угловые скобки. Псевдолитерал. Опреатор осуществляет доступ к объекту дескриптора – файлу, потоку и т.д.

@a=<DATA>; # текст располагающийся после лексемы **__DATA__** или **__END__**

<> является синонимом **<ARGV>**

<filemask> - доступ к файлам папки по маске. Например: *foreach(<*.*>) {print $_."\n"}* выдаст все имена файлов текущего каталога. Это аналогично функции **glob()**, которая позволяет использовать структуру каталогов.

Операторы разыменовывания

${} – воссоздать объект ссылки в скалярном контексте
@{} – воссоздать объект ссылки в списковом контексте
%{} – воссоздать объект ссылки в списковом контексте хэша
&{} – воссоздать объект ссылки - процедуру

***{}** – воссоздать объект ссылки в контексте typeglob
Более подробно см. раздел **«Ссылки»**

Унарные операторы

Инкрементирование и декрементирование

Префиксные и постфиксные операторы автоинкрементирования и автодекрементирования.

++$a — преинкремент. Добавление 1 и использование
$a++ — постинкремент. Использование $a, затем прибавление 1
--$a — предекремент. Вычитание 1 и использование
$a-- — постдекремент. Использование $a, затем вычитание 1

```
print ++($a=99); # 100
print ++($a='Az'); # Ba
```

Идеографические операторы

! $a – отрицание. Аналог **NOT** в BASIC
- $a – отрицание. Например -1
~ $a – поразрядное отрицание. Равное **0** станет **1**, и наоборот
+ $a – например, +1
\ $a – создание ссылки

Именованные операторы

alarm, caller, chdir, chroot, cos, defined, delete, do, eval, exists, exit, gethostbyname, getnamebyname, getpgrp, getprotobyname, glob, gmtime, goto, hex, int, lc, lcfirst, length, localtime, lock, log, lstat, my, oct, ord, quotemeta, rand, readlink, ref, require, return, rmdir, scalar, sin, sleep, sqrt, srand, stat, uc, ucfirst, umask, undef

Более подробное описание см. в разделе **«Встроенные функции»**

```
sllep 4 | 3; # понимается как (sleep 4) | 3
```

Файловые операторы

-e $a – TRUE если файл $a существует

-S $a – TRUE если файл $a – это имя сокета sock
-d $a – TRUE если файл $a – это каталог
-l $a – TRUE если файл $a – это ссылка
-p $a – TRUE если файл $a – это конвейер pipe (FIFO)
-r $a – TRUE если файл $a доступен для чтения (UID)
-w $a – TRUE если файл $a доступен для записи (UID)
-f $a – TRUE если файл $a – это обычный файл
-x $a – TRUE если файл $a – это исполняемый файл (UID)
-o $a – TRUE если файл $a принадлежит UID
-u $a – TRUE если для файл $a установлен бит setuid
-g $a – TRUE если для файл $a установлен бит setgid
-k $a – TRUE если для файл $a установлен бит sticki bit
-t $a – TRUE если файл $a открыт на текущем терминале
-T $a – TRUE если файл $a – текстовый файл
-c $a – TRUE если файл $a – символьный файл
-b $a – TRUE если файл $a – блочный файл
-B $a – TRUE если файл $a – двоичный файл

-z $a – TRUE если файл $a пуст
-s $a – размер файла $a
-A $a – время доступа к файлу $a, дни
-M $a – время изменения файла $a, дни
-C $a – время создания файла $a, дни

-R, -W, -X, -O – тоже, что и **–r, -w, -x, -o** но относительно реального UID

```
print "size ($a): ".(-s $a) if –e $a; # размер существующего файла
```

Бинарные операторы

Оператор «граф» (->)

Бинарный оператор разыменовывания.

$link->method – доступ к методу объекта через ссылку на него
$link->{key} – доступ к значению хэша через ссылку на него
$link->[index] – доступ к значению массива через ссылку
$link->$link->... – многокаскадность обращения

Оператор «запятая» и «дограф» (=>)

Бинарные операторы разделения.

В скалярном контексте выполняется левый аргумент, результат игнорируется, затем правый и его результат есть результат действия оператора. В списковом контексте это разделитель элементов списка, который включает указанные элементы в список.

```
$a=(1,3); # $a = 3
@a=(1,3); # $@ = (1,3)
```

Дограф заставляет интерпретатор понять, что левый его аргумент – это строковое значение. По этой причине в хэшах доступен двоякий способ записи ключей (см. раздел **«Структуры данных»)**. Кроме того, он облегчает читабельность пар значений, например, в хэшах.

```
%hash = ( $key => $value ); # хэш
login( $username => $password ); # аргументы процедуры
```

Аддитивные операторы

$a + $b – сложение
$a - $b – вычитание
$a . $b – конкатенация строк. **$a** «склеивается» с **$b**.
“ab” – интерполяция. Аналог **$a . $b**

Мультипликативные операторы

$a * $b – умножение
$a / $b – деление
$a % $b – взятие по модулю (остаток от деления $a/$b) [= MOD]
$a x $b – повторение. $a повторяет $b раз

Степенные операторы

$a ** $b – возведение в степень (**$a** в степени **$b**)

Операторы связывания

=~ - связывание строковое выражение с поиском по шаблону
!~ - аналог =~, но возвращается логическое отрицание

Операторы сдвига

Поразрядный сдвиг левого аргумента на количество бит правого

1 << 4 – возвращает 16
32 >> 4 – возвращает 2

Операторы сравнения и равенства

== (eq) – равно (для строк в скобках)
!= (ne) – не равно
< (lt) – меньше
> (gt) — больше
<= (le) – меньше или равно
>= (ge) – больше или равно
<=> (cmp) – **0**, если равно, **1**, если **$a** больше, **-1**, если **$b** больше

Операторы поразрядного действия

$a & $b – побитное AND – конъюнкция

$a | $b – побитное OR – дизъюнкция
$a ^ $b – побитное XOR

$a	$b	AND	OR	XOR
T	T	T	T	F
T	F	F	T	T
F	T	F	T	T
F	F	F	F	F

Логические операторы

Операторы высокого приоритета

$a && $b – И - $a, если $a ложно, иначе $b
$a || $b – ИЛИ - $a, если $a истинно, иначе $b
! $b – НЕ – истинно, если $a ложно и наоборот

Операторы низкого приоритета

$a and $b – И - $a, если $a ложно, иначе $b
$a or $b – ИЛИ - $a, если $a истинно, иначе $b
not $b – НЕ – истинно, если $a ложно и наоборот
$a xor $b – хИЛИ – истинно, если $a или $b истинны, но не оба

```
open SESAME, "filename" or die "ошибка открытия файла: $!\n"
open(SESAME, "filename") || die "ошибка открытия файла: $!\n"
```

Операторы диапазона

.. – обычный оператор диапазона
… - без проверки правого операнда на достижение конца
if (101 .. 200) {print; } – вывести вторую сотню строк
for (101 .. 200) {print; } – вывести числа от 101 до 200
@a=@b[0..5] – срез массива
@a=('a'..'z') – от a до z
@a=('01'..'31') – от 01 до 31 с ведущими нулями

Операторы присваивания

l-value = expression
l-value operator= expression

что эквивалентно:
l-value = l-value operator expression

=, +=, -=, *=, /=, %=, **= - бинарные операторы
&=, |=, ^=, <<=, >>= - поразрядного действия
&&=, ||= - булевы операторы
.=, x= - строковые операторы

```
$a += $b; # $a становится равным результату $a+$b
$a ||= $b; # $a становится равным $b если $a = FALSE
$a=$b=$c=0; # обнуление всех переменных ряда
($t -= 32) *= 5/9; # перевод шкалы Фаренгейта в Цельсия
($t *=9/5) += 32; # перевод шкалы Цельсия в Фаренгейта
$a x= 80; # размножить $a до 80 раз
```

Условный оператор

Тернарный оператор **?:**

EXPR?STATEMENT1:STATEMENT2
См. раздел **«Управляющие структуры»**

@a=@b?@b:@c; # если массив **@b** возвращает истину в скалярном контексте, то массив **@a** принимает его значения. Иначе массив **@a** принимает значения массива **@c**, даже если последний возвращает ложь в скалярном контексте.

```
print "abc".(0?"def":"ghi"); # abcghi
```

Управляющие структуры

Простые выражения:

STATEMENT if EXPR
STATEMENT unless EXPR
STATEMENT while EXPR
STATEMENT until EXPR
STATEMENT foreach LIST

Здесь STATEMENT выполняется исходя из условий выполнения, определенные выражением EXPR.

Например:

```
print "hellow" if $a = = 1;
print "hellow\n" foreach qw(Abc Def Ghi);
```

Сложные выражения:

if (EXPR) BLOCK
if (EXPR) BLOCK else BLOCK
if (EXPR) BLOCK elsif (EXPR) BLOCK ... else BLOCK
LABEL while (EXPR) BLOCK
LABEL while (EXPR) BLOCK continue BLOCK
LABEL until (EXPR) BLOCK
LABEL until (EXPR) BLOCK continue BLOCK
LABEL for (EXPR; EXPR; EXPR) BLOCK
LABEL foreach VAR (LIST) BLOCK
LABEL foreach VAR (LIST) BLOCK continue BLOCK
LABEL BLOCK continue BLOCK

BLOCK – Это конструкция вида:

{
STATEMENTS
}

Истина и ложь

FALSE (false) – ложь. Ложью называется любое значение не равное **«0»** или **«»** (пусто), а также неопределенное значение.

TRUE (true) – правда. Всё, что является **НЕ** ложью.

```
if ($a) { # если $a TRUE
       print "hello world! :)"
}
```

0 – ложь
1 – истина
-1 – истина
10 – 10 – ложь
"0" – ложь
0.00 – ложь
"0.00" – истина
"0.00" + 0 – ложь
\$a – истина
undef() – ложь

Условные структуры

if (EXPR) BLOCK
if (EXPR) BLOCK else BLOCK
if (EXPR) BLOCK elsif (EXPR) BLOCK ... else BLOCK
STATEMENT if EXPR

unless (EXPR) BLOCK
unless (EXPR) BLOCK else BLOCK
STATEMENT unless EXPR

EXPR?STATEMENT1:STATEMENT2
Эта конструкция аналогична следующей

```
if (EXPR) {
        STATEMENT1
} else {
        STATEMENT2
}
```

if – условие «если истина»
unless – условие «если ложь»

Итеративные структуры

while/until – позволяют выполнить блок многократно

LABEL while (EXPR) BLOCK
LABEL while (EXPR) BLOCK continue BLOCK
LABEL until (EXPR) BLOCK
LABEL until (EXPR) BLOCK continue BLOCK
STATEMENT while EXPR
STATEMENT until EXPR

while – выполнение происходит пока EXPR истина
until – выполнение происходит пока EXPR ложь

next/last/redo – позволяют контролировать ход итерации. При использовании **while/until** вводится понятие блока **continue**, который выполняется сразу после завершения итерации, кроме выхода из итерации с помощью команд **last** и **redo**. Для итерационного механизма **for** блок **continue** не указывается.

next - Переходит к началу текущего цикла, блок **continue** выполняется и повторяется итерация

```
M1: while ($i < 6) {
       ++$i;
       next M1 if $i < 3;
       ++$i
} continue {
       print "$i "
}
```

Результат: 1 2 4 6

last - Немедленно прерывает цикл. Блок **continue** пропускается.

```
M1: while ($i < 6) {
       ++$i;
       last M1 if $i > 3;
       ++$i
} continue {
       print "$i "
}
```

Результат: 2 4

redo - начать новый цикл не вычисляя EXPR и не выполняя **continue** блок.

```
M1: while ($i < 6) {
       ++$i;
       redo M1 if $i == 3;
       ++$i
} continue {
       print "$i "
}
```

Результат: 2 5 7

for – стандартный механизм итераций

LABEL for (EXPR1; EXPR2; EXPR3) BLOCK

EXPR1,2,3 являются списковыми выражениями. Перед началом цикла выполняется EXPR1, если EXPR2 = true выполняется блок, затем выполняется EXPR3. Первый и третий EXPR можно опускать. Если не указывать и второй – то будет создана ситуация бесконечного цикла.

```
for ($i = 2; $i < 5; ++$i) {
       print $i, " ";
}
print " После цикла i = $i";
```

Результат: 2 3 4 После цикла i = 5

foreach – механизм аналогичный **for**, для списков и массивов

LABEL foreach VAR (LIST) BLOCK

```
foreach my $a (3,5,7) {
       print "$a ";
}
```

Результат: 3, 5, 7

label – метки участков кода. Именованные блоки

```
SWITCH:
       {
       if ($i ==1 ) { .....; last SWITCH; }
       if ($i ==2 ) { .....; last SWITCH; }
       if ($i ==3 ) { .....; last SWITCH; }
       $default = 13;
}
```

goto – переход к определенной label

goto LABEL
goto EXPR
goto &NAME

```
goto SWITCH; # переходит к LABEL с именем SWITCH
goto ("M1", "M2", "SWITCH")[2]; # EXPR вычисляет LABEL
goto &Camel; # Переход к подпрограмме на выполнение явно
```

Блоки. Подпрограммы

sub – объявление подпрограмм.

sub NAME PROTO :ATTRS – предопределение
sub NAME PROTO :ATTRS BLOCK – явное определение
sub PROTO :ATTRS BLOCK – анонимное определение

NAME – имя подпрограммы
PROTO – прототипы подпрограммы
ATTRS – атрибуты выполнения
BLOCK – блок кода { }

Использование процедур модуля MODULE
use MODULE qw/NAME1 NAME2 … NAMEn/

Вызов процедур:
NAME(LIST) – NAME не предопределена
NAME LIST – NAME предопределена или импортирована
&NAME(LIST) – универсальный механизм

Существуют встроенные предопределенные процедуры:
BEGIN, CHECK, INIT, END, AUTOLOAD, DESTROY и др.
См. раздел **«Стандартные методы»**

Эти процедуры можно использовать и переопределять как именованные блоки без **sub**. Например:

```
BEGIN {
       use CGI;
}
```

Все параметры принимаются в массив **@_**; по умолчанию, возвращается результат выполнения последней команды, либо результат команды **return()**.

return EXPR – возврат из процедуры с результатом EXPR

Прототипы

Прототипы позволяют предопределить типы переменных, которые ожидаются процедурой как функцией.

Префикс **&** при вызове подпрограммы отключает действие прототипов.

$ - SCALAR – скалярная переменная.
@ - LIST – список переменных
% - HASH – список переменных (как и **@**)
& - SUBPROGRAM – блок
***** - HANDLE – заголовок файла
\$- исключительно переменная с префиксом **$**
\@ - исключительно массив с префиксом **@**
\% - исключительно хэш с префиксом **%**
; - отделяет обязательные прототипы от необязательных, например:

```
sub mysub ($;&); # mysub("A") или mysub ("A", {/foo/})
sub mysub (\$\%;$); # mysub($a,%a,[1..100])
sub mysub (); # mysub можно использовать как любую
переменную-константу, например time()
```

Атрибуты

locked и **method**

sub func : locked { … } – допускается только один поток
sub func : locked method { … } – ~ в рамках данного объекта

```
sub func (): lvalue { $a; } - возврат модифицированного
lvalue
func = 5; # 5
```

Модули. Прагмы

Прагмы – это модули, влияющие на ход компиляции текущего пакета. Самые распространенные из них:

use warnings (ключ **–w**) – записывать все замечания в лог
use strict – аналогично warnings, но компиляция прекращается
use vars – позволяет экспортировать переменные пакета
use constant – работа с константами
use integer – работа с целыми числами

use позволяет задействовать прагму
no – выключить.

При use выполняется метод **import()** а при no – **unimport()** указанной прагмы. См. далее.

Подключение

Все модули имеют имя **MODULE**, а имена файлов, в которых располагаются модули: **MODULE.pm**

Файлы, в которых располагаются модули, имеют название, совпадающее с именем модуля и дополненное расширением **.pm**

Если модули размещены в подкаталогах, то каждый каталог разделяется знаком **'::'** (двойное двоеточие). Например: **CGI::SessionM**

При подключении модуля и тогда, когда подключение не произошло, интерпретатор предоставляет стандартные имена пакетов:

main:: - текущий пакет

UNIVERSAL:: - Первичный класс «предков». Имеет ряд методов:

INVOCANT->isa(CLASS)
INVOCANT->can(METHOD)
INVOCANT->VERSION(NEED)

- **isa** – метод возвращает true, если классом инвоканта является CLASS или любой класс, наследующий CLASS

```
use FileHandle;
print "В FileHandle есть Exporter" if (FileHandle-
>isa("Exporter"));
```

- **can** возвращает ссылку на оригинальный первоначальный метод (подпрограмму), в противном случае, undef

- **VERSION** возвращает значение номера версии класса инвоканта. Если NEED больше чем версия возвращенного результата, то возникает исключительная ситуация - ошибка

SUPER:: - псевдокласс. Содержит имена замещенных (оригинальных) методов. Доступ к SUPER имеется только внутри Вашего класса, а не в вызывающем класс коде программы.

CORE:: - псевдопакет. Содержит имена замещенных (оригинальных) функций

Предварительная загрузка модуля на этапе компиляции.

use MODULE LIST
use MODULE

LIST – список импортируемых имен. Если список опущен, то выполняется импорт имен, который контролируется реализацией самого модуля посредством **use vars** или **use Exporter**. См. ниже.

use использует каталоги для поиска исходя из массива **@INC**. Добавить свой каталог можно с помощью прагмы **use lib PATH** или использовать массив или хэш из блока **BEGIN**. Например: *use lib './' или BEGIN {$INC{'MODULE.pm'}='./'}*

use MODULE VERSION
VERSION говорит о том, какой версии требуется модуль. Если VERSION указана больше, чем импортируемый модуль, то возникает ошибка компиляции.

use заставляет интерпретатор выполнить подпрограмму с именем **import**, когда как директива no - **unimport**

no заставляет интерпретатор выполнить подпрограмму с именем **unimport**

Непосредственная загрузка модуля на этапе исполнения.

require MODULE
require "MODULE.pm"

Первый вариант – аналог use, но на этапе исполнения и для поиска путей использует также массив @INC но на этапе выполнения.

Второй вариант понимает модуль как файл, поэтому вместо удвоенного двоеточия следует использовать «slash» (/) для поиска этого файла модуля.

do FILENAME – чтение и выполнение отдельного файла с любым расширением. Альтернатива require. Используется редко

Создание модулей

Создание модулей процедурного интерфейса:

package mymodule; # с этого места начинается код относящийся исключительно к пакету **mymodule**

```
require Exporter;
use vars qw($VERSION); # Имена, используемые с префиксом
mymodule ($mymodule::VERSION)
our @ISA = qw(Exporter); # наследуем модуль Exporter
our @EXPORT = qw(mysubs @myarrays $myscalar); # Имена по
умолчанию для использования без префикса имени модуля
our @EXPORT_OK = qw(mysubs @myarrays $myscalar); # Имена по
запросу с помощью конструкции use mymodule qw(mysubs
@myarrays $myscalar)
our @EXPORT_TAGS = (subs=> [qw (mysubs)], variables=>[
qw(@myarrays $myscalar)]); Доступ по требованию имен классов
subs и variables
our $VERSION= 1.00; # номер версии
…
1;
```

Создание модуля ООП интерфейса:

```
package mymodule;
use vars qw($VERSION);
$VERSION = 1.00;
…
sub new {
       my $class = shift; # получаем имя класса
       my $self = bless {}, $class; # создаем экземпляр класса
       return $self; # возвращаем ссылку на объект
}
…
1;
```

О объектной реализации см. раздел **«Классы и объекты»**.

Стандартные библиотеки.

Стандартный набор библиотек обычно поставляется с дистрибутивом Перл они разделяются на pragma библиотеки (работают как директивы компилятору) и стандартные библиотеки. Pragma библиототеки. Данные библиотеки

используют как: **use NAME**; когда хотят включить действие и **no NAME**; когда выключить.

В стандартный набор прагм входят:

diagnostics Включить режим расширенной диагностики.
integer Использовать целочисленную арифметику.
less Режим минимальной загрузки компилятора.
overload Режим переопределения операторов.
sigtrap Режим слежения за прерываниями.
strict Режим ограниченного использования "опасных" операторов.
subs Режим обязательного декларирования подпрограмм.

В стандартный набор библиотек входят:

AnyDBM_File Возможность работы с разными типами баз данных.
AutoLoader Загрузка в память функций только во время вызова.
AutoSplit Разделить модуль для автозагрузки.
Benchmark Анализ скорости исполнения программы.
Carp Предупреждения об ошибках.
Config Доступ к конфигурации Perl.
Cwd Получить имя текущей рабочей директории.
DB_File Работа с базой данных формата Berkley DB.
Devel::SelfStubber Режим отладки автозагрузки.
DynaLoader Динамическая загрузка библиотек C.
English Использовать длинные имена встроенных переменных.
Env Импортировать имена переменных окружения
Exporter Обеспечивает экспорт/импорт для модулей.
ExtUtils::LibList Определяет используемые библиотеки.
ExtUtils::MakeMaker Создает файл проекта Makefile
ExtUtils::Manifest Программы для создания и проверки файла MANIFEST
ExtUtils::Mkbootstrap Применеие файла начальной загрузки для DynaLoader.
Fcntl Определения как и в C Fcntl.h

File::Basename Синтаксический разбор спецификации файла.
File::CheckTree Быстрый проход по директориям диска.
File::Find Быстрый поиск файлов по директориям.
FileHandle Обеспечивает объектный метод доступа к указателям файлов.
File::Path Создание/удаление директорий.
Getopt::Long Расширенная обработка опций.
Getopt::Std Стандартная обработка опций.
I18N::Collate Сравнение символов локальной кодировки.
IPC::Open2 Межпроцессорный обмен по чтению и записи.
IPC::Open3 Межпроцессорный обмен по чтению, записи, и обрабоки ошибок.
Net::Ping Тест доступа к хосту.
POSIX Стандартный интерфейс по IEEE Std 1003.1
SelfLoader Загрузка функций только по вызову.
Socket Определение структур и констант как и в C socket.h
Test::Harness Стандартный тест с статистикой.
Text::Abbrev Создание таблицы сокращений по списку.
Подробное описание каждой библиотеки записано в самом файле.

Классы и объекты

Класс – это, иными словами, пакет, реализованный по механизму ООП. См. раздел **«Создание модулей»**. Имя класса всегда можно получить с помощью литеральной лексемы **__PACKAGE__**

Объект – это экземпляр класса

Свойства – набор переменных данного объекта

Методы – процедуры обработки и работы со свойствами объекта

Конструкторы и деструкторы

Конструктор – это процедура уровня модуля, которую можно вызывать посредством графа (**->**) или непосредственно. Например:

```
use CGI;
my $q = new CGI;
```

здесь **new** – это имя процедуры конструктора. Чаще всего, это имя **new**, но допускаются и другие имена конструкторов

Пример стандартного конструктора:

```
sub new {
       my $class = shift; # получаем имя класса
       my $self = bless {
              attr1=>$a,
              attr2=>$b,
              …
                     }, $class; # создаем экземпляр класса
       return $self; # возвращаем ссылку на объект
}
```

Деструктор запускается сразу после завершения выполнения программы. Его используют, как правило, для закрытия

соединений. Если не указывать в своем классе этот блок, то будет выполнятся метод AUTOLOAD.

```
DESTROY {
  my $self = shift; # получение имени класса
  # выполнение запуска деструктора по принуждению.
  $self->SUPER::DESTROY if $self->can("SUPERR::DESTROY");
  …
}
```

Методы

new – метод создания экземпляра класса. Конструктор. См. раздел **«Конструкторы и деструкторы»**

Любой метод отличается от обычной подпрограммы только тем, что характеризуется первым принимаемым аргументом, как ссылкой на экземпляр класса. Для получения этого класса можно пользоваться следующим общим принципом:

```
sub mymethod {
  my $self = shift; # получить ссылку на объект
  …
}
```

Доступ к правильно сформированному объекту можно получать через **$self**. А доступ к его методам – посредством диграфа. Например:

```
$a = $self->{A}; # получить значение свойства A объекта $self
```

Стандартные методы

BEGIN – блок выполняется сразу после синтаксического разбора (ASAP – as soon as parsed) и перед выполнением оставшейся части.

CHECK – запускается после **BEGIN** но перед запуском программы. Создан для возможности вторичного контроля результатов блока **BEGIN** и других.

INIT – Запускается в начале выполнения кода. После **CHECK**.

END – блок выполняется перед выходом. После выполнения основной части кода.

- Если блоков **BEGIN** или **INIT** несколько, то они выполняются в порядке объявления (FIFO)
- Если же блоков **CHECK** или **END** несколько – то в порядке, обратном порядку объявления (LIFO)

Например:

```
print "начало выполнения main";
die "завершение main";
die "это уже не будет достигнуто!"
END {print "1-ый END"}
CHECK { print "1-ый CHECK"}
INIT { print "1-ый INIT"}
END {print "2-ой END"}
BEGIN {print "1-ый BEGIN"}
INIT { print "2-ой INIT"}
BEGIN {print "2-ой BEGIN"}
CHECK { print "2-ой CHECK"}
END {print "3-ий END"}
```

Этот пример даст результат:

1-ый BEGIN
2-ой BEGIN
2-ой CHECK
1-ый CHECK
1-ый INIT
2-ой INIT
начало выполнения main
завершение main
3-ий END
2-ой END
1-ый END

AUTOLOAD – блок выполняется при обращении к несуществующей подпрограмме пакета. Название ее

сохраняется в **our** переменной **$AUTOLOAD**, например: **main::func**.

DESTROY – блок выполняется при завершении работы объекта для чистки «мусора». Блок доступен только на уровне класса. См. раздел **«Конструкторы и деструкторы»**

Свойства

Свойства объекта задаются при его создании с помощью **bless**

$a=bless REF, CLASSNAME

REF – ссылка на любую структуру данных, содержащей в себе свойства объекта. Включая мнимые структуры. Чаще – это хэши { }.

CLASSNAME – имя класса. Если опустить это имя, то будет использовано текущее имя. Имя класса возвращается в первом аргументе при вызове любого метода с помощью оператора графа (**->**).

Пример см. в разделе **«Конструкторы и деструкторы»**.

Создание дистрибутивов модулей (CPAN)

http://cpan.org
http://pause.perl.org

Размещение модуля на CPAN

1. Выбираем подходящее имя для модуля. Например: CGI::SessionM.
2. Проверяем, нет ли случайно одноименного модуля на CPAN: *http://search.cpan.org/*
3. Создаем файл модуля SessionM.pm
   3.1. Пишем название модуля в директиве: package CGI::SessionM
   3.2. Устанавливаем версию в переменную $VERSION? Например “1.00”

3.3. Пишем документацию POD согласно разделу «POD»
3.4. Формируем модуль либо по процедурному механизму доступа, либо по механизму ООП.
3.5. Отлаживаем модуль с помощью прагм warnings и strict

4. Создаем папку в любом месте (например в корне диска C:\) с названием: CGI-SessionM-1.00 и переходим в нее с помощью *cd c:\CGI-SessionM-1.00*
5. Запускаем команду: *h2xs -AX -n CGI::SessionM -v 1.00* (для помощи *h2xs -h*). Создается автоматически структура каталогов: CGI\SessionM* с заглушками: Changes (файл с историей изменения версий модуля); MANIFEST (список файлов в дистрибутиве); README (заглушка для README-файла); Makefile.PL (скрипт для генерации Makefile, в который вам следует вписать собственное имя и координаты); test.pl (тестовый скрипт); SessionM.pm (Файл модуля)
6. Переносим все эти файлы в нашу папку C:\CGI-SessionM-1.00* и удаляем ветку подкаталогов CGI\SessionM Проверяем структуру с помощью команд:

```
perl Makefile.PL
make
make test
make clean
```

7. Вставляем код созданного модуля SessionM в файл заглушки C:\CGI-SessionM-1.00\SessionM.pm
8. Проверяем модуль с помощью команд

```
perl Makefile.PL
make
make test
make install
```

9. Добавляем команды тестов в файл test.pl
10. Проверяем POD документацию с помощью команды: *pod2html SessionM.pm > SessionM.htm*
11. Редактируем файл README (можно воспользоваться *pod2text SessionM.pm > README*)

12. Проверяем наличие tar и gz (http://unxutils.sourceforge.net). Переходим в родительский каталог нашего каталога (C:\ - cd \) и выполняем команду упаковки:

```
tar cvf CGI-SessionM-1.00.tar CGI-SessionM-1.00
gzip --best CGI-SessionM-1.00.tar
```

13. После этого появится конечный архив дистрибутива CGI-SessionM-1.00.tar.gz, который можно публиковать на CPAN с сайта *http://pause.perl.org*

Установка модуля с CPAN

Для установки следует запустить одну из следующих команд:

perl -MCPAN -e shell
CPAN
ppm

Откроется интерактивная среда CPAN. Если Вы открыли эту среду впервые, то система попросит вас настроить ее. А после этого все настройки вступят в силу, и вы увидите приглашение в виде мигающего курсора.

С помощью команды install – можно устанавливать модуль, а с помощью команды search – получать статус модуля на CPAN. Более подробно см. встроенную справочную систему по команде help

Связывание

Можно привязывать методы к встроенным структурам Perl для облегчения работы с последними. Для связывания используется команда **tie** и **tied**. Для разрыва связи – **untie**

tie VARIABLE, CLASSNAME, LIST
tied VARIABLE
untie VARIABLE

tie возвращает ссылку на экземпляр класса, с которым связана VARIABLE. **tied** же позволяет просто «узнать» эту ссылку по имени переменной VARIABLE. **untie** – разрывает эту связь

более подробно см. **Tie::Hash**, **Tie::Array**, **Tie::Scalar**, и **Tie::Handle** модули.

Пример связывания скаляра:

```
#!/usr/bin/perl -w
package main;

my $var;
my $t = tie $var,"sample";
$var = "Hello ;)";
$var =~ s/\s+\;\)/!/g;

exit;

package sample;
sub import {
    print "- import (включение в работу механизма tie) -\n";
}
sub TIESCALAR {
    my $class = shift;
    my $instance = shift || undef;
    print "- tied (создание вторичного tie-класса) -\n";
    return bless \$instance => $class;
}

sub FETCH {
    print "- fetch (чтение значение переменной) -\n";
    return ${$_[0]};
}
```

```
sub STORE {
    ${$_[0]} = $_[1];
    print "- store (присвоение значения переменной) -
$_[1]\n";
}

sub DESTROY {
    undef ${$_[0]};
    print "- destroy (чистим за собой мусор в памяти) -\n";
}
1;
__END__
```

В итоге мы получим:

- tied (создание вторичного tie-класса) -
- store (присвоение значения переменной) - Hello ;)
- fetch (чтение значение переменной) -
- store (присвоение значения переменной) - Hello!
- destroy (чистим за собой мусор в памяти) -

Связывание скаляров

TIESCALAR classname, LIST
FETCH this,
STORE this, value
DESTROY this
UNTIE this

Связывание массивов

TIEARRAY classname, LIST
FETCH this, key
STORE this, key, value
FETCHSIZE this
STORESIZE this, count
CLEAR this
PUSH this, LIST
POP this
SHIFT this
UNSHIFT this, LIST
SPLICE this, offset, length, LIST

EXTEND this, count
DESTROY this
UNTIE this

Связывание хэшей

TIEHASH classname, LIST
FETCH this, key
STORE this, key, value
DELETE this, key
CLEAR this
EXISTS this, key
FIRSTKEY this
NEXTKEY this, lastkey
SCALAR this
DESTROY this
UNTIE this

Связывание заголовков

TIEHANDLE classname, LIST
READ this, scalar, length, offset
READLINE this
GETC this
WRITE this, scalar, length, offset
PRINT this, LIST
PRINTF this, format, LIST
BINMODE this
EOF this
FILENO this
SEEK this, position, whence
TELL this
OPEN this, mode, LIST
CLOSE this
DESTROY this
UNTIE this

Регулярные выражения

qr/PATTERN/imsox
Позволяет предварительно скомпилировать выражение

m//

Поиск с возвратом значения истина или ложь:

$status = $a =~ **m/PATTERN/MODIFY**;
$status = $a =~ **/PATTERN/MODIFY**;
$status = $a =~ **‘PATTERN’MODIFY**;
$status = $a =~ **?PATTERN?MODIFY**;
$status = $a =~ **m#PATTERN#MODIFY**; # с **m** допустимы любые символы кавычек. Тоже относится и к операторам **s** и **tr**

Кавычки **‘ ’** в третьем случае подавляют интерполяцию.
Кавычки **? ?** в четвертом случае инициализируют сброс позиции поиска в начальную позицию. И следующий за ним **m**// будет работать со строкой так, как будь то начал работать с новой строкой.
Если опустить *«$status = $a =~»* то поиск будет вестись для **$_**

s///

Подстановка найденного подтекста по шаблону:

$lvalue = “bla bla bla”;
$status = $lvalue =~ **s/PATTERN/REPLAEMENT/MODIFY**;

$status содержит в себе количество успешных замен

```
s/(^\s+)|(\s+$)//; # отбросить ведущие и последние пробелы
s/\s+/  /g; # сжать внутренние пробелы
```

Для работой с массивами можно воспользоваться термом:

for (@a) {s/foo/bar/g}
или:

s/foo/bar/g for @a

```
1 while s/(\d)(\d\d\d)(?!\d)/$1'$2/; # расстановка
разрядностей
1 while s/\([^()]*\)//g; # удаление (вложений (как (это)))
1 while s/\b(\w+) \1\b/$1/gi; # Удаление повторений слов
```

tr/// или y///

Замена найденного подтекста – транслитерация. Шаблоны здесь не используются!

$lvalue = “bla bla bla”;
$status = $lvalue =~**tr/SEARCHLIST/REPLACELIST/MODIFY**;

```
tr/A-Za-z/N-ZA-Mn-za-m/; # rot13 шифрование
tr/aeiou/!/; # замена гласных на «!»
tr{/\\\r\n\b\f. }{_}; # замена необычных символов на «_»
tr /A-Z/a-z/ for @ARGV; # перевод в нижний регистр
$count=($a=~tr/\n//); # количество строк в $a
$count=tr/0-9//; # количество цифр в $_
tr/a-zA-Z//s; # bookkeeper -> bokeper
tr/@$%*//d; # удалить перечисленные
tr#a-zA-Z0-9+/##cd; # удалить не эти
tr/a-zA-Z/_/cs; # Заменить не эти символы на «_»
tr[\200-377][\000-\177]; # сброс 8-го разряда
tr/AAA/XYZ/; # поменять A на X
```

для перевода регистра лучше использовать **lc()**, **uc()** и их аналоги

Специальные переменные REPLACEMENT

$_ - Переменная по умолчанию для регулярных выражений

$& - текущий участок текста по шаблону замены.

```
s/abc|def|ghi/\u$&/g; # замена всех совпадений перечисления
верхним регистром
```

$`** - Весь текст до **$&

$'** – Весь текст после **$&

$* - признак многострочности. **0** – однострочная; **1** – многострочная

$1, $2 ... – Переменные, определяющие захваченный блок. См. Раздел **«Кластеризация»**

$+ - Последнее совпадение с помощью **(?(cond)yes|no)** или **(?(cond)yes)**

$^R – Ожидает результат конструкции **(?{...})**

$^N - Ожидает самый последний закрытый захват

@- - Смещения начала группы. **$-[0]** указывает на начало всего совпадения

@+ - Смещения конца группы. **$-[0]** указывает на конец всего совпадения

Модификаторы

i – игнорирование различие регистра для латинских символов
s – разрэшение сопоставлять «**.**» (точку) с **\n** и игнорировать **$***
m – разрэшение сопоставлять «^» и «**$**» только до символа **\n**. Т.е. сопоставлять шаблон "**^.+$**" только для одной подстроки, а не для всей строки от начала до конца «пожирая» все **\n**
x – игнорировать почти все пробельные символы и разрэшать комментарии
o – компилировать шаблон один раз

Модификаторы для m//cgimosx

cg – Отмена действия **??** в конструкции типа **m??**.
g – глобальный поиск всех соответствий, а не одно. В списковом контексте возвращаются все найденные комбинации. В скалярном – статус выполнения.

Модификаторы для s///egimosx

e – REPLAEMENT вычислять как выражение EXPRESSION

Модификаторы для tr///cds

c – дополнение REPLACELIST символами из SEARCHLIST.
d – удаление найденных, но не замененных символов
s – подавлять повторяющиеся замененные символы. Сжатие

Метасимволы

\ -Следующий символ - спецсимвол
. – Любой символ (если использован **/s**, то даже **\n**)

Спецсимволы

\. – точка «**.**»
\0 – нулевой символ
\a – сигнал
\e – escape-символ
\f – новая страница
\n – новая строка
\r – возврат коретки
\t – tab (09)
\v – вертикальный пробел
\037 – восьмеричный символ ASCII
\x7f - шестнадцатеричный символ ASCII
\x{263a} – тоже самое но в диапазоне Unicode
\cX - Control-X
\N{name} – именованный символ
\C – байт
\pP – Совпадение с P-символом (Unicode)
\p{...} - Совпадение с именованным P-символом (Unicode)
\PP – Не P-символ
\P{...} Не P-символ по имени
\X - Совпадение с любым символом (Unicode)

Управляющие

\l – Перевод поледующего символа в нижний регистр
\u – Перевод поледующего символа в верхний регистр
\L – Перевод в нижний регистр текста до **\E**
\U – Перевод в верхний регистр текста до **\E**
\Q – Экранировать неалфавитные символы «\» до **\E**
\E – завершает действие управляющих символов

Маркеры. Символы «нулевой» длины

^ - Начало строки (или подстроки если использован **/m**)
$ - Конец строки (или подстроки если использован **/m**)

\b - Символ между границей слова (между **\w** и **\W**)
\B - Символ НЕ между границей слова (не между **\w** и **\w** или **\W** и **\W**)
\A – Символ начала всей строки (независимо от **/m**)
\z - Символ конца всей строки (независимо от **/m**)
\Z - Символ конца строки или подстроки (зависит от **/m**)
\G – Символ конца совпадения **m//g**

Эквиваленты

\s – Пробелы **[\t\n\r\f]**
\S – Не пробелы **[^ \t\n\r\f]**
\w – Символы слова **[a-zA-Z_0-9]**
\W – Символы не слова **[^a-zA-Z_0-9]**
\d – Символ цифр **[0-9]**
\D – Символы не цифры **[^0-9]**

Квантификаторы

*** - 0** или более совпадений подшаблона
+ - 1 или более совпадений подшаблона
? - 0 или **1** совпадений подшаблона **{0,1}**
{count} – равно count совпадений подшаблона
{min,} – не меньше чем min совпадений подшаблона
{min, max} – не меньше min и не больше max совпадений

{n,m}? {n,}? {n}? *? +? – Знак вопроса означает, что используется не все подшаблоны для разового замещения, а только один – первый по порядку. Например:

```
$a=”abc-abc-abcL”;
$a=~s/.*c/d/; # dL
$a=~s/.*?c/d/; # d-abc-abcL
```

Пример демонстрирует т.н. «жадность» квантификаторов и преодоление этой «жадности»

Кластеризация

[...] – Выборка одного из перечисленных символов
[amy] Выборка 'a', 'm' или 'y'

[^...] – Выборка одного НЕ из перечисленных символов

(...) – Группировка подшаблонов с помещением результата в $1, **$2...** по принципу инкапсуляции:

/^((\w+)(\w+))$/
$1 – “^((\w+)(\w+))$”
$2 – “^((\w+)”
$3 – “(\w+))$”

(?:...) – Группировка без захвата в **$1**, **$2** и т.д.

...|... - Дизъюнкция последовательностей

[…\-…-] – Экранированное тире, тире в начале или в конце конструкции означает “тире”

Расширения кластеризации

(?#text) – Не учитывать комментарии
(?:text) – Не учитывать этот кластер вообще
(?imxs-imsx) – Вкл./откл. модификаторов для **m//**
(?imxs-imsx:...) – тоже самое, плюс кластер
(?=...) – True, если выполняется условие ПОСЛЕ кластера

(?!...) - False, если выполняется условие …
(?<=...) - True, если выполняется условие ПЕРЕД кластером
(?<!...) - False, если выполняется условие ПЕРЕД кластером
(?>...) – Совпадение с кластером. Заглядывание без захвата
(?{code}) – Выполнение code с возвратом в **$^R**
(??{code}) – Выполнение code как regexp
(?(cond)yes|no) – аналог **cond?yes:no**
(?(cond)yes) тоже самое но без **no**

Обратные ссылки. Заглядывание

\1, **\2** ... – Эти символы понимаются как **$1**, **$2** … но не в блоке REPLAEMENT и тексте, а в блоке PATTERN. Это позволяет сравнивать будущие «подшаблоны» в шаблоне.

/(\w)\w+(\w)w*\2\w\1/; # палиндром. SAIPPUAKIVIKAUPPIAS SAIPPUAKIVIKAUPPIAS (торговец щелоком).

Функции

pos SCALAR – Возвращает позицию найденной подстроки в скаляре

quotemeta EXPR – Экранирование всех символ не **\d** и **\w**.

reset EXPR - Переустановка *?pattern?* статуса, сброс **pos**

study SCALAR - Анализ скаляра для оптимизации поиска

Документация

Ремарки “#”

Все строки от символом «#» и далее до конца строки называются ремарками. Например:

```
# ремарка
```

или

```
print $a; # ремарка
```

«here» документы

print <<[”]identifier[”];

Интерполируемый текст. Здесь можно работать с большими блоками текстовой информации. В таком блоке работает обычная интерполяция, что позволяет использовать метасимволы и переменные. Двойные кавычки у первого идентификатора можно опускать. Важно заметить, что второй идентификатор следует начинать с крайнего левого края текста без пробелов и разделителей!

identifier

print <<’identifier’;

Блок текста с ограниченной интерполяцией

identifier

$a = <<identifier;

Содержание переменной присваивания

identifier

($a = <<identifier) =~ s/^\s+//gm;

Содержание переменной с последующим регулярным выражением

identifier

```
print <<'odd'
2345
odd
+1000; # 12345
```

В качестве кавычек могут быть использованы кавычки ` ` и другие, соответствующие значению содержимого.

Форматы

printf FILEHANDLE FORMAT, LIST
sprintf FORMAT, LIST
format FORMAT

FORMAT = "<%><ФЛАГ><ЦИФРА><СПЕЦИФИКАТОР>"

print – выдает на канал вывода список указанного формата
sprint – возвращает скаляр, преобразуя список по формату
format – предопределение формата

format FORMATNAME =
FORMLIST
.

Если FORMATNAME' отсутствует, то значение по умолчанию - STDOUT.
FORMLIST - это строки формата. Они бывают трех типов:

1. Комментарий. Строка начинается символом '#'.
2. Описатель полей данных (picture).
3. Строка аргументов используемых описателем.

Описатель — это строка, которая выводится в виде «как есть», за исключением специально обозначенных форматов полей данных. Каждое поле начинается либо символом '**@**', либо '^'. В описательной строке указывается только положение и вид выводимых данных, но не имена полей и переменных. Для этого предназначена следующая строка аргументов, которая следует всегда после описателя и содержит имена переменных или

целые выражения в порядке, указанном описателем. Размер и вид поля в описателе обозначается следующими символами:

">>>>" - выравнивать значение по правому краю.
"<<<<" - ~ по левому.
"||||" - ~ по центру.
"**####.###**" - формат числа с точкой.
"***@****" - многострочная строка. Данные выводятся в колонку.

Размер поля равен количеству указанных символов. Символ '^' в начале поля имеет специальное значение.
Так:
"^*####*"- пусто, если переменная не определена.
для строчного скаляра: "^<<<<<" - Выводится сколько возможно символов, а значение переменной меняется на остаток вывода, которого можно продолжить на следующих строках, причем они могут иметь свои поля. Пример:

```
printf "%9d", -123; # _ _ _ _ _ - 1 2 3
printf "%9.3d", -123.4567; # _ _ - 1 2 3 . 4 5 6
printf "%*.*f",7,2,98.736; # -98.74
```

7 – поле **7** = по левому краю
2 – точность

```
sprinf "version is v%vd\n", $^V; # Версия Perl
sprinf "addres is %vd\n", $addr; # Адрес IPv4
sprinf "addres is %*vX\n",":", $addr; # Адрес IPv6
sprinf "bits are %*vbd\n"," ", $bits; # Битовые строки
```

Спецификаторы форматов

%% - знак процента %
%c – символ заданного номер ASCII (=CHR)
%s – преобразование списков в строку
%d, %D, %i – целое со знаком. Десятичные отбрасываются
%u, %U – целое без знака
%o, %O – восьмеричная форма числа
%x, %X – шестнадцатеричная форма числа
%e, %E – плавающая точка, экспоненциальная форма.
%f, %F – число с плавающей точкой

%g, %G - число с плавающей точкой в двух форматах
%b – двоичная запись
%p – адрес ширины в шестнадцатеричном формате
%n – записывает предыдущие числа в переменную

Флаги форматов

пробел – пробел перед положительным числом
+ - плюс перед положительным числом
- — выравнивание по левому краю
0 – использовать нули, а не пробелы, для заполнения левого поля при выравнивании по правому полю
- использовать префиксы **0** и **0x** для восьмеричных и шестнадцатеричных чисел
число – минимальная ширина поля
.число – точность. Количество цифр после точки
l – C-формат целых чисел (long/unsigned long)
h – C-формат целых чисел (int/unsigned)
V – Perl формат целых чисел
v – вектор целых чисел. Для вывода порядковых номеров символов в произвольных строках
***** - любой числовой символ точности или количество знаков

POD

POD (Plain Old Documentation) – документация, сопровождающая код

POD-строками считаются все строки, начинающиеся со знака "=" в начале строки (без пробельных отступов) и по строку "**=cut**" в начале строки (без пробельных отступов). Такие блоки пропускаются интерпретатором, что позволяет внедрять эту документацию в любое место кода в любом количестве.

Например:

```
=head1 Description

…

=cut
```

Директивы

В примере конструкция "**=head1 Description**" называется директивами POD

=head1
=head2
=head3
…

Директивы заголовков уровней 1,2,3 и т.д. (HTML аналог <H1> и др.)

=cut

Ограничитель участка POD документа.

=pod

Синоним **=head**, но для целей комментирования участков кода.

=over NUMBER
=item SYMBOL
=back

over начинает участок списка, back его завершает. item определяет элемент списка. NUMBER определяет количество пробелов до пункта от левого края. Чаще этот параметр имеет значение 8. SYMBOL определяет символ списка. Например:

=over 4
=item *

String1 bla-bla-bla

=item *

String2 bla-bla-bla

=back

SYMBOL бывает: *****, 0-9 и т.д., a-z и т.д., имена и др.

=for TRANSLATOR
=begin TRANSLATOR
=end TRANSLATOR

Эти директивы избыточны, поэтому здесь не описываются.

Последовательности

I<text> - курсив
B<text> - полужирный шрифт
C<text> - код. Аналог HTML <code>
S<text> - текст с неразрывными пробелами
L<text> - ссылка
L<text/ident> - ссылка на имя заголовка
L<text/”sec”> - ссылка на раздел в другой страницы
L<”sec”> - ссылка на имя заголовка
F<text> - курсив для имен файлов

X<text> - указатель
E<escape> - литералы (Escape-символы):
lt - <
gt - >
sol, verbar – для **L<>**
NNN – символ ISO-8859-1
entity – случайность (не используется)
Z< > - символ нулевой ширины. Например *Z<>=cut*

Пример: *C<$a E<lt>=E<gt> $b>*
Вывод: $a <=> $b

Трансляторы POD

pod2text – преобразует POD в Text
```
pod2text filename.pm > filename.txt
```

pod2html – преобразует POD в HTML
```
pod2html filename.pm > filename.html
```

Стандарт заголовков head1

=head1 NAME

Имя вашего модуля

=head1 SYNOPSIS

Краткий синтаксис вызова процедур модуля. Интерфейс

=head1 DESCRIPTION

Детальное описание интерфейса и модуля

=head1 DIAGNOSTICS

Выполненные диагностики и проверки

=head1 HISTORY

История изменений в порядке увеличения версии

=head1 DISCLAIMER

Правовая информация. Ответственность сторон

=head1 THANKS

Благодарности

=head1 AUTHOR

Автор. Координаты

=head1 COPYRIGHT

Авторские права

Структуры данных

```
%a=(
       jeden=>1,
       dwa=>2,
       3=>['trzy', 'czterej'],
       'piec'=\%b
);
```

Скаляры

$a = “hello”; # строка
$_ = “hello”; # строка присваивается скаляру по умолчанию
$a = 40.5; # число
$a = system(“vi …”); # статус выполнения команды
$a = $b; # значение другой переменной
$a = \$b; # ссылка на скалярную переменную $b
$a = 6.02e11; # экспоненциальное значение
$a = [1,2,3,4]; # ссылка на анонимный массив
$a = {Na => 19, Cl =>35}; # ссылка на анонимный хэш
$a = sub {print ‘hello’}; # ссылка на анонимную подпрограмму
$a = new Camel; # ссылка на объект
${a} = “hello”; # устанавливается переменная $a

Массивы и списки

@a = (1, 2, ‘trzy’, 3, -500.5, $b); # список значеинй
@a = qw(jeden dwa trzy); # список значеинй
@a = (1..5); # список значеинй от 1 до 5
@a = 1..5; # список значеинй от 1 до 5
@a = (‘a’..’z’); # список значеинй от a до z
@a=(‘a’..’z’)[1,2,3]; # b,c,d
@a = @b; # другой массив
@a = @b[1,2,3]; # срез другого массива
@a = ( ); # пустое значение. «Обнуление» массива
$#a = 5; # значение $#a содержит индекс последнего элемента
$a[0] = 'dwa'; # значение массива с первым индексом 0
$a[-1] = 'dwa'; # значение массива с последним индексом
($a,$b) = (1,2); # значения скаляров списка
($a,$b) = ($b,$a); # обмен значениями скаляров списка

($a,$b) = @a; # массив

Работа с массивами

[$status =] push(@a,$a); # вставка элемента в конец массива и возврат истины в случае успеха

$a=pop(@a); # возвращение последнего элемента и его удаление

unshift(@a,$a); # вставка элемента в начало массива со сдвигом

$a=shift(@a); # возвращение первого элемента и его удаление

$a=scalar(@a); # возвращение общего числа элементов массива

$a=$#a; # последний индекса массива. Модифицируемый

[$status =]splice(@a[,$offset [,$length [,@b]]])

- опущен **@b** – удаление указанного диапазон
- опущен **$length** – удаляется участок до конца
- опущен **$offset** – удаляется весь массив

reverse @a; # переворачивает массив наоборот
@b = reverse(@a); # переворачивает список и возвращает итог

sort @a; # сортировка массива по ASCII таблице
@b = sort(@a); # сортирует список и возвращает итог
sort BLOCK @a; сортировка массива используя блок
sort SUBNAME @a; сортировка массива используя имя подпрограммы.

- BLOCK и SUBNAME принимают автоматически переменные '**$a**' и '**$b**', а возвратить должны результат сравнения (-1, 0 или 1), например:
  "*sort {$a <=> $b} @a*" в этом примере сортировка будет осуществляться в возрастающем порядке числового значения.
  "*sort {$b <=> $a} @a*" – сортировка в обратном порядке.
  "*sort {$a cmp $b} @a*" – сортировка в прямом порядке, но выполняется, отталкиваясь от ASCII таблицы символов. Последний пример выполняется по умолчанию.

@a = grep $_ < 6, @b; # возврат значений меньше 6
@a = grep BLOCK @b; # в качестве выражения выступает блок **{ }** который должен вернуть истину для подтверждения возврата текущего элемента массива. Например: *@a = grep {$_ < 6} @b;*

@a = map $_ * 6, @b; # возврат увеличенных в 6 раз значений
@a = map BLOCK @b; # в качестве выражения выступает блок **{ }** по аналогии с **grep()**

delete $[$index]; # удаление элемента массива

$a = exists $a[0]; # true, если элемент с индексом 0 существует, даже если его значение неопределенно (**undef**)

$a = defined $a[0]; # true, если значение определено

Анонимные массивы

```
$a=[1,2,3,'sss',$a,\$a]; # создание анонимного массива
$b=$a->[1]; # чтение значения индекса 1 из анонимного массива
@b=@$a; # чтение всех значений анонимного массива
$a->[1] = 'ddd'; # запись значения индекса 1 анонимного
массива
```

Многомерные массивы

Многомерные массивы можно представить как массивы ссылок на другие массивов.

$a[0][0] эквивалентно: **$a[0]->[0]**

Срезы

@a=@b[2,5,8,12..15]; # срез массива

Хэши

%a = (jeden, 1, dwa, 2, 3, 'trzy'); # ключи, значения
%a = (jeden=>1, dwa=>2, 3=>'trzy'); # ключи => значения
$a{jeden} = 1; # значение ключа 'jeden'
$a{"jeden"} = 1; # значение ключа 'jeden'
%a=@a; # массив

Работа с хэшами

@a = keys %a; # список ключей

@a = values %a; # список значений

(@a, @b) = each %a; # пара – ключ/значение

%a = reverse(%b); # замена всех ключей значениями и наоборот
reverse %b; # тоже самое но без возврата

delete $a{'key'}; # удаление пары ключ/значение

$a = exists $a{'key'}; # true, если элемент 'key' существует, даже если его значение неопределенно (**undef**)

$a = defined $a{'key'}; # true, если значение определено

Анонимные хэши

$a={a=>1,b=>2}; # ссылка на анонимный хэш
$b = $a->{a}; # ссылка на значение ключа 'a' хэша

Многомерные хэши

Многомерные хэши можно представить как хэши ссылок на другие хэши.

$a{jeden}{dwa}** эквивалентно: **$a{jeden}->{dwa}

Срезы

@a=@b{'jeden', 'dwa'}; # срез хэша $b{jeden} и $b{dwa}

Файловая система. Процессы

Файлы

Открытие дескриптора файла.

open FILEHANDLE [, MODE] [, EXPRESSION] [, LIST];
open FILEHANDLE, MODE, REFERENCE;

open SESAME, ‘path/filename’; # чтение файла
open SESAME, “<path/filename”; # чтение файла явно
open SESAME, “>path/filename”; # создание и запись файла
open SESAME, “>>path/filename”; # дописывание файла
open SESAME, “| program”; # организовать выходной фильтр
open SESAME, “program |”; # организовать входной фильтр

Использование дескрипторов файла.

$a = <SESAME>; # чтение строки файла
@a = <SESAME>; # чтение всего файла в массив построчно
print SESAME $a; # запись строки в файл
print SESAME @a; # запись массива в файл

Закрытие дескриптора файла.

close FILEHANDLE;

close SESAME; # закрытие открытого дескриптора

Закрытие дескриптора происходит автоматически, если открыть его повторно!

Блокировка файла.

flock FILEHANDLE, OPERATION

Блокировка/разблокировка открытых файлов

OPERATION:

2 – заблокировать файл

8 – разблокировать файл (**close()** выполняет это же)

Каталоги

См. функции работы с каталогами

Каналы (конвейеры)

См. функции работы с каналами **pipe**

Потоки

См. функции работы с потоками **fork**

Процессы и сигналы

См. функции работы с сигналами

Встроенные функции

Встроенные функции используются как термы выражений и подразделяются на две категории: списковые операторы и унарные операторы. Это влияет на их приоритет по отношению к оператору ',' - запятая. Списковые операторы могут иметь множество (список) аргументов, а унарные только один. Таким образом, запятая завершает аргументы унарного оператора и разделяет аргументы спискового.

print [FILEHANDLE] LIST
Вывод на экран или в указатель файла список. FILEHANDLE по умолчанию используется как STDOUT. См. раздел «Заголовки»

```
print 123; # 123
print 'abc'; # abc
print "\$$\uadfg"; # $Adfg
print @a; # все элементы массива в одну строчку используя $,
print "@a"; # все элементы массива в одну строчку используя
$"
print scalar(@a); # количество элементов массива
print %a; # все элементы хэша в одну строчку используя $, и
$;
```

Обработка скаляров

chomp VARIABLE
chomp LIST
Удаляет последний символ **\n** в переменной и возвращает его. **\n** берется из переменной **$/**

chop VARIABLE
chop LIST
Удаляет последний символ строки переменной и возвращает его

chr NUMBER – возвращает символ кода NUMBER

crypt PLAINTEXT, SALT – кодирование коротких скаляров

hex HEXEXPR – возврат десятичное значение HEXEXPR

index STR, SUBSTR[, OFFSET] – возвращает позицию найденной подстроки. **-1** – строка не найдена

lc EXPR – перевод строки в нижний регистр

lcfirst EXPR – перевод первого символа строки в нижний регистр

length EXPR – возвращает длину строки

oct EXPR – перевод восьмеричного EXPR в десятичное число

ord EXPR – возвращает числовое значение первого символа EXPR

pack TEMPALTE, LIST – принимает список и сжимает его в одну строку исходя из TEMPLATE

```
pack ("Cf", 244, 3.14); # создает строку упакованных символов
pack "C/a","\04Gurusamy"; # Guru
pack "H8","5065726c"; # Perl
```

Символы:

a – строка байт, дополняемая нулями
A - строка байт, дополняемая пробелами
Z - строка байт, законченная нулем
b – битовая строка с возрастанием порядка разрядов в каждом байте (VEC)
B - битовая строка с убыванием порядка разрядов в каждом байте
h – шестнадцатеричная строка. Младший полубайт впереди
H – шестнадцатеричная строка. Старший полубайт впереди
c – число со знаком (8 бит)
C – число без знака (8 бит)
s – короткое целое со знаком (16 бит)
S – короткое целое без знака (16 бит)

i – целое со знаком
I – целое без знака
l – длинное целое со знаком (32 бита)
L – длинное целое без знака (32 бита)
n – 16-битное число в сетевом порядке
N – 32-битное число в сетевом порядке
v - 16-битное число в порядке VAX
V - 32-битное число в порядке VAX
q – 64-разрядное целое со знаком
Q - 64-разрядное целое без знаком
j – альтернатива i для Perl
J - альтернатива I для Perl
f – число с плавающей запятой одинарной точности
d - число с плавающей запятой двойной точности
F - альтернатива f для Perl
D - альтернатива d для Perl
p – указатель на строку завершающуюся нулем
P – указатель на строку фиксированной длины
u – строка в кодировке UUencode
U – номер сивола Unicode
w – BER сжатое целое
x – не символ. Проскочить на байт вперед
X - не символ. Проскочить на байт назад
( ) – группировка символов
@ - заполнить нулями до абсолютной позиции
/ - length-item/string-item – упаковка и распаковка

q//, **qq**// - см раздел **«Кавычки и интерполяция»**

reverse LIST – переворачивает значения LIST и возвращает конкатенированный результат

rindex STR, SUBSTR[, POSITION] – тоже, что и index, но возвращает значение последнего найденного соответствия

sprintf FORMAT, LIST – См. Раздел **«Форматы»**

substr EXPR, OFFSET[, LENGTH[, REPLACEMENT]]
Возвращает подстроку из строки с указанными смещением, длиной и, возможно, строкой замещения участка во входной строке

tr///, y/// - см. раздел **«Регулярные выражения»**

uc EXPR – перевод строки в верхний регистр

ucfirst EXPR – перевод первого символа строки в верхний регистр

Регулярные выражения

m//, s///, qr// - см. раздел **«Регулярные выражения»**

pos, quotemeta, study - см. раздел **«Регулярные выражения»**

Обработка Чисел

abs VALUE – возвращает абсолютное значение VALUE

atan2 VALUEX, VALUEY – арктангенс отношения X/Y в диапазоне от –п до п

cos EXPR – косинус значения в радианах

exp EXPR – возвращает e в степени EXPR

hex EXPR – возвращает десятичное значение указанного шестнадцатеричного в EXPR

int EXPR – возвращает целую часть (отбрасывая десятичные)

log EXPR – возвращает натуральный логарифм EXPR

oct EXPR – возвращает десятичное число из восьмеричного в EXPR

rand EXPR – псевдослучайное число с пл. точкой в диапазоне от **0** до EXPR или **0** до **1** если EXPR опущено

sin EXPR – возвращает синус значения в радианах

sqrt EXPR – возвращает квадратный корень от EXPR

srand EXPR – генерация стартовой точки для rand исходя из EXPR

Работа с массивами

pop, push, shift, splice, unshift – см. раздел **«Массивы и списки»**

Работа со списками

grep, map, reverse, sort – см. раздел **«Массивы и списки»**

split /PATTERN/[, EXPR[, LIMIT]] – разрезает строку на отдельные компоненты списка используя в качестве разделителя шаблон. Если шаблон опустить, то разделитель будет делить строку EXPR по символам. LIMIT ограничивает количество полей.

join EXPR, LIST - обратный **split**. Создает скаляр из элементов списка, используя EXPR в качестве разделителя.

qw/STRING/ - см раздел **«Кавычки и интерполяция»**

unpack TEMPLATE, EXPR – действие, обратное функции **pack()**. Возвращает список распакованных последовательностей.

Работа с хэшами

delete, each, exists, keys, values – см. Раздел **«Хэши»**

Ввод/Вывод

binmode FILEHANDLE[, DISCIPLINE] – Установка для данного FILEHANDLE дисциплины (правил) работы с байтами содержимого. По умолчанию DISCIPLINE работает «как есть». См. Прагму **open**

close FILEHANDLE – закрывает связанный с FILEHANDLE объект привязки – файл, канал и т.д. см. раздел **«Файловая система. Процессы»**

closedir DIRHANDLE– Закрытие каталога, с помощью **opendir**

dbmclose, dbmopen – Привязка и ее разрыв. Не описывается глубже

die LIST – Выход с предварительной записью в STDERR сообщения LIST и результатом из **$!**.Из **eval{}** возвращается не **$!**, а **$@**.

eof FILEHANDLE – возвращает истину на последнем прочтенном символе файла. Определяет достижение конца файла.

fileno FILEHANDLE – возврат дескриптора файла в основе FILEHANDLE

flock FILEHANDLE, OPERATION - см. раздел **«Файловая система. Процессы»**

format – см. раздел **«Форматы»**

getc FILEHANDLE – возвращает очередной байт из входного файла

print FILEHANDLE LIST
print LIST

Выводит LIST в стандартный STDOUT или другой любой FILEHANDLE

printf – см. раздел **«Форматы»**

read FILEHANDLE, SCALAR, LENGTH[, OFFSET] – читает LENGTH байт со смещения 0 или OFFSET из FILEHANDLE в SCALAR и возвращает результат выполнения чтения.

readdir DIRHANDLE – Читает список файлов в открытом каталоге DIRHANDLE

rewinddir DIRHANDLE – устанавливает текущую позицию для readdir DIRHANDLE.

seek FILEHANDLE, OFFSET, WHENCE – Установка указателя OFFSET для FILEHANDLE отталкиваясь от атрибута WHENCE (**0** – от начала, **1** –от текущей позиции или **2** – от конца файла)

seekdir DIRHANDLE, POS – установка позиции для **readdir** DIRHANDLE.

select FILEHANDLE – возвращает работающий указатель выходного файла и переопределяет его как FILEHANDLE

syscall LIST – вызывает системный вызов **syscall**, но не команду ОС. Глубже не рассматривается

sysread – системная альтернатива **read()**

sysseek – системная альтернатива **seek()**

syswrite FILEHANDLE, SCALAR[, LENGGTH[, OFFSET]] Запись в FILEHANDLE значения из SCALAR длиной LENGGTH в позицию со смещением OFFSET или установленную ранее командой **sysseek()**

tell FILEHANDLE – возвращает текущую позицию в FILEHANDLE

telldir DIRHANDLE – возвращает текущую позицию в директории DIRHANDLE

truncate FILEHANDLE, LENGTH
truncate EXPR, LENGTH
Усекает файл FILEHANDLE или EXPR до указанной длины LENGTH.

warn – аналог **die()** но не завершает работу программы

write FILEHANDLE – вывод отформатированной записи в файл. Не использовать для двоичных операций!

Работа с данными фиксированной длины

pack – см. **«Обработка скаляров»**

read, syscall, sysread, syswrite - см. **«Ввод/Вывод»**

unpack – см. **«Работа со списками»**

vec EXPR, OFFSET, BITS – хранение беззнаковых целых чисел EXPR. Используется редко. Описание опущено

Файлы/Каталоги

-X – см. раздел **«Операторы работы с файлами»**

chdir EXPR – открыть директорию

chmod MODE, LIST – Изменяет права списка файлов LIST на значение MODE

chown UID, GID, LIST – Изменяет владельца и группу для списка файлов LIST

chroot FILENAME – FILENAME становится корневым каталогом для текущего процесса

fcntl FILEHANDLE, FUNCTION, SCALAR – Вызов системных функций на уровне ОС. Глубже не рассматривается

glob EXPR – аналог **<*> -** возвращает список файлов указанного каталога в EXPR

ioctl FILEHANDLE, FUNCTION, SCALAR – Вызов системных функций на уровне ОС. Глубже не рассматривается

link OLDFILE, NEWFILE – Функция создает новое имя файла являющееся ссылкой для старого

lstat EXPR – аналог системного stat, но для имени ссылки на файл

mkdir FILENAME, MASK – Создание каталога FILENAME с правами в MASK.

open – см. раздел **«Файловая система. Процессы»**

opendir DIRHANDLE, EXPR – открытие каталога с именем EXPR.

readlink EXPR – возвращает имя файла адресата символической ссылки EXPR

rename OLDNAME, NEWNAME – переименование фала

rmdir FILENAME - Удаление каталога FILENAME

stat FILEHANDLE – возвращает список атрибутов файла в виде 13-элементов.
(**dev, ino, mode, nlink, uid, gid, rdev, size, atime, mtime, ctime, blksize, block**)

dev - имя устройства
ino - номер i-узла
mode - права доступа
nlink - количество связей
uid - идентификатор владельца
gid - идентификатор группы
rdev - тип устройства
size - размер файла в байтах
atime - дата последнего обращения
mtime - дата последней модификации
ctime - дата последнего изменения статуса
blksize - размер блока на диске
block - количество блоков в файле.

symlink OLDNAME, NEWNAME – создание символической ссылки

sysopen FILEHANDLE, FILENAME, MODE[, MASK] – аналог **open()** но с расширенными возможностями, указанными в MODE и возможностью установки маски MASK

MODEs:

O_RDONLY – Только чтение
O_RRONLY – Только запись
O_RDWR – Чтение и запись
O_CREAT – Создать файл, если он не существует
O_EXCL – Ошибка, если файл уже существует

O_APPEND – Дописывание
O_TRUNC – Усечение
O_NONBLOCK – не блокирующий доступ

O_NDELAY – синоним **O_NONBLOCK**
O_SYNC – Записывает блок пока данные не запишутся физически в устройство
O_DIRECTORY – Отказ если файл не каталог
O_NOFOLLOW – Отказ если последняя часть пути – ссылка
O_BINARY – Указатель BINBODE для Microsoft ОС
O_LARGEFILE – Требуется больше 2 Гб пространства

O_NOCTTY – Открытие файла терминала не делает его файлом-терминалом процесса

И др.

Например:

sysopen FH, $path, O_WRONLY|O_EXCL|O_CREAT – Открыть и создать файл для записи, который не должен предварительно существовать

umask EXPR – Установка маски по умолчанию на время сессии для всех операций по работе с файлами и каталогами

unlink LIST – удалить файлы из списка

utime LIST – Замена времени доступа и модификации файлов по списку файлов LIST

Выполнение команд

caller EXPR – Возвращает имена запущенных вызовах EXPR кадров стека назад, **0** – текущий стек
($package, $filename, $line) = caller; # список по умолчанию

continue, goto, last, next, redo – см. раздел **«Управляющие структуры»**

do {BLOCK} – Выполняет блок и выполняет результат в виде результата последней выполнившейся операции.

dump LABEL – переход к указанной метке памяти и немедленное выполнение. Не используется.

eval BLOCK – Выполнение кода BLOCK в момент выполнения
eval EXPR – Выполнение кода EXPR в момент компиляции
Функции возвращают ошибки в переменной **$@**

exit EXPR – Вычисление EXPR до целого числа и выход с ним как статусом возвратом

return EXPR – заставляет **sub**, **do** или **eval** завершиться с возвратом результата EXPR

sub – см. раздел **«Управляющие структуры»**

wantarray – возвращает истину если подпрограмма, в которой используется эта команда, ищет списковый контекст. Иначе – ложь

Области видимости

import – Это метод. См. раздел **«Модули. Прагмы»**
use - См. раздел **«Модули. Прагмы»**
local, my, our, package - См. раздел **«Типы данных. Переменные»**

Разное

defined – см. **«Структуры данных»**

formline PICTURE, LIST – формирует данные из LIST исходя из формата PICTURE и складывает результат в **$^A**

reset – см. раздел **«Регулярные выражения»**

scalar EXPR – Вычисление явно скалярного значения. Например, *scalar(ARRAY)* возвращает количество элементов массива.

undef [EXPR] – Вычисляет, если EXPR опущено, или устанавливает EXPR (как lvalue) в «неопределенность». Вычисляется с помощью **defined()**

Процессы и группы

alarm EXPR – посылает текущему процессу сигнал SIGALRM по истечению EXPR секнд

exec [PATHNAME] LIST – Завершает выполнение текущей программы и выполняет команду из LIST. PATHNAME может содержать прямое указание, что в качестве обработчика команды LIST нужно сделать программу PATHNAME

fork – Делает из одного процесса – два. С помощью системного **fork()**

getpgrp, getppid, getpriority, setpgrp, setpriority – здесь не объясняются

kill SIGNAL, LIST – Посылает сигнал списку процессов

pipe READHANDLE, WRITEHANDLE – Открытие пары связанных каналов

qx/STRING/ - см. раздел **«Типы данных. Переменные»**

sleep EXPR – Сделать паузу в программе на EXPR секунд

system [PATHNAME] LIST – аналог **exec()**, но возвращается по окончанию своей работы с кодом возврата.

times – возвращает время работы пользователя.

```
$start = times();
…
$finish = times();
print $finish -$start; # время работы блока …
```

В списковом контексте возвращается детальный список времени загрузки процессора как пользователем, так и системой как текущего процесса, так и дочерних:

```
($user, $system, $cuser, $csystem) = times();
```

$user – время загрузки CPU при работе пользователя
$system – время загрузки CPU при работе системы
$cuser - $user для потомка
$csystem - $system для потомка

Но при этом сумма этих значений равна разнице *($finish -$start)*

wait – ожидает завершения выполнения дочернего процесса вернув PID или **-1** если процессов таких нет.

waitpid PID, FLAGS – ожидает конкретного дочернего процесса и ведет себя с ним также, как и **wait()**

Библиотеки и модули

do, import, no, require, use – см. Раздел **«Модули. Прагмы»**
package – см. Раздел **«Типы данных. Переменные»**

Классы и объекты

bless REF[, CLASSNAME] – прикрепление REF к классу CLASSNAME или текущему классу, если CLASSNAME опущен («благословление»). Функция возвращает ссылку на созданный экземпляр этого класса – объект. См. раздел **«Модули. Прагмы»**

package – см. раздел **«Типы данных. Переменные»**

ref EXPR – возвращает значения (**SCALAR, ARRAY, HASH, CODE, GLOB, REF, LVALUE** или **IO::Handle**) в зависимости от EXPR. EXPR должна быть скаляром, содержащим ссылку на указанный тип структур.

tie, tied, untie - см. раздел **«Связывание»**

use – см. раздел **«Управляющие структуры»**

Время

gmtime EXPR – возвращает массив временных величин по гринвичскому меридиану:
($sec, $min, $hour, $mday, $mon, $year, $wday, $yday, $isdst) = gmtime(time).

$sec – секунды
$min – минуты
$hour – часы
$mday – день
$mon - месяц
год = **$year** + 1900
$wday – день недели
$yday – день от начала года
$isdst – 1 - летнее время, 0 – зимнее

Здесь все числа, кроме **$year** и **$isdst**, отсчитываются от **0**, а не от **1**, как в привычном понимании.

localtime EXPR - возвращает массив временных величин для текущей страны. Результаты в том же составе, как и **gmtime**

time – Количество секунд от начала 1970 года без учета високосных лет.

times – см. раздел **«Процессы и группы»**

Низкоуровневый доступ к сокетам

accept, bind, connect, getpeername, getsockname, getsockopt, listen, recv, send, setsockopt, shutdown, socket, socketpair – здесь не объясняются

System V межпроцессорное взаимодействие

msgctl, msgget, msgrcv, msgsnd, semctl, semget, semop, shmctl, shmget, shmread, shmwrite – здесь не объясняются

Пользователи и группы

endgrent, endhostent, endnetent, endpwent, getgrent, getgrgid, getgrnam, getlogin, getpwent, getpwnam, getpwuid, setgrent, setpwent – здесь не объясняются

Сетевая информация

endprotoent, endservent, gethostbyaddr, gethostbyname, gethostent, getnetbyaddr, getnetbyname, getnetent, getprotobyname, getprotobynumber, getprotoent, getservbyname, getservbyport, getservent, sethostent, setnetent, setprotoent, setservent – здесь не объясняются

Термины

Терм – Синтаксическая конструкция. Члены функций. Все что называется «сложные предложения» (**for{} foreach{} sub{}** и т.д.)

Лексема – Единица языка, имеющая лексический смысл для интерпретатора

Литерал – Набор символов, сегментов строк и т.д., вплоть до конструкций типа “abc” или **__PACKAGE__**

Скаляр – единственная величина

Переменная – именованная величина, необязательно в единственном числе