**Прекрасный язык Perl**

**Perl** **(Practical Extraction and Report Language)**- очень удобный язык, впитавший из других все лучшие черты. Может у вас возникнуть сомнение: Ну вот! Изучать новый язык программирования!? Спешу вас успокоить, изучение **Perl** не будет в тягость. Вы даже сами не заметите, как выучите его. Для **Perl** родной платформой является **Unix** поэтому пользователям PC он мало известен.   
  
Все в нем сделано для удобства программиста.  
Начнем с переменных, они в **Perl** бывают 3х типов *скалярные*, *списковые(массивы)* и *хэши (ассоциативные массивы)*. Для указания компилятору (да и для немалого удобства программиста) перед именем скалярной переменной стоит знак **'$'** перед массивом **'@'**, перед хешем **'%'**. т.е. например **$scalar\_var, @array\_var, %hash\_var.** Скалярные переменные могут быть как числовые так и строковые, но это не надо указывать **Perl** сам по контексту в зависимости от операций может привести одно к другому.  
Например: **"123"+"4"** будет **127** (или **"127"**) так как операция **'+'** действует над числами, а вот если применить операцию конкатенации строк **'.',** то строковое **"test" . 1** будет **"test1"**

Вот операции над скалярными переменными:

|  |  |  |
| --- | --- | --- |
| **Операции** | **Описание** | **Пример** |
| **+ - \* / %** | Арифметические | **print 2\*7+4/(8%3); print int(127/15); #целая часть** |
| **\*\*** | Возведение в степень | **print 2\*\*16;** |
| **++ --** | Инкремент-декремент | **$i++;** |
| **& | ^ ~ << >>** | Побитовые | **$x=3;$y=4; print $x|$y; print $x&$y;** |
| **== != < > <= >= <=>** | Числовые операции сравнения | **if($x==9){print "Ok!";}** |
| **eq ne lt gt le ge cmp** | строковые операции сравнения | **if($game eq 'doom'){print "You are doomer!\n";}** |
| **|| && !** | Логические | **if(($x==9)||($game eq 'doom')){print "hello you!\n";}** |
| **?:** | Условный оператор | **$x=($game eq 'quake'?9:8);** |
| **,** | Последовательное вычисление | **$x=10,$y=20;** |
| **.** | Конкатенация | **$x='http://'.'www.uic.nnov.ru';** |
| **X** | Повторение | **$x='1234'x5; #$x='12341234123412341234'** |
| **=~** | Сопоставление с образцом | **if($url=~/http/){print "HTTP";}** |
| **!~** | То же но с отрицанием | **if($url!~/http/){print "No HTTP";}** |
| **= += -= \*= /= %= \*\*= |= &= ^= ~= <<= >>= .= x=** | Присваивание | **$x+=$y;** |
|  |  |  |

Пусть это будет вам справочником. Насчет строк, вы заметили, что они могут быть в двойных и одинарных кавычках, разница между ними состоит в том, что в одинарных не осуществляется подстановка переменных, а в двойных осуществляется. Например:

**$x='qwerty';**

**print 'my var is $x'; #выведет my var is $x**

**print "my var is $x"; #выведет my var is qwerty**

Списки:

Списочные переменные начинаются с символа **'@'** конструируются следующим образом

**@List1=(1,2,5,70);**

**@List2=(12,23,@List1); #12,23,1,2,5,70**

**@Rgb=($r,$g,$b);**

Также можно список использовать как **lvalue**:

**@List=(1,2,3..8,15);**

**($x,$y,$z)=@List; #$x=1,$y=2,$z=3**

**($x,$y,$z,@list2)=@List; #$x=1,$y=2,$z=3,@list2=(4,5,6,7,8,15);**

**($r,$g,$b)=@Rgb;**

Можно обращаться к нескольким выбранным элементам массива(срезу массива):

**@list=(1..10);**

**@list[2,3,5,9]=(100,200,300,400); #@list=(1,100,200,4,300,6,7,8,400,10)**

**@list[1,10]=@list[10,1];#меняет местами элементы**

Обратиться к скалярному значению - элементу массива можно **$имя\_массива[индекс]**, здесь обратите внимание на знак **'$'**- мы ведь обращаемся к скаляру-элементу.

Теперь немного о *хешах*:  
хеш это такой массив который состоит из пар ключ-значение, весь хеш обозначается **%хеш**, к отдельным элементам доступ **$хеш{скалярное выражение},** конструируется хеш так:

**$my\_hash{1}="doom";**

**$my\_hash{'quake'}="www.idsoftware.com";**

**$my\_hash{1+2}=100;**

Хеш может быть также сконструирован из массива с четным числом элементов, где пары превращаются в ключ-значение:

**%hash=(1,20,2,100);#аналогично $hash{1}=20;$hash{2}=100;**

удаление из хеша - операция delete:

**delete $hash{1};**

Есть функции выдающие ключи и значения соответственно.

**%hash=(1,20,2,100,3,'doom');**

**@k=keys %hash; #@k=(1,2,3);**

**@v=values %hash;#@v=(20,100,'doom');**

Операторы:  
Набор операторов в **Perl** очень широк, многие из них прямые аналоги имеющихся в других языках, например **if, for, while;** но есть и значительные улучшения имеющихся и конечно новые.

Оператор **if** имеет две формы (как когда удобнее):

**if(условие)оператор;**

**оператор if условие;**

В пару к оператору **if** имеется оператор **unless** : означающий **if** с отрицанием:

**unless(($method eq 'GET')||($method eq 'POST')){print "Unsupported method";}**

**print "Ok" unless $x < $y;**

Также в пару **while** существует **until**  
синтаксис оператора **for** полностью аналогичен **C**:

**for($i=0;$i<10;$i++)**

**{**

**print $i;**

**}**

Новшеством является **foreach,** позволяющий пройтись по всем элементам массива, присваивая по очереди его элементы какой-то переменной, его синтаксис такой:

**foreach $переменная (@массив)**

**{**

**блок операторов;**

**}**

**или**

**foreach (@массив)**

**{**

**операторы;**

**}**

Последний пример особенно важен для упрощения вашего тяжкого труда программиста и демонстрирует интересную особенность **Perl** - переменную по умолчанию **$\_**: в огромном количестве операторов и функций при опускании аргумента она подразумевается по умолчанию.

Она также по умолчанию сопоставляется с регулярным выражением:

**следующий пример**

**@Data=<STDIN>; //Аналогично READ в паскале, только здесь считываются все строки в массив @Data**

**foreach(@Data)**

**{**

**chomp;**

**print if /^From:/;**

**}**

**аналогичен такому:**

**@Data=<STDIN>;**

**foreach $\_ (@Data)**

**{**

**chomp($\_);**

**print $\_ if $\_ =~ /^From:/;**

**}**

как видите затраты труда значительно сокращаются, благодаря этому маленькому трюку.  
  
***Регулярные выражения.***  
регулярное выражение записывается между двух слэшей **/рег\_выр/**

**if(/abc/)**

**{**

**print '$\_ содержит abc\n';**

**}**

это самый простой пример применения регулярного выражения. А теперь посложнее вот тут в табличке:

|  |  |  |
| --- | --- | --- |
| **Символ** | **Значение** | **Пример применения** |
| **.** | Соответствует любому символу | **print if /ab.c/;** |
| **[мн-во симв]** | Соответствует любому символу из данного мн-ва | **/[abc]d/;#соответствует ad,bd,cd** |
| **[^мн-во]** | Отрицание мн-ва символов | **/[^xyz]/;#** |
| **(....)** | Группировка элементов(и также запоминание в переменных **$1 $2 $3 ...**) | **/(xyz)\*/ /([abc].[^xy]qwerty)/** |
| **(..|..|..)** | Одна из альтернатив |  |
| **\*** | повторение образца 0 или более раз | **/.\*/;#соответствует всему** |
| **?** | Повторение 0 или 1 раз | **/(http:\/\/)?.\*\.cgi/** |
| **+** | Повторение 1 или более раз |  |
| **{n,m}** | повторение от n до m раз |  |
| **{n}** | повторение точно n раз |  |
| **{n,}** | повторение n и более раз |  |
| *Спец символы:* |  |  |
| **\t \r \n ...** | Управляющие символы:табуляции,возврат каретки,перевод строки..... |  |
| **\d** | Соответствует цифре,Аналог [0-9] |  |
| **\D** | Соответствует нецифровому символу, аналог[^0-9] |  |
| **\w** | Соответствует букве |  |
| **\W** | Соответствует небуквеному символу |  |
| **\s** | Соответствует пробельным символам(пробелы, табуляции, новые строки..) |  |
| **\S** | Соответствует непробельному символу |  |
| **\b** | Соответствует границе слова | **$test1="this is test"; $test2="wise"; if($test1=~/\bis\b/){print "1";}#соответствует if($test2=~/\bis\b/){print "2";}#нет** |
| **\B** | Соответствует не границе слова | **/\Bis\B/ соответсвует 'wise' но не 'is'** |
|  |  |  |

Это далеко не полный список.

Для того чтоб поместить в регулярное выражение любой специальный символ, поставьте перед ним обратный слэш. Заставить **Perl** игнорировать регистр можно поставив **i** после регулярного выражения

**print "Are you sure?:";**

**$answer=<STDIN>;**

**if($answer=~/Y/i)**

**{**

**#че-нибудь сделаем...**

**}**

Полезные функции.  
В **Perl** очень много различных функций, как говорится на все случаи жизни, все о них я конечно не опишу, но обо многих. Начну с тех, которые больше относятся к операторам. Операция замены **s/рег.выражение/строка/** игнорировать регистр - опция **i** глобальная(по всей строке) замена -опция **g**; Пример:

**$x="This is test";**

**$x=~s/ /\_/g;**

**print $x; #This\_is\_test**

Очень полезная опция у **s///** **e** -она означает что вторая строка не строка а выражение, результат которого и будет подставлен. Например, у вас есть файл в котором все записи о возрасте через год надо менять

**open(OLD,"oldfile.txt") || die "Cannot open oldfile.txt $!\n";**

**open(NEW,">newfile.txt") || die "Cannot open newfile.txt $!\n";**

**foreach()**

**{**

**s/(\d+)(\s+год)/($1+1).$2/gie;**

**s/(\d+)(\s+лет)/($1+1).$2/gie;**

**print NEW $\_;**

**}**

**close(NEW);**

**close(OLD);**

или более показательным примером послужит функция **urldecode**, которая будет встречаться в каждой вашей программе, обрабатывающей формы:

**sub urldecode**

**{**

**local($val)=@\_;**

**$val=~s/\+/ /g;**

**$val=~s/%([0-9a-hA-H]{2})/pack('C',hex($1))/ge;**

**return $val;**

**}**

Также важным удобством в **Perl** являются операции для работы с файлами для выполнения схожих функций в других языках приходиться проделывать огромную массу работы. Аргументами могут быть как файловые переменные, так и строки, представляющие имя файла.

функция **open** открывает файл

**open(ФАЙЛОВАЯ\_ПЕРЕМЕННАЯ,"имя файла"); #открыть файл для чтения**

**open(ФАЙЛОВАЯ\_ПЕРЕМЕННАЯ,">имя файла"); #для записи**

**open(ФАЙЛОВАЯ\_ПЕРЕМЕННАЯ,">>имя файла");#для записи в конец**

**open(ФАЙЛОВАЯ\_ПЕРЕМЕННАЯ,"+<имя файла");#для чтения и записи**

**open(ФАЙЛОВАЯ\_ПЕРЕМЕННАЯ,"|комманда"); #направить информацию на вход программы**

**open(ФАЙЛОВАЯ\_ПЕРЕМЕННАЯ,"комманда|"); #считать информацию с выхода программы**

**open(ФАЙЛОВАЯ\_ПЕРЕМЕННАЯ,"|комманда|"); #и то и другое вместе**

Что касается открытия файлов, то вам как программистам все очевидно, но с командами тоже все здорово, что пояснит хороший пример (из практики):

**open(MAIL,"|mail paaa@uic.nnov.ru");#Пошлем информацию по почте**

**print MAIL "Hello\n";**

**print MAIL "...\n";**

**print MAIL "...\n";**

**close(MAIL);**

когда вы открыли файл вы можете считать из него строку в скалярную переменную. Вот так:**$str=<FILE> .**  
Избавиться от символа новой строки на конце поможет функция **chomp**, ведь этот символ может помешаться, например в имени файла или при выводе на экран

**print "Введите имя файла:";**

**$fname=<STDIN>;**

**chomp($fname);**

**open(F,$fname)|| die "Cannot open $fname $!\n";**

**.....**

Если также подставить списочную переменную, то получим список строк файла от текущей строки и до конца

**print "Что искать:";**

**$search=<STDIN>;**

**chomp($search);**

**@L=<F>;**

**foreach(@L)**

**{**

**print if /$search/;**

**}**

**а можно и так:**

**print "Что искать:";**

**$search=<STDIN>;**

**chomp($search);**

**foreach(<F>)**

**{**

**print if /$search/;**

**}**  
функции **split** и **join**:

**@Список=split(/рег.выр/,$скаляр);**  
**$скаляр=join(строка,@Список);**

**#Разбить строку слов, разделенных пробелами в список вы можете**

**@WordList=split(/ /,$String);**

**#После обработки снова объединить**

**$String=join(' ',@WordList);**

Встроенные функции **Perl** можно вызывать со скобками или без (как вам удобно), скобки программисты указывают или для красоты, или чаще, чтобы устранить возможную неоднозначность в выражении:

**printf "x=%d",$x;**

**printf ("x=%d",$x);#аналогично**

Более подробную информацию о **Perl** вы можете получить по адресам:  
<http://www.perl.com>  
<http://www.metronet.com/0/perlinfo/perl5/manual/perl.html>  
<http://www.ActiveWare.com/>