# Учебник по PHP

Данный справочник создан на основе материалов, находящихся на сайте Справочник Web-языков www.spravkaweb.ru
Дата создания справочника: 20.08.2004

Если Вы нашли ошибки в справочнике, или у Вас есть предложения по его улучшению, прошу писать на майл.

Скачать обновленный справочник можно отсюда

## Содержание:

## Выражения

**if**
Позволяет организовывать выполнение фрагментов кода по условию.

**Синтаксис :**
if (*выражение*) statement

Может иметь неограниченную степень вложенности в другие IF.

```
if($a>$b)
  print "$a больше $b";
if($a>$b){echo "$a больше $b;$b=$a;}
```

---

**else**
Расширяет возможности IF по части обработки вариантов выражения, когда оно равно FALSE.

Выражение ELSE выполняется только если IF равно FALSE.

```
if ($a>$b) {
  echo "a больше b";
} else {
  echo "a меньше b";
}
```

**elseif**
Является комбинацией IF и ELSE. Позволяет выполнить выражение, если значение IF равно FALSE, но в отличии от ELSE оно выполниться, если выражение ELSEIF равно TRUE.

```
if ($a>$b) {
  echo "a больше b";
} elseif ($a==$b) {
  echo "a равно b";
} else {
  echo "a меньше b";
}
```

**if_endif**
Один из возможных вариантов группирования операторов с оператором IF.

Удобно при внедрении больших блоков HTML-кода внутрь оператора IF.

```
if ($a==1):
  echo "a равно 1";
elseif ($a==2):
  echo "a равно 2";
else:
  echo "a не равно 1 и 2";
endif;
<?php if ($a==5):?>A=5<?php endif;?>
-Блок HTML-кода A=5 будет виден,
     если выполнется условие $a==5
```

**while**
Простейший тип цикла в PHP. Заставляет PHP выполнять вложенные операторы до тех пор, пока *условие* равно TRUE. Если *условие* равно FALSE с самого начала, то цикл не выполнится не разу.

**Синтаксис :** WHILE(*условие*)выражения

Можно сгруппировать несколько операторов внутри фигурных скобок или использовать **альтернативный синтаксис :**
WHILE(*условие*)выражения... ENDWHILE;

```
$a=1;
while ($a<=5) {
  echo $a++; }
$a=1;
while ($a<=5):
  echo $a;
  $a++;
endwhile;
```

- Эти два примера выводят номера с 1 до 5.

**do_while**
Цикл, аналогичный WHILE, но значение логического выражения проверяется не до, а после окончания итерации. Основное отличие - то что цикл хоть один раз но выполнится.

```
$a=1;
do {
  echo $a;
} while ($a>1);
```

Можно прекратить использовать блок операторов в середине путем внедрения оператора **BREAK** в цикл **DO..WHILE(0)** :

```
do {
  if ($a==5) {
    echo "A равно 5"
    break;
  }
  $a *= $b;
  if ($a<$minimum) {
    break;
  }
  echo "A равен $a";
} while(0);
```

---

**for**
Наиболее мощный цикл в PHP.

**Синтаксис :**
FOR (*условие1*; *условие2*; *условие3*)выражения

*условие1* - Безусловно выполняется (вычисляется) в начале цикла
*условие2* - Проверяется в начале каждой итерации. Если оно равно TRUE, то цикл продолжается и выполняются вложенные операторы. Если оно равно FALSE, то цикл заканчивается.
*условие3* - Выполняется (вычисляется) в конце каждой итерации.
Каждое из этих условий может быть пустым.

**Пример 1:**
```
for ($a = 1; $a <= 5; $a++) {
  echo $a;
}
```
**Пример 2:**
```
for ($a = 1;;$a++) {
  if ($a > 5) {
    break;
  }
  echo $a;
}
```
**Пример 3:**
```
$a = 1;
for (;;) {
  if ($a > 5) {
    break;
  }
  print $a;
  $a++;
}
```
**Пример 4:**
```
for ($a = 1; $a <= 5; print $a, $a++);
```
PHP поддерживает альтернативный синтаксис FOR:
FOR(*усл1; усл2; усл3;*):операторы;...;ENDFOR;

---

**break**
Прерывает выполнение текущего цикла.

**Пример :**
```
$a = 0;
while ($a < 5) {
  if ($arr[$a] == "stop") {
    break;
  }
  $a++;
}
```

---

**continue**
Переходит на начало ближайшего цикла.

```
while (list($key,$value) = each($arr)) {
  if ($key % 2) {
    continue;
  }
  do_something_odd ($value);
}
```

---

**switch**
Сравнивает переменную или выражение с различными значениями и выполняет различные фрагменты кода в зависимости от того, чему будет равно значение выражения.

```
switch ($a) {
  case 0:
    echo "A равно 0";
    break;
  case 1:
    echo "A равно 1";
    break;
  case 2:
    echo "A равно 2";
    break;
  default:
    echo "A не равно 0, 1, 2";
}
```

default - соответствует всем значениям, которые не удовлетворяют другим CASE.
CASE - могут быть любого скалярного типа, т.е. целые числа или числа с плавающей запятой и строки.

---

**require**
Заменяет себя содержимым указанного файла.

**Пример :**
require("include.inc");

Но его нельзя поместить внутрь цикла и ожидать, что он включит содержимое другого файла несколько раз в процессе каждой итерации. Для этого есть INCLUDE.

---

**include**
Вставляет и выполняет содержимое указанного файла.

```
$files = array ("first.inc", "second.inc", "third.inc");
for ($a = 0; $a < count($files); $a++) {
  include($files[$a]);
}
```

Так как INCLUDE() это специальный оператор, требуется заключать его в фигурные скобки при использовании внутри условного оператора.

```
if ($a < 5) {
  include("file_1.inc");
} else {
  include("file_2.inc");
}
```

---

**function**
Объявление функции.

Внутри функции может быть любой верный код PHP, даже объявление другой функции или класса. Функции должны быть объявлены перед тем, как на них ссылаться.

```
function foo ($arg_1, $arg_2,...,$arg_n) {
  echo "Пример функции.";
  return $retvalue;
}
```

***Возвращение результатов :***

Результаты возвращаются через необязательный оператор return.
Возвращаемый результат может быть любого типа, включая списки и объекты.

```
function my_sqrt ($num) {
  return $num * $num;
}
echo my_sqrt(4); //выведет 16
```

Множественные результаты не могут быть возвращены в качестве результата, но вы можете реализовать это путем возврата списка :

```
function foo() {
  return array (0, 1, 2);
}
list ($zero, $one, $two) = foo();
```

***Аргументы :***

Информация может быть передана функции через список аргументов, которые являются разделенным запятыми списком переменных и/или констант.
Списки аргументов переменной длины не поддерживаются, но того же можно достичь, передавая массивы.

```
function takes_array($input) {
  echo "$input[0] + $input[1] = ", $input[0]+$input[1];
}
```

***Передача по ссылке :***

По умолчанию, аргументы функции передаются по значению. Для изменения аргументов в функции их надо передавать по ссылке.
Для этого надо поставить амперсанд (&) перед именем аргумента в объявлении функции :

```
function foo( &$bar) {
  $bar .= "и добавочная строка.";
}
$str = "Это строка, ";
foo($str);
echo $str;   // выведет : "Это строка, и добавочная строка."
function foo($bar) {
  $bar .= "и добавочная строка.";
}
$str = "Это строка, ";
foo($str);
echo $str;  //выведет : "Это строка, "
foo(&$str);
echo $str;  //выведет : "Это строка, и добавочная строка."
```

***Значения по умолчанию :***

Значение по умолчанию должно быть константой, а не переменной или членом класса.

```
function day ($type = "понедельник") {
  echo "Сегодня $type.";
}
echo day();           //выведет : Сегодня понедельник.
echo day("вторник");  //выведет : Сегодня вторник.
```

Аргументы по умолчанию при описании должны находиться справо от остальных аргументов.

```
function day($day_num, $type = "понедельник") {
  return "Сегодня $day_num - $type.";
}
```

**old_function**
Оператор OLD_FUNCTION позволяет вам определяить функцию используя синтаксис PHP/FI2 (за исключением того, что вы должны заменить "function" на "old_function").

Это свойство только для совместимости и должно использоваться лишь конверторами PHP/FI2 -> PHP3. Описанные таким образом функции не могут быть вызваны из служебного кода PHP. Вы можете обойти это путем введения специальной функции в терминах PHP3, которая будет вызывать OLD_FUNCTION.

---

**class**
Набор переменных и функций, работающих с этими переменными.

```
<?php
class Cart {
  var $items; // Количество вещей в корзине покупателя
  // Добавить $num наименований типа $artnr в корзину
  function add_item ($artnr, $num) {
    $this->items[$artnr] += $num;
  }
  // Убрать $num наименований $artnr из корзины
  function remove_item ($artnr, $num) {
    if ($this->items[$artnr] > $num) {
        $this->items[$artnr] -= $num;
        return true;
    } else {
      return false;
    }
  }
}
?>
```

Классы это типы, то есть, заготовки для реальных переменных. Вы должны создавать переменные желаемого типа, используя оператор new :
$cart = new Cart;
$cart->add_item("10", 1);

Классы могут быть расширениями других классов. Расширенный класс обладает всеми переменными и функциями базового класса и тем, что вы определите при расширении класса. Это делается используя ключевое слово extends :

```
class Named_Cart extends Cart {
  var $owner;
  function set_owner ($name) {
    $this->owner = $name;
  }
}
```

Это определяет класс Named_Cart, который имеет все переменные и функции класса Cart плюс дополнительную переменную $owner и дополнительную функцию set_owner(). Вы можете создать поименованую корзину обычным образом и установить или получить владельца корзины. Также вы можете использовать и нормальные функции корзины в поименованой корзине :

```
$ncart = new Named_Cart;   //Создать корзину
$ncart->set_owner ("kris");//Указать владельцаprint
$ncart->owner;           //Распечатать имя владельца корзины
$ncart->add_item ("10", 1);//унаследовано из обычной корзины
```

---

## Математические функции : Функции округления

**abs**
Возвращает модуль числа.

**Синтаксис :**
mixed abs(mixed $number)

Тип параметра *$number* может быть float или int, а ти п возвращаемого значения всегда совпадает с типом этого параметра.

```
$x = abs(-4);    // $x=4
$x = abs(-7.45); // $x=7.45
```

---

**round**
Округление дробного числа до целого.

**Синтаксис :**
double round(double $val)

Округляет *$val* до ближайшего целого и возвращает результат.

```
$foo = round(3.4); // $foo == 3.0
$foo = round(3.5); // $foo == 4.0
$foo = round(3.6); // $foo == 4.0
$x = round(5.3);  // $x=5
$x = round(5.4);  // $x=5
$x = round(5.45); // $x=5
$x = round(5.5);  // $x=6
```

---

**ceil**
Дополнение дробного числа до следующего целого.

**Синтаксис :**
int ceil(float $number)

Возвращает наименьшее целое число, не меньше *$number*. Разумеется, передавать в *$number* целое число бессмысленно.

```
$x = ceil(5.0);  // $x=5
$x = ceil(5.1);  // $x=6
$x = ceil(5.9);  // $x=6
```

---

**floor**
Удаление дробной части числа.

**Синтаксис :**
int floor(float $number)

Возвращает максимальное целое число, не превосходящее *$number*.

```
$x = floor(5.1);  // $x=5
$x = floor(5.9);  // $x=5
```

---

## Математические функции : Случайные числа

**srand**
Производит инициализацию генератора случайных чисел.

**Синтаксис :**
void srand(int seed)

Инициализирует генератор случайных чисел занчением seed.

```
srand((double) microtime()*1000000);
$random = rand();
echo $random;
```
Вот что получиться:
```
5645
```

---

**getrandmax**
Возвращает максимально возможное случайное число.

**Синтаксис :**
int getrandmax()

Эта функция возвращает максимальное значение, которое можно получить при помощи функции генерации случайных чисел rand().
Обычно это 32767

---

**rand**
Производит генерацию случайного числа.

**Синтаксис :**
int rand([int max [, int min]])

При вызове с необязательными параметрами min и max эта функция генерирует случайное число, лежащее в пределах этих параметров включительно.
Если параметры min и max отсутствуют, возвращается число, лежащее в пределах от 0 до RAND_MAX.
Для корректной работы данной функции перед ее использованием нужно проинициализировать генератор случайных чисел функцией srand().

---

**mt_rand**
Функция возвращает MT-случайное число, достаточно равномерно даже для того, чтобы использовать его в криптографии.

**Синтаксис :**
int mt_rand(int $min=0, int $max=RAND_MAX)

Если вы хотите генерировать числа не от 0 до *RAND_MAX* (эта константа задает максимально допустимое случайное число, и ее можно получить при помощи вызова **mt_getrandmax()**), задайте соответствующий интервал в параметрах *$min* и *$max*. Не забудьте только перед первым вызовом этой функции запустить **mt_srand()**.

```
mt_srand(time()+(double)microtime()*1000000);
$x = mt_rand(1,100); // $x - значение от 1 до 100
```

---

**mt_srand**
Настраивает MT-генератор случайных чисел на новую последовательность.

**Синтаксис :**
void mt_srand(int seed)

Дело в том, что хотя числа, генерируемые mt_rand(), достаточно равновероятны, но у них есть один нелостаток: последовательность сгенерированных чисел будет одинакова если сценарий вызывать несколько раз подряд. Функция **mt_srand()** как раз решает данную проблему: она выбирает новую последовательность на основе параметра *$seed*, причем практически непредсказуемым образом.
```
mt_srand(time()+(double)microtime()*1000000);
for($i=0;$i<=10;$i++) {
```

```
  $x = mt_rand(1,10);
};
```
В этом случае последовательность устанавливается на основе времени завуска сценария (в секундах), поэтому она достаточно непредсказуема. Для еще более надежного результата рекомендуется приплюсовать сюда еще микросекунды (что и было сделано), а также идентификатор процесса, вызывавшего сценарий.

---

**mt_getrandmax**
Возвращает максимальное MT-случайное число.

**Синтаксис :**
int mt_getrandmax()

Возвращает максимальное число, которое может быть сгенерированно функцией mt_rand() - иными словами, константу *RAND_MAX*

```
$max = mt_getrandmax();
// $max = 2147483647
```

---

**lcg_value**
функция генерирует случайное дробное число.

**Синтаксис :**
double lcg_value()

Эта функция возвращает псевдослучайное дробное число в диапазоне от 0 до 1.

---

## Математические функции : Перевод в различные системы счисления

**base_convert**
Конвертация числа из одной системы счисления в другую.

**Синтаксис :**
string base_convert(string $number, int $frombase, int $tobase)

Переводит число *$number* (заданное как строка в системе счисления по основанию *$frombase*) в систему по основанию *$tobase*. Параметры *$frombase* и *$tobase* могут принимать значения только от 2 до 36 включительно. В строке *$number* цифры обозначают сами себя, а буква *a* соответствует 11, *b* -12, и т.д. до *z*, которая обозначает 36. Например, следующие команды выведут 11111111 (8 единичек), потому что это - не что иное, как представление шестнадцатиричного числа *FF* в двоичной системе счисления:

```
$x = base_convert("FF",16,2);       //$x = 11111111
$x = base_convert("11111111",2,16); //$x = FF
$x = base_convert("200",10,16);     //$x = C8
```

---

**bindec**
Производит конвертацию двоичного числа в десятичное.

**Синтаксис :**
int bindec(string binary_string)

Преобразует двоичное число, заданное в строке *binary_string*, в десятичное число. Максимальное число, которое еще может быть преобразовано, равно 2 147 483 647

```
$x = bindec(11111111); // $x = 255
$x = bindec(10101010); // $x = 170
$x = bindec(2147483647); // $x = 11111111111111111111111111111111
```

---

**decbin**
Производит конвертацию десятичного числа в двоичное.

**Синтаксис :**
string decbin(int $number)

Возвращает строку, представляющую собой двоичное представление целого числа *$number*. Максимальное число, которое еще может быть преобразовано, равно 2 147 483 647, которое выглядит как 31 единичка в двоичной системе.

Существует аналогичные функции для восьмеричной и шестнадцатиричной систем. Называются они так же, только вместо "bin" подставляются соответственно "oct" и "hex".

```
$x = decbin(255); // $x = 11111111
$x = decbin(2147483647); // $x = 11111111111111111111111111111111
```

---

**dechex**
Производит конвертацию десятичного числа в шестнадцатеричное.

**Синтаксис :**
string dechex(int number)

Возвращает строку, представляющую собой шестнадцатеричное представление целого числа *number*. Максимальное число, которое еще может быть преобразовано, равно 2 147 483 647

```
$x = dechex(2147483647); // $x = 7fffffff
```

---

**decoct**
Производит конвертацию десятичного числа в восьмеричное.

**Синтаксис :**
string decoct(int number)

Возвращает строку, представляющую собой восьмеричное представление целого числа *number*. Максимальное число, которое еще может быть преобразовано, равно 2 147 483 647

```
$x = dechex(2147483647); // $x = 17777777777
```

---

**hexdec**
Производит конвертацию шестнадцатеричного числа в десятичное.

**Синтаксис :**
int hexdec(string hex_string)

Преобразует шестнадцатеричное число, заданное в строке *hex_string*, в десятичное число. Максимальное число, которое еще может быть преобразовано, равно 7fffffff

```
$x = hexdec(7fffffff); // $x = 2147483647
```

---

**octdec**
Производит конвертацию восьмеричного числа в десятичное.

**Синтаксис :**

int octdec(string octal_string)

Преобразует восьмеричное число, заданное в строке *octal_string*, в десятичное число. Максимальное число, которое еще может быть преобразовано, равно 17777777777

```
$x = octdec(17777777777); // $x = 2147483647
```

---

**deg2rad**
Производит конвертацию градусов в радианы.

**Синтаксис :**
double deg2rad(double number)

Преобразует градусы, заданные в параметре *number*, в радианы.

---

**rad2deg**
Производит конвертацию радианов в градусы.

**Синтаксис :**
double rad2deg(double number)

Преобразует радианы, заданные в параметре *number*, в градусы.

---

**number_format**
Форматирование числа.

**Синтаксис :**
number_format($number, $decimals, $dec_point=".", $thousands_sep=",");

Эта функция форматирует число с плавающей точкой с разделением его на триады с указанной точностью. Она может быть вызвана с двумя или четырьмя аргументами, но не с тремя! Параметр *$decimals* задает, сколько цифр после запятой должно быть у числа в выходной строке. Параметр *$dec_point* представляет собой разделитель целой и дробной частей, а параметр *$thousands_sep* - разделитель триад в числе (если указать на его месте пустую строку, то триады не отделяются друг от друга).

---

## Математические функции : Минимум и максимум

**min**
Эта функция возвращает наименьшее из чисел, заданных в ее аргументах.

**Синтаксис :**
mixed min(mixed $arg1 [int $arg2, ..., int $argn])

Различают два способа вызова этой функции: с одним параметром или с несколькими. Если указан лишь один параметр (первый), то он обязательно должен быть массивом и возвращается минимальный элемент этого массива. В противном случае первый (и остальные) аргументы трактуются как числа с плавающей точкой, они сравниваются, и возвращается наименьшее. Тип возвращаемого значения выбирается так: если хотябы одно из чисел, переданных на вход, задано в формате с плавающей точкой, то и результат будет с плавающей точкой, в противном случае результат будет целым числом. С помощью этой функции нельзя лексографически сравнивать строки - только числа.

```
$x = min(5,3,4,6,5,6,8,9);
// $x = 3
$x[0]=4;
$x[1]=1;
$x[2]=5;
$x[3]=2;
echo min($x); // выведет 1
```

**max**
Получение наибольшего аргумента.

**Синтаксис :**
mixed max(mixed $arg1 [int $arg2, ..., int $argn])

Функция работает аналогично **min()**, только ищет максимальное значение.

```
$x = max(5,3,4,6,5,6,8,9);
// $x = 9
$x[0]=4;
$x[1]=1;
$x[2]=5;
$x[3]=2;
echo max($x); // выведет 5
```

## Математические функции : Степенные функции

**sqrt**
Возвращает квадратный корень из аргумента.

**Синтаксис :**
float sqrt(float $arg)

Если аргумент отрицателен, то генерируется предупреждение, но работа программы не прекращается!

```
$x = sqrt(9);  // $x = 3
echo sqrt(25); // выведет 5
echo sqrt(-25); // выведет -1.#IND
```

**log**
Возвращает натуральный логарифм аргумента.

**Синтаксис :**
float log(float $arg)

В случае недопустимого числа печатает предупреждение, но не завершает программу.

```
$x = log(exp(2));  // exp(2) - e в степени 2
                   // $x = 2
$x = log(M_E);     // $x = 1
echo log(10);      // выведет 2.302585092994
```

**log10**
Возвращает десятичный логарифм аргумента.

**Синтаксис :**
float log10(float $arg)

В случае недопустимого числа печатает предупреждение, но не завершает программу.

```
echo log10(100); // выведет 2
```

---

**exp**
Возвращает e (2,718281828) в степени *$arg*.

**Синтаксис :**
float exp(float $arg)

```
$x = exp(1); // $x = 2.718281828459
```

---

**pow**
Возведение в степень.

**Синтаксис :**
float pow(float $base, float $exp)

Возвращает *$base* в степени *$exp*.

```
$x = pow(3,2);   // $x = 9
$x = pow("3",2); // $x = 9
```

---

## Математические функции : Тригонометрия

**sin**
Возвращает синус аргумента.

**Синтаксис :**
float sin(float $arg)

Аргумент задается в радианах.

```
$x = sin(M_PI_2); // $x = 1
```

---

**cos**
Возвращает косинус аргумента.

**Синтаксис :**
float cos(float $arg)

```
$x = cos(0); // $x = 0
$x = cos(M_PI); // $x = -1
```

---

**tan**
Возвращает тангенс аргумента, заданного в радианах.

**Синтаксис :**
float tan(float $arg)

```
$x = tan(M_PI_4); // $x = 1
```

---

**acos**
Возвращает арккосинус аргумента.

**Синтаксис :**
float acos(float $arg)

```
$x = acos(0); // $x = pi/2
$x = acos(1); // $x = 0
```

---

**asin**
Возвращает арксинус.

**Синтаксис :**
float asin(float $arg)
```
$x = asin(0); // $x = 0
$x = asin(1); // $x = pi/2
```

---

**atan**
Возвращает арктангенс аргумента.

**Синтаксис :**
float atan(float $arg)
```
$x = atan(0); // $x = 0
$x = atan(1); // $x = pi/4
```

---

**atan2**
Получение арктангенса двух чисел.

**Синтаксис :**
float atan2(float $y, float $x)

Возвращает арктангенс величины *$y/$x*, но с учетом той четверти, в которой лежит точка (*$x,$y*). Эта функция возвращает результат в радианах, принадлежащий отрезку от -? до ?.
```
$x = atan2(1,1);   // $x = pi/4
$x = atan2(-1,-1); // $x = -3*pi/4
```

---

**pi**
Возвращает число пи - 3,14.

**Синтаксис :**
double pi()

Эту функцию обязательно нужно вызывать с парой пустых скобок:

```
$x = pi()*2 // $x = 31.415926535898
```

---

## Математические функции : Функции повышенной точности BCMath

**bcadd**
Сложение двух чисел произвольной точности.

**Синтаксис :**
string bcadd(string left_operand, string right_operand [, int scale]);

Эта функция возвращает строковое представление суммы двух параметров (left_operand + right_operand) с точностью, которая указана в необязательном параметре scale.
Точность (scale) указывает количество десятичных знаков после запятой).

---

**bccomp**
Сравнение двух чисел произвольной точности.

**Синтаксис :**

int bccomp(string left_operand, string right_operand, [int scale]);

Сравнивает числа (left_operand с right_operand) и возвращает результат типа integer (целое). Параметр scale используется для установки количества цифр после десятичной отметки, используемых при сравнении. При равенстве двух частей возвращается значение 0. Если левая часть больше правой части возвращается +1, и если левая часть меньше правой части возвращается -1.

---

**bcdiv**
Операция деления для двух чисел произвольной точности.

**Синтаксис :**
string bcdiv(string left_operand, string right_operand [, intscale]);

Делит left_operand на right_operand и возвращает результат с точностью (знаками после запятой), заданной в параметре scale.

---

**bcmod**
Возвращает остаток целочисленного деления.

**Синтаксис :**
string bcmod(left_operand, string modulus);

Данная функция возвращает остаток от целочисленного деления left_operand на modulus.

---

**bcmul**
Операция умножения для двух чисел произвольной точности.

**Синтаксис :**
string bcmul(string left_operand, string right_operand [, int scale]);

Производит умножение left_operand на right_operand, и выдает результат в виде строки с точностью, заданной в переменной scale.

---

**bcpow**
Возведение одного числа произвольной точности в степень другого.

**Синтаксис :**
string bcpow(string x, string y, [int scale]);

Возведение x в степень y. Параметр scale может использоваться для установки количества цифр после точки.

---

**bcscale**
Устанавливает точность вычислений.

**Синтаксис :**
string bcscale(int scale);

Эта функция устанавливает заданную по умолчанию точность вычислений для всех математических функций BCMath, которые явно не определяют точность.

---

**bcsqrt**
Получение квадратного корня числа произвольной точности.

**Синтаксис :**

string bcsqrt(string operand [,int scale]);

Возвращает квадратный корень аргумента operand. Параметр scale устанавливает количество цифр после десятичной отметки в результате.

---

**bcsub**
Вычитает одно число произвольной точности из другого.

**Синтаксис :**
string bcsub(string left_operand, right_operand [, int scale]);

Возвращает разность двух переменных, указанных в параметрах функции (left_operand - right_operand) с точностью, указанной в необязательном параметре scale.

---

## Математические функции : Функции GMP

Функции этого вида позволяют работать с целыми числами повышенной точности определенного формата используя библиотеку GNU MP.
Эта библиотека не входит в стандартный пакет PHP. Загрузить коды библиотеки и документацию по ней можно на сайте http://www.swox.com/gmp/.

Функции, приведенные в этой библиотеке, могут также работать с обычными целочисленными аргументами. В этом случае они будут автоматически преобразовываться в формат GMP. Но для увеличения производительности рекомендуется все же использовать числа формата GMP.

---

## Математические функции : Функции GMP. Значения функции GMP

**gmp_init**
Создает число GMP.

**Синтаксис :**
resource gmp_init(mixed number)

Число GMP создается из целочисленного или строкового аргумента.
В строке может быть указано число десятеричного или шестнадцатеричного формата. Если это шестнадцатеричный формат, то перед числом должен стоять префикс 0x.

```
$x = gmp_init(45);
$y = gmp_init("46");
$z = gmp_init("0xfa4b");
```

Данная функция не обязательна (аргументы автоматически конвертируются в формат GMP), но желательна (при использовании функции gmp_init() повышается быстродействие).

---

**gmp_intval**
Преобразование GMP-числа в целое.

**Синтаксис :**
int gmp_intval(resource gmpnumber)

Эта функция конвертирует GMP-число в целое в том случае, если получаемое число не превышает своего максимально допустимого размера.

---

**gmp_strval**
Преобразование GMP-числа в строку.

**Синтаксис :**
string gmp_strval(resource gmpnumber [, int base])

Функция возвращает число gmpnumber в строковом формате в системе счисления, заданной в необязательном параметре base. По умолчанию возвращает в десятичной системе счисления).
Параметр base может принимать значения от 2 до 36.

```
$x = gmp_init("0xf1a5");
echo "В десятичной : ".gmp_strval($x);
echo "В base-36: ".gmp_strval($x,36);
```

---

**gmp_abs**
Вычисляет модуль GMP-числа.

**Синтаксис :**
resource gmp_abs(resourse x)

Возвращает абсолютное значение числа, заданного в параметре x.

---

**gmp_sign**
Возвращает знак числа.

**Синтаксис :**
int gmp_sign(resource x)

Функция gmp_sign() возвратит 1, если x - положительное число, и 0 - если отрицателоное.

---

**gmp_neg**
Возвращает отрицательное значение числа.

**Синтаксис :**
recource gmp_neg(resource x)

Возвратит -x.

---

## Математические функции : Функции GMP. Арифметика

**gmp_add**
Сложение двух чисел.

**Синтаксис :**
recource gmp_add(resource x, recource y)

Функция возвратит GMP-число, равное сумме аргументов x и y.

---

**gmp_sub**
Вычитание двух чисел.

**Синтаксис :**
recource gmp_sub(resource x, recource y)

Функция возвратит GMP-число, равное разности аргументов x и y.

---

**gmp_mul**
Умножение двух чисел.

**Синтаксис :**
recource gmp_mul(resource x, recource y)

Функция возвратит GMP-число, равное произведению аргументов x и y.

---

**gmp_div**
Деление двух чисел.

**Синтаксис :**
recource gmp_div(resource x, recource y [, int round])

Функция возвратит GMP-число, равное делению аргументов x на y. В зависимости от необязательного параметра round, результат деления будет округляться следующим образом:
- GMP_ROUND_ZERO - цифры после точки отбрасываются
- GMP_ROUND_PLUSINF - результат деления округляется в большую сторону
- GMP_ROUND_MINUSINF - результат деления округляется в меньшую сторону

Эта функция - синоним gmp_div_q().

---

**gmp_div_q**
Деление двух чисел.

**Синтаксис :**
recource gmp_div_q(resource x, recource y [, int round])

Функция возвратит GMP-число, равное делению аргументов x на y. В зависимости от необязательного параметра round, результат деления будет округляться следующим образом:
- GMP_ROUND_ZERO - цифры после точки отбрасываются
- GMP_ROUND_PLUSINF - результат деления округляется в большую сторону
- GMP_ROUND_MINUSINF - результат деления округляется в меньшую сторону

У этой функции есть синоним - gmp_div().

---

**gmp_div_r**
Возвращате остаток от целочисленного деления.

**Синтаксис :**
recource gmp_div_r(resource x, recource y [, int round])

Функция возвращает остаток от деления x на y. Знак будет наследован от аргумента x.

---

**gmp_div_qr**
Производит деление с остатком.

**Синтаксис :**
array gmp_div_qr(resource x, recource y [, int round])

Данная функция комбинирует в себе действие двух предыдущих функций gmp_div_q() и gmp_div_r(). Она возвращает массив, состоящий из двух элементов: под индексом [0] - целое частное, под индексом [1] - остаток деления.

```
$x=gmp_init("0xf3c3b5");
$result=gmp_div_qr($x, "0xb1");
echo "Целое: ".gmp[strval($result[0]);
echo "Остаток: ".gmp[strval($result[1]);
```

---

**gmp_mod**
Возвращает модуль остатка деления.

**Синтаксис :**
resource gmp_mod(resource x, recource y)

Данная функция эквивалентна gmp_div_r(), за исключением того, что она возвращает абсолютное значение.

---

**gmp_divexact**
Производит безостаточное деление.

**Синтаксис :**
resource gmp_divexact(resource x, recource y)

Данная функция использует алгоритм "точного" деления. Результат будет достоверным, только если x будет нацело делим y.

---

**gmp_cmp**
Производит сравнение двух чисел.

**Синтаксис :**
int gmp_cmp(resource x, recource y)

Функция возвратит положительное значение, если x > y; ноль, елси x = y; отрицательное значени, если x < y.

---

## Математические функции : Функции GMP. Математика

**gmp_fact**
Вычисляет факториал.

**Синтаксис :**
resource gmp_fact(resourse x)

Возвращает факториал числа, заданного в параметре x.

---

**gmp_sqrt**
Вычисляет квадратный корень.

**Синтаксис :**
resource gmp_sqrt(resourse x)

Возвращает квадратный корень числа, заданного в параметре x.

---

**gmp_sqrtrm**
Вычисляет квадратный корень с остатком.

**Синтаксис :**
array gmp_sqrtrm(resourse x)

Данная функция возвращает массив, в котором элемент с индексом [0] - это квадратный корень аргумента, элемент с индексом [1] - разность между аргументом и элементом [0] в квадрате.

---

**gmp_perfect_square**
Определяет, является ли число полным квадратом.

**Синтаксис :**
bool gmp_perfect_square(resourse x)

Функция gmp_perfect_square() возвратит true, если x является квадратом целого числа. В противном случае вернет false.

---

**gmp_pow**
Возведение в степень.

**Синтаксис :**
resource gmp_pow(resourse x, int y)

Эта функция возвращает результат, равный возведения аргумента x в степень y, при условии, что y не отрицательный.

```
echo gmp_pow(2,3); // Выведет 8
echo gmp_pow(0,0); // Выведет 1
```

---

**gmp_powm**
Возвращает остаток деления степени числа.

**Синтаксис :**
resource gmp_powm(resourse x, resource y, resource mod)

Возвращает остаток от деления (x в степени y) на mod, в случае, если y положетелен.

---

**gmp_prob_prime**
Проверка "вероятно" простого числа.

**Синтаксис :**
int gmp_prob_prime(resourse x [, int reps])

Эта функция возвратит 0, в случае если x сложное число, т.е. имеющее более двух целых делителей. Возвратит 1, если x, возможно, простое число. Если возвращает 2, то тогда x - наверняка простое число.
Аргумент reps определяет качество проверки. Чем большее это число, тем точнее результат. Может принимать занчения от 5 до 10 (по умолчанию).

Эта функция использует алгоритм вероятностного теста Миллера-Рабина.

---

**gmp_gcd**
Находит наибольший общий делитель.

**Синтаксис :**

resource gmp_gcd(resourse x, resource y)

Всегда возвращает положительный результат.

---

**gmp_gcdext**
Нахождение наибольшего общего делителя со множителями.

**Синтаксис :**
array gmp_gcdext(resourse x, resource y)

Функция gmp_gcdext() возвращает массив со значениями g,s,t, такими, что x*s+y*t=g=НОД(x,y), гдн НОД - наибольший общий делитель.

---

**gmp_invert**
Производит инверсию по модулю.

**Синтаксис :**
resource gmp_invert(resourse x, resource y)

Функция возвращает дополнение x до значения, делящегося нацело на y.Вслучае, если результат не может быть найден, возвращает false.

---

**gmp_legendre**
Возвращает число Легранжа.

**Синтаксис :**
int gmp_legendre(resourse x, resource p)

Функция возвращает число Легранжа. p должен быть четным положительным.

---

**gmp_jacobi**
Возвращает число Якоби.

**Синтаксис :**
int gmp_jacobi(resourse x, resource p)

Функция возвращает число Якоби. p должен быть четным положительным.

---

**gmp_random**
Производит генерацию случайного числа.

**Синтаксис :**
resource gmp_random(int limited)

limited задает длину генерируемого числа. В случае, если значение limited отрицательно, генерируется отрицательное число.

---

**gmp_popcount**
Получение популяции.

**Синтаксис :**
int gmp_popcount(resource x)

Функция возвращает числитель популяции.

---

**gmp_hamdist**
Вычисление дистанции.

**Синтаксис :**
int gmp_hamdist(resource x, resource y)

Функция возвращает дистанцию между числами x и y. Аргументы x и y должны быть неотрицательными.

---

## Математические функции : Функции GMP. Бинарные операции

**gmp_and**
Логическое И (AND).

**Синтаксис :**
resource gmp_and(resource x, resource y)

---

**gmp_or**
Логическое ИЛИ (OR).

**Синтаксис :**
resource gmp_or(resource x, resource y)

---

**gmp_xor**
Логическое исключающее-ИЛИ (XOR).

**Синтаксис :**
resource gmp_xor(resource x, resource y)

---

**gmp_setbin**
Установка бита.

**Синтаксис :**
resource gmp_setbin(resource &x, int index [, bool set_clear])

Устанавливает бит в позиции index в числе x. Аргумент set_clear указывает, в какое значение устанавливать бит: 0 или 1 (по умолчанию).

---

**gmp_clrbit**
Производит сброс бита.

**Синтаксис :**
resource gmp_clrbit(resource &x, int index)

Устанавливает бит в позиции index в числе x в значение 0.

---

**gmp_scan0**
Производит поиск бита 0.

**Синтаксис :**
0int gmp_scan0(resource x, int start)

Функция gmp_scan0() ищет в числе x бит 0, начиная с позиции start, в сторону увеличения значимости разрядов. Возвращает позицию найденного бита.

**gmp_scan1**
Производит поиск бита 1.

**Синтаксис :**
1int gmp_scan1(resource x, int start)

Функция gmp_scan0() ищет в числе x бит 1, начиная с позиции start, в сторону увеличения значимости разрядов. Возвращает позицию найденного бита.

---

## Работа с массивами : Создание массива

**array**
Создание и инициализация массива.

**Синтаксис :**
array array([mixed ...])

Функция возвращает созданный массив. Индексы и значения в массиве разделяются оператором =>. Пары index=>value разделяются запятыми, они определяют индекс и значение.
Индекс может быть как числовым, так и строковым. В ассоциированных массивах индекс всегда ведет себя как строковой. В случае, если индекс не указан, будет подставляться автоинкремент (на 1 больше), начиная с 0. Если при создании массива были указаны два элемента с одинаковыми индексами, то последний элемент заменяет первый.

```
$arr=array( // Далее мы создадим двумерный массив
"fruit" => array("a"=>"orange", "b"=>"banan", "c"=>"apple"),
// эта запись эквивалентна записи: $arr["fruit"]["a"]="orange"; и т.д.
"number" => array(1,2,3,4,5,6),
// эта запись эквивалентна записи: $arr["number"][]=1; и т.д.
"hotel" => array("first", 5=>"second", "third")
);
$arr=array(1, 1, 1, 1, 2=>5, 19, 3=>20);
print_r($arr);
// Далее распечатка этого массива
Array
(
    [0] => 1
    [1] => 1
    [2] => 5
    [3] => 20
    [4] => 19
)
$arr=array(1 => "Январь", "Февраль", "Март");
print_r($arr);
// распечатка
Array
(
    [1] => Январь
    [2] => Февраль
    [3] => Март
)
```

---

**range**
Заполняет список целыми числами.

**Синтаксис :**

list range(int low, int high)

Функция **range()** создает список, заполненный целыми числами от *low* до *high* включительно. Ее удобно применять, если мы хотим быстро сгенерировать массив для последующего прохождения по нему циклом forech.

```
$arr=range(4,9);
// теперь $arr = array(4, 5, 6, 7, 8, 9);
```

---

## Работа с массивами : Сортировка массивов

**array_reverse**
Расстановка элементов массива в обратном порядке.

**Синтаксис :**
array array_reverse(array arr);

Функция **array_reverse()** возвращает массив, элементы которого следуют в обратном порядке относительно массива, переданного в параметре. При этом связи между ключами и значениями, конечно, не теряются. Например, вместо того, чтобы ранжировать массив в обратном порядке при помощи **arsort()**, мы можем отсортировать его в прямом порядке, а затем перевернуть:

```
$A=array("a"=>"Zero","b"=>"Weapon","c"=>"Alpha","d"=>"Processor");

asort($A);

$A = array_reverse($A);
```

---

**shuffle**
Перемешивание элементов массива.

**Синтаксис:**
void shuffle(array arr);

Функция **shuffle()** "перемешивает" список, переданный ей первым параметром *arr*, так, чтобы его значения распределялись случайным образом. При этом измениться сам массив и ассоциативные массивы воспринимаются как списки.

```
$A = array(10,20,30,40,50);
shuffle($A);
foreach($A as $v) echo "$v ";
// Выведет 10,20,30,40,50 в случайном порядке
```

---

**sort**
Сортировка массива по возрастанию.

**Синтаксис:**
void sort(array arr [, int sort_flags])

Эта функция предназначена для сортировки списков (списки - массивы, ключи которых начинаются с 0 и не имеют пропусков) в порядке возрастания.

```
$A = array("One", "Two", "Tree", "Four");
sort($A);
for($i=0; $i<count($A); $i++) echo "$i:$A[$i] ";
// выводит "0:Four 1:Two 2:Tree 3:One"
```

Любой ассоциативный массив Воспринимается этой функцией как список. То есть после упорядочивания последовательность ключей превращается в 0,1,2,..., а значения нужным образом перераспределяются. Как видим, связи между параметрами *ключ=>значение* не сохраняются, более того - ключи просто

пропадают, поэтому сортировать что-либо, отличное от списка, вряд ли целесообразно.

Аргумент *sort_flags* задает следующие флаги сортировки:
- SORT_REGULAR - сравнивает элементы "как есть"
- SORT_NUMERIC - сравнивает элементы как числа
- SORT_STRING - сравнивает элементы как строки

---

**rsort**
Сортировка массива по убыванию.

**Синтаксис:**

void rsort(array arr [, int sort_flags])

Аналогична функции **sort()**, только сортирует по убыванию.

---

**asort**
Сортировка ассоциативного массива по возрастанию.

**Синтаксис :**
void asort(array arr [, int sort_flags]);

Функция **asort()** сортирует массив, указанный в ее параметре, так, чтобы его значения шли в алфавитном (если это строки) или возрастающем (для чисел) порядке. При этом сохраняются связи между ключами и соответствующими им значениями, т.е. некоторые пары *ключ=>значение* просто "всплывают" наверх, а некоторые - наоборот, "опускаются".

```
$A=array("a"=>"Zero","b"=>"Weapon","c"=>"Alpha","d"=>"Processor");
asort($A);
foreach($A as $k=>$v) echo "$k=>$v ";
// выводит "c=>Alpha d=>Processor b=>Weapon a=>Zero"
// как видим, поменялся только порядок пар ключ=>значение
```

По умолчанию функция **asort()** сортирует массив в алфавитном порядке. Значения флагов сортировки *sort_flags* приведены в описании функции **sort()**.

---

**arsort**
Сортировка ассоциативного массива по убыванию.

**Синтаксис :**
void arsort(array arr [, int sort_flags]);

Эта функция аналогична функции **asort()**, только она упорядочивает массив не по возрастанию, а по убыванию.

```
$arr=array("d"=>"lemon", "a"=>"orange", "b"=>"banana", "c"=>"apple");
arsort($arr);
reset($arr);
while(list ($key, $val) = each ($arr)) {
  echo "$key = $val<BR>l";
}
// выведет:
a = orange
d = lemon
b = banana
c = apple
```

---

**ksort**
Сортировка массива по возрастанию ключей.

**Синтаксис :**
int ksort(array arr [, int sort_flags]);

Функция практически идентична функции **asort()**, с тем различием, что сортировка осуществляется не по значениям, а по ключам (в порядке возрастания).

```
$A=array("d"=>"Zero","c"=>"Weapon","b"=>"Alpha","a"=>"Processor");
ksort($A);
for(Reset($A); list($k,$v)=each($A);) echo "$k=>$v ";
// выводит "a=>Processor b=>Alpha c=>Weapon d=>Zero"
```

Аргумент *sort_flags* указывает параметры сортировки.

---

**krsort**
Сортировка массива по убыванию индексов.

**Синтаксис :**
int krsort(array arr [, int sort_flags]);

Эта функция аналогична функции **ksort()**, только она упорядочивает массив по ключам в обратном порядке.

---

**natsort**
Выполняет "естественную" сортировку массива.

**Синтаксис :**
void natsort(array arr);

Функция **natsort()** сортирует массив в "естественном" для человека порядке.

```
$arr1 = array("html_12.html", "html_10.html", "html_2.html", "html_1.html");
$arr2 = $arr1;
sort($arr1);
echo "Стандартная сортировка:\n";
print_r($arr1);
natsort($arr2);
echo "Естественная сортировка:\n"
print_r($arr2);
```

Этот пример выведет следующее:

```
Стандартная сортировка:
Array
(
    [0] => html_1.html
    [1] => html_10.html
    [2] => html_12.html
    [3] => html_2.html
)
Естественная сортировка:
Array
(
    [3] => html_1.html
    [2] => html_2.html
    [1] => html_10.html
    [0] => html_12.html
)
```

---

**uasort**
Пользовательская сортировка ассоциативного массива.

**Синтаксис:**
void uasort(array arr, function cmp_function)

Функция **uasort()** сортирует массив *arr* с сохранением индексных ассоциаций, используя для сравнения индексов элементов пользовательскую функцию, указанную аргументом *cmp_function*.

---

**uksort**
Пользовательская сортировка массива по ключам.

**Синтаксис:**
void uksort(array arr, function cmp_function)

Функция **uksort()** сортирует массив *arr* по индексам с сохранением индексных ассоциаций, используя для сравнения индексов элементов пользовательскую функцию, указанную в аргументе *cmp_function*. В эту функцию передаются два сравниваемых индекса элементов, а она должна вернуть положительное или отрицательное число или 0.

Довольно часто нам приходится сортировать что-то по более сложному критерию, чем просто по алфавиту. Например, пусть в *$Files* храниться список имен файлов и подкаталогов в текущем каталоге. Возможно, мы захотим вывести этот список не только в лексографическом порядке, но также чтобы все каталоги предшествовали файлам. В этом случае нам стоит воспользоваться функцией **uksort()**, написав предварительно функцию сравнения с двумя параметрами, как того требует **uksort()**.

```
// Эта функция должна сравнивать значения $f1 и $f2 и возвращать:
// -1, если $f1<$f2,
//  0, если $f1==$f2
//  1, если $f1>$f2
// Под < и > понимаем следование этих имен в выводимом списке
function FCmp($f1,$f2)
{ // Каталог всегда предшествует файлу
  if(is_dir($f1) && !is_dir($f2)) return -1;
  // Файл всегда идет после каталога
  if(!is_dir($f1) && is_dir($f2)) return 1;
  // Иначе сравниваем лексиграфически
  if($f1<$f2) return -1; elseif($f1>$f2) return 1; else return 0;
}
// Пусть $Files содержит массив с ключами - именами файлов
// в текущем каталоге. Отсортируем его.
uksort($Files,"FCmp");  //передаем функцию сортировки "по ссылке"
```

---

**usort**
Пользоваетльская сортировка массива.

**Синтаксис:**
void usort(array arr, function cmp_function)

Функция **usort()** сортирует массив *arr* с сохранением индексных ассоциаций, используя для сравнения индексов элементов пользовательскую функцию, указанную в аргументе *cmp_function*. В эту функцию передаются два сравниваемых индекса элементов, а она должна вернуть положительное или отрицательное число или 0.

Эта функция как бы является "гибридом" функций **uasort()** и **sort()**. От **sort()** она отличается тем, что критерий сравнения обеспечивается пользовательской функцией. А от **uasort()** - тем, что она не сохраняет связей между ключами и значениями, а потому пригодна разве что для сортировки списков.

```
function FCmp($a,$b) { return strcmp($a,$b); }
$A = array("One","Two","Three","Four");
```

```
usort($A);
for($i=0; $i<count($A); $i++) echo "$i:$A[$i] ";
// выводит "0:Four 1:One 2:Three 3:Two"
```
Пример одномерного массива:
```
function cmp($a, $b) {
  if($a==$b) return 0;
  return ($a > $b) ? -1 : 1;
}
$a=array (3,2,5,6,1);
usort($a, "cmp");
while(list($key,$val)=each($a)) {
  echo "$key: $val\n";
}
```
При выполнении будет напечатано:
```
0: 6
1: 5
2: 3
3: 2
4: 1
```
Пример многомерного массива:
```
function cmp($a,$b) {
  return strcmp($a["fruit"],$b["fruit"]);
};
$fruit[0]["fruit"]="lemons";
$fruit[1]["fruit"]="apples";
$fruit[2]["fruit"]="grapes";

usort($fruit, "cmp");

while(list($key,$val)=each($fruit)) {
  echo "\$fruit[$key]:".$val["fruit"]."\n";
}
```
При сортировке многомерных массивов $a и $b содержит ссылки на первый индекс массива.
Будет напечатано:
```
$fruit[0]: apples
$fruit[1]: grapes
$fruit[2]: lemons
```

---

**array_multisort**
Сортировка релятивных массивов.

**Синтаксис:**
bool array_multisort(array ar1, [, mixed o1 [, mixed t1 ... [, array ...]]])

Функция **array_multisort()** сортирует многомерные массивы с сохранением индексной ассоциации, возвращая true при отсутствии ошибок.

Исходные массивы рассматриваются как столбцы таблицы, сортируемой построчно. Поэтому массивы должны иметь одинаковое число элементов, и взаимосвязь между ними, как в строках таблицы, сохраняется. Приоритетом сортировки пользуются первые массивы. Флаги сортировки могут указваться для каждого массива, и их действие распространяется только на тот массив, после которого они указаны.

Флаги определения порядка сортировки (аргументы *ox*):
- SORT_ASC - сортировка в порядке возрастания (по умолчанию)
- SORT_DESC - сортировка в порядке убывания Флаги типа сортировки (аргументы *tx*):
- SORT_REGULAR - сравнивать элементы как есть (по умолчанию)

- SORT_NUMERIC - сравнивать элементы как числа
- SORT_STRING - сравнивать элементы как строки

```
ar1 = array("10", 100, 100, "a");
ar2 = array(1, 3, "2", 1);
array_multisort($ar1, $ar2);
  // $ar1 = array("10", "a", 100, 100);
  // $ar2 = array(1, 1, "2", 4);
```

Элементы второго массива, соответствующие одинаковым элементам (100 и 100) первого массива, также отсортированны.

```
$ar = array(array("10", 100, 100, "a"), array(1, 3, "2", 1));
array_multisort($ar[0], SORT_ASC, SORT_STRING,
                  $ar[1], $SORT_NUMERIC, SORT_DESC);
```

$ar[0] = ("10", 100, 100, "a") - сортируются как строки по возрастанию $ar[1] = (1, 3, "2", 1) - сортируются как числа по убыванию

---

## Работа с массивами : Курсор массива

**reset**
Производит сброс курсора массива.

**Синтаксис :**
mixed reset(array arr);

Функция **reset()** устанавливает внутренний курсор массива *arr* на его начало и возвращает значение начального элемента.

---

**end**
Производит перенос курсора в конец массива.

**Синтаксис :**
mixed end(array arr);

Функция **end()** устанавливает внутренний курсор массива *arr* на последний элемент и возвращает значение начального элемента.

---

**next**
Производит перенос курсора вперед.

**Синтаксис :**
mixed next(array arr);

Функция **next()** возвращает значение элемента, на котором в данный момент находится курсор, и перемещает курсор массива на следующий элемент. Возвращает false, если элементов больше не осталось.

Также false возвращается, если встречается элемент с пустым значением, следовательно для корректной работы с массивом, содержащим пустые элементы, лучше использовать функцию **each()**.

---

**prev**
Производит перенос курсора назад.

**Синтаксис :**
mixed prev(array arr);

Функция **prev()** возвращает значение элемента, на котором в данный момент

находится курсор, и перемещает курсор массива на предыдущий элемент. Возвращает false, если элементов больше не осталось.

Также false возвращается, если встречается элемент с пустым значением, следовательно для корректной работы с массивом, содержащим пустые элементы, лучше использовать функцию **each()**.

---

**current**
Определение текущего элемента массива.

**Синтаксис :**
mixed current(array arr);

Функция **current()** возвращает значение элемента, на котором в данный момент находится курсор массива, при этом не сдвигая курсор.

Функция возвратит false, если курсор оказался вне пределов массива, или массив не имеет элементов.

---

**pos**
Определение текущего элемента массива.

**Синтаксис :**
mixed pos(array arr);

Эта функция синоним функции **current()**.

---

**key**
Функция возвращает индекс текущего элемента массива.

**Синтаксис :**
mixed key(array arr);

---

**each**
Получение текущего элемента массива.

**Синтаксис :**
array each(array arr);

Функция **each()** возвращает в массиве пару "индекс и значение" текущего элемента массива, на который указывает внутренний курсор, и сдвигает курсор массива на следующий элемент. Возващаемый массив имеет четыре элемента:

```
[0] => индекс
[1] => "значение"
[key] => индекс
[value] => "значение"
```

Функция возвращает false, если курсор достиг конца массива.

```
$foo = array("bob", "fred", "jussi", "joini", "egon", "marliese");
$bar = each($foo);
// теперь $bar = (0=>0, 1=>"bob", key=>0, value=>"bob")
```

Обычно функцию **each()** используют в паре с **list()** для перебора элементов массива:

```
reset($HTTP_POST_VARS);
while(list($key, $val) = each($HTTP_POST_VARS)) {
  echo "$key = %val<BR>";
}
```

---

**array_walk**
Применение функции к элементам массива.

**Синтаксис :**
int array_walk(array arr string func, mixed userdata);

Функция **array_walk()** применяет пользовательскую функцию *func* к каждому элементу массива *arr*. В пользовательскую функцию передаются три или два (в случае, если аргумент *userdata* не указан) аргумента: значение текущего элемента, его индекс и аргумент *userdata*.

В случае, если *func* требует более трех аргументов, при каждом ее вызове будет выдаваться предупреждение. Чтобы блокировать выдачу этих предупреждений, поставте знак "@" перед функцией **array_walk()** или воспользуйтесь функцией **error_reporting()**.

Функция *func* будет получать значения и индексы массива *arr* по занчению, т.е. не сможет вносить в него изменения. Если это необходимо, передайте аргумент *arr* по ссылке, указав перед его именем "&", и тогда все изменения отразяться в массиве.

В PHP 4 необходимо явно вызывать функцию **reset()**, чтобы установить внутренний курсор на первый элемент.

```
$v = array("d"=>"A1", "a"=>"B2", "b"=>"C3", "c"=>"D4");
function test_alter(&$item1, $key, $prefix) { // по ссылке
  $item1 = "$prefix $item1";
};

function test_print($item2, $key) {
  echo "$key. $item2<BR>";
};

array_walk($v, "test_print");
reset($v);
array_walk($v, "test_alter");
reset($v);
array_walk($v, "test_print");
```

---

## Работа с массивами : Ключи и значения

**array_flip**
Меняет местами индексы и значения массива.

**Синтаксис :**
array array_flip(array arr)

Эта функция "пробегает" по массиву и меняет местами его ключи и значения. Исходный массив arr не изменяется, а результирующий массив просто возвращается. Конечно, если в массиве присутствовало несколько элементов с одинаковыми значениями, учитываться будет только последний из них:

```
$A = array("a"=>"aaa", "b"=>"aaa", "c"=>"ccc");

$A = array_flip($A);

// Теперь $A===array("aaa"=>"b", "ccc"=>"c");
```

---

**array_keys**
Возвращает список из ключей массива.

**Синтаксис :**
list array_keys(array arr [,mixed search_value])

Функция возвращает список, содержащий все ключи массива *arr*. Если задан необязательный параметр *search_value*, то она вернет только те ключи, которым соответствуют значения *search_value*.

```
$arr = array(0 => 100, "color" => "red", 15);
array_keys($arr); // возвратит array(0, "color", 1)

$arr = array("blue", "red", "green", "blue", "blue");
array_keys($arr, "blue"); // возвратит array(0, 3, 4)
```

---

**array_values**
Удаление ассоциативных индексов массива.

**Синтаксис :**
list array_values(array arr)

Функция **array_values()** возвращает список всех значений в ассоциативном массиве *arr*, т.е. превращает ассоциативный массив в простой (скалярный).

```
$arr = array("size" => "XL", "color" => "gold");
array_values($arr);
// возвратит array("XL", "gold")
```

Очевидно, такое действие бесполезно для списков, но иногда оправданно для хешей.

---

**in_array**
Осуществляет проверку массива на наличие значения.

**Синтаксис :**
bool in_array(mixed val, array arr)

Функция **in_array()** возвратит true, если в массиве *arr* содержится элемент со значением *var*.

```
$arr = array("1", "2", "tree");
if(in_array["2", $arr)) echo "2 есть";
```

---

**array_count_values**
Возвращает количество одинаковых значений массива.

**Синтаксис :**
array array_count_values(array arr)

Эта функция подсчитывает, сколько раз каждое значение встречается в массиве *arr*, и возвращает ассоциативный массив с ключами - элементами массива и значениями - количеством повторов этих элементов. Иными словами, функция **array_count_values()** подсчитывает частоту появления значений в массиве *arr*.

```
$LIst = array(1, "hello", 1, "world", "hello");
array_count_values($array);
// возвращает array(1=>2, "hello"=>2, "world"=>1)
```

---

**sizeof**
Возвращает число элементов массива.

**Синтаксис :**

int sizeof(array arr)

Функция **sizeof()** возвращает количество элементов в массиве *arr* на подобие действия функции **count()**.

---

**count**
Возвращает число элементов в массиве или объекте.

**Синтаксис :**
int count(mixed var)

Функция **count()** возвращает число элементов в массиве или объекте *var*. В случае, если *var* - скалярная переменная, то функция возвращает 1, если такая переменная существует, или 0, если такой переменной нет.
Надо отметить, что 0 возвращается и тогда, когда указан массив, не содержащий элементов.
Для проверки существования переменной лучше воспользоваться функцией **isset()**.

```
$a[0] = 1;
$a[1] = 3;
$a[2] = 5;
$result = count($a) // возвратит 3
```

---

**array_sum**
Возвращает сумму всех элементов массива.

**Синтаксис :**
mixed array_sum(array arr [, int num_req])

Эта функция возватит сумму всех числовых элементов массива. От типа значений в массиве зависит тип возвращаемого числа (integer или float).

```
$arr = array(2,4,6,7);
echo "Сумма: ".array_sum($arr);
// выведет Сумма: 19
```

---

**array_rand**
Производит случайную выборку индексов массива.

**Синтаксис :**
mixed array_rand(array arr [, int num_req])

Функция **array_rand()** возвращает в массиве выбранные случайным образом индексы элементов массива *arr*.
Аргумент *num_req* указывает число возвращаемых индексов. В случае, если выбирается один элемент, то возвращается не массив, а значение.

```
srand((double)microtime() *1000000);
// здесь мы проинициализировали генератор случайных чисел
$arr = array("Neo", "Morpheus", "Trinity", "Cypher", "Tank");
$rand_keys = array_rand($arr, 2);
echo $arr[$rand_key[0]]."<BR>";
echo $arr[$rand_key[1]]."<BR>";
```

---

## Работа с массивами : Комплексная замена в строке

**strtr**
Комплексная замена в строке.

**Синтаксис :**
string strtr(string str, string from, string to)
string strtr(string str, array from)

В первом случае функция **strtr()** возвращает строку *str*, у которой каждый символ, присутствующий в строку *from*, заменяется на корреспондирующий из строки *to*. В случае, если строки *from* и *to* различной длины, то лишние конечные символы длинной строки игнорируются.

Во втором случае функция **strtr()** возвращает строку, в которой фрагменты строки *str* заменяются на соответствующие индексам значения элементов массива *from*. При этом функция пытается заменить сначало наибольшие фрагменты исходной строки и не выполняет замену в уже модифицированных частях строки. Таким бразом, теперь мы можем выполнить несколько замен сразу:

```
$Subs = array(
  "<name>" => "Larry",
  "<time>" => date("d.m.Y")
);
$st="Привет, <name>! Сейчас <time";
echo strtr($st,$Subs);
```

А вот как можно "отменить" действие функции **HtmlSpecialChars()**:

```
$Trans=array_flip(get_html_translation_table());
$st=strtr($st, $Trans);
```

В результате мы из строки, в которой все спецсимволы заменены на их HTML-эквиваленты, получим исходную строку.

---

## Работа с массивами : Работа с несколькими массивами

**array_diff**
Определение исключительного пересечения массивов.

**Синтаксис :**
array array_diff(array arr1, array arr2 [, array ...])

Данная функция возвращает массив, который содержит значения, имеющиеся только в массиве *arr1* ( и не имеющиеся в любых других). При этом индексы не сохраняются.

```
$arr1 = array("a" => "green", "red", "blue");
$arr2 = array("b" => "green", "yellow", "red");
$result = array_diff($arr1, $arr2);
// $result = array("blue")
```

---

**array_intersect**
Определение включительного пересечения массивов.

**Синтаксис :**
array array_intersect(array arr1, array arr2 [, array ...])

Функция **array_intersect()** возвращает массив, который содержит значения массива *arr1*, имеющиеся во всех остальных массивах. При этом индексы не сохраняются.

```
$arr1 = array("a" => "green", "red", "blue");
$arr2 = array("b" => "green", "yellow", "red");
$result = array_intersect($arr1, $arr2);
// $result = array("a" => "green", "red")
```

---

**array_merge**
Слияние массивов.

**Синтаксис :**
array array_merge(array arr1, array arr2 [, array ...])

Функция **array_merge()** призвана устранить все недостатки, присущие оператору + для слияния массивов. А именно, она сливает массивы, перечисленные в ее аргументах, в оди большой массив и возвращает результат. Если в массивах встречаются одинаковые ключи, в результат помещается пара *ключ=>значение* из того массива, который расположен правее в списке аргументов. Однако это не затрагивает числовые ключи: элементы с такими ключами помещаются в конец результирующего массива в любом случае.

```
$L1=array(10,20,30);
$L2=array(100,200,300);
$L=array_merge($L1, $L2);
// теперь $L===array(10,20,30,100,200,300);
```

---

**array_merge_recursive**
Объединение сложных массивов.

**Синтаксис :**
array array_merge_recursive(array arr1, array arr2 [, array ...])

Функция **array_merge_recursive()** сильно напоминает функцию **array_merge()** с тем дополнением, что она может работать с многомерными и древовидными массивами, а элементы с одинаковыми строковыми индексами превращаются в подмассивы. Для числовых индексов поведение функции аналогично **array_merge()**.

```
$arr1 = array("color" => array("favorite" =>"red"), 5);
$arr2 = array(10, "color" => array("favorite" =>"green"), "blue");
$result = array_merge_recursive($arr1, $arr2);
// $result = array("color" => array (
//                   "favorite" => array("red", "green"),
//                   "blue"), 5, 10)
```

---

## Работа с массивами : Получение и удаление части массива

**array_slice**
Получение части массива.

**Синтаксис :**
array array_slice(array arr, int offset [, int len])

Эта функция возвращает часть ассоциативного массива *arr*, начиная с элемента со смещением (номером) *offset* от начала и длиной *len* (если последний параметр не задан, до конца массива).

Параметры *offset* и *len* задаются по точно таким же правилам, как и аналогичные параметры в функции **substr()**. А именно, если *offset>0*, то последовательность будет начинаться с элемента, имеющего позицию *offset* от начала массива, а если <0, то отсчет производится от конца массива. Надо отметить, что первый элемент имеет нулевую позицию, а последний (-1).
Если указать *length*>0, то это число возвращаемых в массиве элементов, а если

*length*<0, то это позиция последнего возвращаемого элемента в массиве *arr* от его конца.

```
$input = array("a", "b", "c", "e");
$output = array_slice($input, 2);        // "c", "d", "e"
$output = array_slice($input, 2, -1);    // "c", "d"
$output = array_slice($input, -2, 1);    // "d"
$output = array_slice($input, 0, 3);     // "a", "b", "c"
```

---

**array_splice**
Удаляет часть массива или заменяет ее частью другого массива.

**Синтаксис :**
array array_splice(array arr, int offset [, int len] [, int repl])

Эта функция, также как и **array_slice()**, возвращает подмассив *arr* начиная с индекса *offset* максимальной длины *len*, но, вместе с тем, она делает и другое полезное действие. А именно, она заменяет только что указанные элементы на то, что находится в массиве *repl* (или просто удаляет, если *repl* не указан). Если *offset>0*, то последовательность будет начинаться с элемента, имеющего позицию *offset* от начала массива, а если <0, то отсчет производится от конца массива. Надо отметить, что первый элемент имеет нулевую позицию, а последний (-1). Если указать *length*>0, то это число возвращаемых в массиве элементов, а если *length*<0, то это позиция последнего возвращаемого элемента в массиве *arr* от его конца.

```
$input = array("red", "green", "blue", "yellow");
array_splice($input, 2);
// Теперь $input===array("red", "green")
array_splice($input, 1, -1);
// Теперь $input===array("red", "yellow")
array_splice($input, -1, 1, array("black", "maroon"));
// Теперь $input===array("red", "green", "blue","black",
          "maroon")
array_splice($input, 1, count($input), "orange");
// Теперь $input===array("red", "orange")
```

Последний пример показывает, что в качестве параметра *repl* мы можем указать и обычное, строковое значение, а не массив из одного элемента.

---

## Работа с массивами : Вставка/удаление элементов

**array_pad**
Добавляет в массив несколько элементов.

**Синтаксис :**
array array_pad(array input, int pad_size, mixed pad_value)

Функция **array_pad()** возвращает копию массива *input*, в который были добавлены элементы с значениями *pad_value*, так, что число элементов в получившемся массиве будет равно *pad_size*.
Если *pad_size*>0, то элементы будут добавлены справа, а если <0 - то слева.
В случае, если значение *pad_size*меньше элементов в исходном массиве *input*, то никакого добавления не произойдет, и функция вернет исходный массив *input*.

```
$arr = array(12, 10, 4);
$result = array_pad($arr, 5, 0);
// $result = array(12, 10, 4, 0, 0);
$result = array_pad($arr, -7, -1);
// $result = array(-1, -1, -1, -1, 12, 10, 4)
$result = array_pad($arr, 2, "noop");
// не добавит
```

**array_pop**
Извлекает и удаляет последние элементы массива.

**Синтаксис :**
mixed array_pop(array arr);

Функция **array_pop()** снимает элемент с "вершины" стека (то есть берет последний элемент списка) и возвращает его, удалив после этого его из *arr*. С помощью этой функции мы можем строить конструкции, напоминающие стек. Если массив *arr* был пуст, функция возвращает пустую строку.

```
$stack = array("orange", "apple", "raspberry");
$fruits = array_pop($stack);
// $fruit = "raspberry"
// $stack = array("orange", "apple")
```

**array_push**
Добавляет элементы в конец массива.

**Синтаксис :**
int array_push(array arr, mixed var1 [, mixed var2, ..])

Эта функция добавляет к массиву *arr* элементы *var1,var2* и т.д. Она присваивает им числовые индексы - точно так же, как это происходит для стандартных []. Если вам нужно добавить всего один элемент, наверное, проще будет воспользоваться этим оператором:

```
array_push($Arr,1000);     // вызываем функцию
$Arr[]=100;                // то же самое, но короче
```

Обратите внимание, что функция **array_push()** воспринимает массив, как стек, и добавляет элементы всегда в его конец.

**array_shift**
Извлекает и удаляет первый элемент массива.

**Синтаксис :**
mixed array_shift(array arr)

Эта функция извлекает первый элемент массива *arr* и возвращает его. Она сильно напоминает **array_pop()**, но только получает начальный, а не конечный элемент, а также производит довольно сильную "встряску" всего массива: ведь при извлечении первого элемента приходится корректировать все числовые индексы у всех оставшихся элементов, т.к. все последующие элементы массива сдвигаются на одну позицию вперед.

```
$ar = array("-v", "-f");
$opt = array(_shift($arr);
// теперь $arr = array("-f"), а $opt = "-v"
```

**array_unshift**
Добавляет элементы в начало массива.

**Синтаксис :**
int array_unshift(list arr, mixed var1 [,mixed var2, ...])

Функция очень похожа на **array_push**, но добавляет перечисленные элементы не в конец, а в начало массива. При этом порядок следования *var1, var2* и т.д. остается тем же, т.е. элементы как бы "вдвигаются" в список слева. Новым элементам списка, как обычно, назначаются числовые индексы, начиная с 0; при этом все ключи старых элементов массива, которые также были числовыми,

изменяются (чаще всего они увеличиваются на число вставляемых значений). Функция возвращает новый размер массива.

```
$A = array(10,"a"=>20,30);
array_unshift($A,"!","?");
// теперь $A===array(0=>"!", 1=>"?", 2=>10, a=>20, 3=>30)
```

---

**array_unique**
Создает массив только из уникальных значений.

**Синтаксис :**
array array_unique(array arr)

Функция **array_unique()** возвращает массив, составленный из всех уникальных значений массива *arr* вместе с их ключами, путем удаления всех дублирующих значений. В результирующий массив помещаются первые встретившиеся пары *ключ=>значение*. Индексы сохраняются.

```
$input = array("a" => "green", "red", "b" =>
         "green", "blue", "red");
$result = array_unique($input);
// теперь $result===("a"=>"green", "red", "blue");
```

---

## Работа с массивами : Переменные и массивы

**list**
Заносит элементы массива в переменные.

**Синтаксис :**
**list()** - языковая конструкция (наподобие **array()**). Она присваивает перечисленным переменным значения элементов массива, причем первой переменной присваивается первый элемент массива, второй переменной - второй элемент и т.д.

---

**compact**
Упаковывает в массив переменные из текущего контекста.

**Синтаксис :**
array compact(mixed varname1 [, mixed $varname2, ...])

Функция **compact()** упаковывает в массив переменные из текущего контекста (глобального или контекста функции), заданные своими именами в *varname1, $varname2* и т.д. При этом в массиве образуются пары с ключами, равными содержимому *varnameN*, и значениями соответствующих переменных.
Число аргументов может быть неопределенное.
Если в аргументе указано имя несуществующей переменной, он пропускается.
Действие этой функции противоположно функции **extract()**.

```
$a="Test string";
$b="Some text";
$A=compact("a","b");
// теперь $A===array("a"=>"Test string", "b"=>"Some text")
```

Почему же тогда параметры функции обозначены как mixed? Дело в том, что они могут быть не только строками, но и списками строк. В этом случае функция последнего перебирает все элементы этого списка, и упаковывает те переменные из текущего контекста, имена которых она встретила. Более того - эти списки могут, в свою очередь, также содержать списки строк, и т.д. Правда, последнее используется достаточно редко.

```
$a="Test";
$b="Text";
```

```
$c="CCC";
$d="DDD";
$List=array("b",array("c","d"));
$A=compact("a",$List);
// теперь $A===array("a"=>"Test", "b"=>"Text",
          "c"=>"CCC", "d"=>"DDD")
```

---

**extract**
экспорт элементов массива в переменные.

**Синтаксис :**
void extract(array arr [, int extract_type] [, string prefix])

Эта функция производит действия, прямо противоположные **compact()**. А именно, она получает в параметрах массив *arr* и превращает каждую его пару *ключ=>значение* в переменную текущего контекста.

Параметр *extract_type* предписывает, что делать, если в текущем контексте уже существует переменная с таким же именем, как очередной ключ в *arr*. Он может быть равен одной из констант, перечисленных в следующей таблице:

| Поведение функции **extract** в случае совпадения переменных | |
|---|---|
| EXTR_OVERWRITE | Переписывать существующую переменную. |
| EXTR_SKIP | Не перезаписывать переменную, если она уже существует. |
| EXTR_PREFIX_SAME | В случае совпадения имен создавать переменную с именем, предваренным префиксом из *$prefix*. |
| EXTR_PREFIX_ALL | Всегда предварять имена создаваемых переменных префиксом *$prefix*. |

По умолчанию подразумевается *EXTR_OVERWRITE*, т.е. переменные перезаписываются.

```
// Сделать все переменные окружения глобальными
extract($HTTP_ENV_VARS);
// То же самое, но с префиксом E_
extract($HTTP_ENV_VARS, EXTR_PREFIX_ALL, "E_");
echo $E_COMSPEC;
// Выводит переменную окружения COMSPEC
```

Параметр *prefix* имеет смысл указать только тогда, когда вы применяете режимы *EXTR_PREFIX_SAME* или *EXTR_PREFIX_ALL*.

---

## Строковые функции : Функции для работы с одиночными символами

**chr**
Возвращает один символ с определенным кодом.

**Синтаксис :**
string chr(int ascii)

Возвращает строку из одного символа с кодом *$code*. Эта функция полезна для вставки каких-либо непечатаемых символов в строку - например, кода нуля или символа прогона страницы, а также при работе с бинарными файлами.

```
<?
// Сначало создаем массив того, что мы собираемся выводить,
```

```
// не заботясь о форматировании (дизайне) информации
for($i=0, $x=0; $x<16; $x++) {
  for($y=0; $y<16; $y++) {
    $Chars[$x][$y]=array($i,chr($i));
    $i++;
  }
}
// Теперь выводим накопленную информацию, используя идеологию
// вставки участков кода в HTML-документ
?>

<table border=1 cellpadding=1 cellspacing=0>
<?for($y=0; $y<16; $y++) {?>
  <tr>
  <?for($x=0; $x<16; $x++) {?>
    <td>
      <?=$Chars[$x][$y][0]?>:
      <b><tt><?=$Chars[$x][$y][1]?></tt></b>
    </td>
  <?}?>
  </tr>
<?}?>
</table>
```

---

**ord**
Возвращает ascii код символа.

**Синтаксис :**
int ord(string str)

Эта функция возвращает ASCII код первого символа строки *str*.
Например, ord(chr($n)) всегда равно $n - конечно, если $n заключено между нулем и 255.

---

## Строковые функции : Функции отрезания пробелов

**trim**
Удаляет из заданной строки начальные и конечные пробельные символы.

**Синтаксис :**
string trim(string str)

Возвращает копию *str*, только с удаленными ведущими и концевыми пробельными символами. Под пробельными символами надо понимать "\n", "\r", "\t", "\v", "\0" и пробел.
Например, вызов trim(" test\n ") вернет строку "test".

---

**ltrim**
Удаляет из заданной строки начальные пробельные символы.

**Синтаксис :**
string ltrim(string str)

То же, что и **trim()**, только удаляет исключительно начальные пробельные символы ("\n", "\r", "\t", "\v", "\0" и пробел), а концевые не трогает.

---

**rtrim**
Удаляет из заданной строки конечные пробельные символы.

**Синтаксис :**
string rtrim(string str)

То же, что и **trim()**, только удаляет исключительно конечные пробельные символы ("\n", "\r", "\t", "\v", "\0" и пробел), а начальные не трогает.
Эта функция - синоним **chop()**.

---

**chop**
Удаляет из заданной строки конечные пробельные символы.

**Синтаксис :**
string chop(string str)

Удаляет только концевые пробелы, начальные не трогает.

---

## Строковые функции : Поиск в тексте

**strchr**
Поиск первого вхождения символа в строку.

**Синтаксис :**
string strchr(string haystack, string needle)

Данная функция работает идеентично функции **strstr()**.

---

**strstr**
Поиск первого вхождения подстроки в строку.

**Синтаксис :**
string strstr(string haystack, string needle)

Функция **strstr()** возвращает участок строки, заданной в параметре *haystack*, начиная с первого фрагмента, указанного в параметре *needle* и до конца.
В случае неудачи возвращает false.
Данная функция чувствительна к регистру.
В случае, если *needle* не является строкой, то значение преобразуется в целое и используется как код искомого символа.

```
$email = "mailname@mail.ru";
$domain = strstr($email, "@");
// или
$domain = strstr($email, ord("@"))
echo $domain;
// выведет @mail.ru
```

---

**stristr**
Нахождение первого вхождения подстроки, не учитывая регистр.

**Синтаксис :**
string stristr(string haystack, string needle)

Функция **stristr()** возвращает участок строки, заданной в параметре *haystack*, начиная с первого фрагмента, указанного в параметре *needle* и до конца.
В случае неудачи возвращает false.
Данная функция нечувствительна к регистру.

В случае, если *needle* не является строкой, то значение преобразуется в целое и используется как код искомого символа.

---

**strrchr**
Поиск последнего вхождения подстроки.

**Синтаксис :**
string strrchr(string haystack, string needle)

Функция **strrchr()** возвращает участок строки, заданной в параметре *haystack*, начиная с последнего фрагмента, указанного в параметре *needle* и до конца.
В случае неудачи возвращает false.
Данная функция чувствительна к регистру.
В случае, если *needle* не является строкой, то значение преобразуется в целое и используется как код искомого символа.

```
// получим последний каталог в $PATH
$dir = substr(strrchr($PATH, ":"), 1);
// а здесь получим все после последнего перевода строки
$text = "text 1\nText2\nText3";
echo substr(strrchr($text, 10), 1);
```

---

**strpos**
Находит позицию первого вхождения подстроки в заданной строке.

**Синтаксис :**
int strpos(string where, string what [, int fromwhere])

Функция **strpos()** пытается найти в строке *were* подстроку *what* и в случае успеха возвращает позицию (индекс) этой подстроки в строке. Первый символ строки имеет индекс 0. Необязательный параметр *fromwhere* можно задавать, если поиск нужно вести не с начало строки, а с какой-то другой позиции. В этом случае следует эту позицию передать в *fromwhere*. Если подстроку не удалось найти, функция возвращает false.
Если параметр *what* не строка, в этом случает его значение преобразуется в целое и используется как код искомого символа.

```
if(strpos($text, "a")===false) echo "Не найдено!";
// Проверка: три знака равенства
```

---

**strrpos**
Находит в заданной строке последнюю позицию, в которой находится заданный фрагмент.

**Синтаксис :**
int strrpos(string where, string what)

Данная функция ищет в строке *where* последнюю позицию, в которой встречался символ *what* (если *what* - строка из нескольких символов, то выявляется только первый из них, остальные не играют никакой роли).
Если искомый символ стоит первый в строке или его вообще нет, функция возвратит 0.
В случае, если искомый символ не найден, возвращает false.

---

**substr_count**
Находит количество вхождений фрагмента в строку.

**Синтаксис :**
int substr_count(string where, string what)

Функция **substr_count()** возвращает число фрагментов *what*, присутствующих в строке *where*.

```
echo substr_count("www.spravkaweb.ru", ".");
// Выведет 3
```

---

**strspn**
Определяет присутствие начальных символов в строке.

**Синтаксис :**
int strspn(string str1, string str2)

Функция **strspn()** возвращает длину начального фрагмента строки *str1*, состоящего полностью из символов, которые есть в строке *str2*.

```
echo strspn("www.spravkaweb.ru", "abc");
// Выведет 3
```

---

**strcspn**
Определяет отсутствие начальных символов в строке.

**Синтаксис :**
int strcspn(string str1, string str2)

Функция **strspn()** возвращает длину начального фрагмента строки *str1*, состоящего полностью не из символов, которые есть в строке *str2*.

---

## Строковые функции : Функции сравнения

**strcmp**
Сравнивает строки.

**Синтаксис :**
int strcmp(string str1, string str2)

Эта функция сравнивает две строки посимвольно (точнее, бобайтово) и возвращает:
0 - если строки полностью совпадают;
-1 - если строка *str1* лексикографически меньше *str2*;
1 - если, наоборот, *str1* "больше" *str2*.
Так как сравнение идет побайтово, то регистр символов влияет на результаты сравнений.

---

**strncmp**
Сравнивает начала строк.

**Синтаксис :**
int strncmp(string str1, string str2, int len)

Эта функция отличается от **strcmp()** тем, что сравнивает не все слово целиком, а первые *len* байтов. В случае, если *len* меньше длины наименьшей из строк, то строки сравниваются целиком.
Эта функция сравнивает две строки посимвольно (точнее, бобайтово) и возвращает:
0 - если строки полностью совпадают;
-1 - если строка *str1* лексикографически меньше *str2*;
1 - если, наоборот, *str1* "больше" *str2*.

Так как сравнение идет побайтово, то регистр символов влияет на результаты сравнений.

---

**strcasecmp**
Сравнивает строки без учета регистра.

**Синтаксис :**
int strcasecmp(string str1, string str2)

То же самое, что и **strcmp()**, только при работе не учитывается регистр букв.

```
$str1 = "Привет!";
$str2 = "привет!";
if(!strcesecmp($str1, $str2))
  echo "$str1 == $str2 при сравнении строк без учета регистра";
```

---

**strncasecmp**
Сравнивает начала строк без учета регистра.

**Синтаксис :**
int strncasecmp(string str1, string str2, int len)

Функция **strncasecmp()** является комбинацией функций **strcasecmp()** и **strncmp()**.

---

**strnatcmp**
Производит "естественное" сравнение строк.

**Синтаксис :**
int strnatcmp(string str1, string str2)

Данная функция имитирует сравнение строк, которое использовал бы человек.

```
$arr1 = $arr2 = array("img12.png", "img10.png", "img2.png", "img1.png");
echo "Обычная сортировка\n";
usort($arr1, "strcmp");
print_r($arr1);
echo "\nЕстенственная сортировка\n";
usort($arr2, "strnatcmp");
print_r($arr2);
```

Данный скприпт выведет следующее:

```
Обычная сортировка
Array
(
    [0] => img1.png
    [1] => img10.png
    [2] => img12.png
    [3] => img2.png
)

Естественная сортировка
Array
(
    [0] => img1.png
    [1] => img2.png
    [2] => img10.png
    [3] => img12.png
)
```

---

**strnatcasecmp**
Производит "естественное" сравнение строк без учета регистра.

**Синтаксис :**
int strnatcasecmp(string str1, string str2)

То же, что и **strnatcmp()**, только игнорирует регистр.

---

**similar_text**
Производит определение схожести двух строк.

**Синтаксис :**
int similar_text(string firsrt, string second [, double percent])

Функция **similar_text()** вычисляет схожесть двух строк по алгоритму, описанному Оливером (Oliver [1993]). Но вместо стека (как в псевдокоде Оливера) она использует рекурсивные вызовы.
Сложность алгоритма делает функцию медленной, и ее скорость пропорциональна (N^3), где N - длина наибольшей строки.
Функция возвращает число символов, совпавших в обеих строках. При передаче по ссылке третьего необязательного параметра в нем сохраняется процент совпадения строк.

---

**levenshtein**
Определение различия Левенштейна двух строк.

**Синтаксис :**
int levenshtein(string str1, string str2)
int levenshtein(string str1, string str2, int cost_ins, int cost_rep, int cost_del)
int levenshtein(string str1, string str2, function cost)

"Различие Левенштейна" - это минимальное чило символов, которое требовалось бы заменить, вставить или удалить для того, чтобы превратить строку *str1* в *str2*. Сложность алгоритма пропорциональна произведению длин строк *str1* и *str2*, что делает функцию более быстродействующей, чем *similar_text()*.

Первая форма функции возвращает число необходимых операций над символами строк для трансформации *str1* в *str2*.

Вторая форама имеет три дополнительных параметра: стоимость операции вставки, замены и удаления, что делает ее более адаптированной для вычисления, но при этом менее быстродействующей. Возвращается интегральный показатель сложности трансформации.

Третий вариант позволяет указать функцию, используемую для расчета сложности трансформации. Функция *cost* вызывается со следующими аргументами:
- применяемая операция (вставить, изменить, удалить): "I*quot;, "R", "D";
- фактический символ первой строки
- фактический символ второй строки
- позиция строки 1
- позиция строки 2
- оставшаяся длина строки 1
- оставшаяся длина строки 2

Вызываемая функция должна будет возвратить стоимость этой операции.
Если длина одной из строк более 255 символов, функция **levenshtein()** возвращает -1, но такая длина более чем достаточна.

---

# Строковые функции : Форматирование и вывод строк

**print**
Выводит строку, значение переменной или выражение.

**Синтаксис :**
print(string arg)

Функция **print()** выводит аргумент *arg*, в качестве которого может быть переменная или выражение.

---

**echo**
Производит вывод одного или нескольких значений.

**Синтаксис :**
echo(string arg1, string [argn]...)

Функция **echo()** выводит значения перечисленных параметров.
echo() - фактически языковая конструкция, поэтому для нее не обязательны скобки, даже если используется несколько аргументов.

```
echo "Перенос на следующую строку,
имеющийся в коде, сохраняется
и используется при выводе".
"чтобы избежать этого используйте".
"оператор конкатенации";
```

---

**printf**
Вывод отформатированной строки.

**Синтаксис :**
int printf(string format [, mixed args, ...]);

Делает то же самое, что и **sprintf()**, только результирующая строка не возвращается, а направляется в браузер пользователя.

---

**sprintf**
Производит форматирование строки с подстановкой переменных.

**Синтаксис :** sprintf($format [,args, ...])
Эта функция возвращает строку, составленную на основе строки форматирования, содержащей некоторые специальные символы, которые будут впоследствии заменены на значения соответствующих переменных из списка аргументов.

Строка форматирования *$format* может включать в себя команды форматирования, предваренные символом %. Все остальные символы копируются в выходную строку как есть. Каждый спецификатор формата (то есть, символ % и следующие за ним команды) соответствуют одному, и только одному параметру, указанному после параметра *$format*. Если же нужно поместить в текст % как обычный символ, необходимо его удвоить:

```
echo sprintf("The percentage was %d%%",$percentage);
```

Каждый спецификатор формата включает максимум пять элементов (в порядке их следования после символа %):
- Необязательный спецификатор размера поля, который указывает, сколько символов будет отведено под выводимую величину. В качестве символов-

заполнителей (если значение имеет меньший размер, чем размер поля для его вывода) может использоваться пробел или 0, по умолчанию подставляется пробел. Можно задать любой другой символ-наполнитьел, если указать его в строке форматирования, предварив фпострофом.
- Опциональный спецификатор выравнивания, определяющий, будет результат выровнен по правому или по левому краю поля. По умолчанию производится выравнивание по правому краю, однако можно указать и левое выравнивание, задав символ - (минус).
- Необязательное число, определяющее размер поля для вывода величины. Если результат не будет в поле помещаться, то он &quo;вылезет" за края этого поля, но не будет усечен.
- Необязательное число, предваренное точкой ".", предписывающее, сколько знаков после запятой будет в результирующей строке. Этот спецификатор учитывается только в том случае, если происходит вывод числа с плавающей точкой, в противном случае он игнорируется.
- Наконец, обязательный (заметьте - единственный обязательный!) спецификатор типа величины, которая будет помещена в выходную строку:
    - b - очередной аргумент из списка выводится как двоичное целое число
    - c - выводится символ с указанным в аргументе кодом
    - d - целое число
    - f - число с плавающей точкой
    - o - восьмеричное целое число
    - s - строка символов
    - x - шестнадцатиричное целое число с маленькими буквами a-z
    - X - шестнадцатиричное целое число с большими буквами A-Z

Вот как можно указать точность представления чисел с плавающей точкой:
```
$money1 = 68.75;
$money2 = 54.35;
$money = $money1 + $money2;
// echo $money выведет "123.1"...
$formatted = sprintf ("%01.2f", $money);
// echo $formatted выведет "123.10"!
```
Вот пример вывода целого числа, предваренного нужным количеством нулей:

```
$isodate=sprintf("%04d-%02d-%02d",$year,$month,$day);
```

---

**sscanf**
Производит интерпретацию строки согласно формату и занесение значений в переменные.

**Синтаксис :**
mixed sscanf(string str, string format [, string var1 ...])

Функция **sscanf()** является противоположностью функции **printf()**. Она интерпретирует строку *str* согласно формату *format*, аналогично спецификации **printf()**. При указании только двух аргументов полученные значения возращаются в массиве.
```
// получение серийного номера
$serial = sscanf("SN/235-0001", "SN/%3d-%4d");
echo $serial[0]*10000+$serial[1]; // выводит: 2350001
// и даты изготовления
$date = "January 01 2000";
list($month, $day, $year) = sscanf($date, "%s %d %d");
echo "Дата: $year-".substr($month,0,3)."-$day\n";
// выводит: 2000-Jan-01
```

При указании дополнительных необязательных параметров (их следует передавать по ссылке) функция возвращает их число. Те переменные, которые не получают значений, в возвращаемом значении не учитываются.

```
// гененируем XML запись из строки
$auth = "765\tLewis Carroll";
$n = sscan($auth,"%d\t%s %s", &$id, &$first, &$last);
echo "<author id="$id">
  <firstname>$first</firrstname>
  <surname>$last</surname>
</author>\n";
```

---

## Строковые функции : Составление/разбиение строк

**substr**
Возвращает участок строки с определенной длиной.

**Синтаксис :**
string substr(string str, int start [,int length])

Возвращает участок строки *str*, начиная с позиции *start* и длиной *length*. Если *length* не задана, то подразумевается подстрока от *start* до конца строки *str*. Если *start* больше, чем длина строки, или же значение *length* равно нулю, то возвращается пустая подстрока.

Однако эта функция может делать и еще довольно полезные вещи. К примеру, если мы передадим в *start* отрицательное число, то будет считаться, что это число является индексом подстроки, но только отсчитываемым от конца *str* (например, -1 означает "начинается с последнего символа строки").
Параметр *length*, если он задан, тоже может быть отрицательным. В этом случае последним символом возвращаемой подстроки будет символ из *str* с индексом *length*, определяемым от конца строки.

```
$str = substr("abcdef", 1);     // возвратит "bcdef"
$str = substr("abcdef", 1, 3);  // возвратит "bcd"
$str = substr("abcdef", -1);    // возвратит "f"
$str = substr("abcdef", -2);    // возвратит "ef"
$str = substr("abcdef", -3, 1); // возвратит "d"
$str = substr("abcdef", 1, -1); // возвратит "bcde"
```

---

**str_repeat**
Повторяет строку определенное количество раз.

**Синтаксис :**
string str_repeat(string str, int number)

Функция "повторяет" строку *str number* раз и возвращает объединенный результат.

```
echo str_repeat("test!",3); // выводит test!test!test!
```

---

**str_pad**
Дополняет строку другой строкой до определенной длины.

**Синтаксис :**
string str_pad(string input, int pad_length [, string pad_string [, int pad_type]])

Аргумент *input* задает исходную строку. Аргумент *pad_length* задает длину возвращаемой строки. Если он имеет значение меньше, чем исходная строка, то никакого добавления не производится.

При помощи необязательного аргумента *pad_string* можно указать, какую строку использовать в качестве заполнителя (по умолчанию - пробелы).
При помощи необязательного аргумента *pad_type* можно указать, с какой стороны следует дополнять строку: справо, слево или с обеих сторон.
Этот аргумент может принимать следующие значения:

- STR_PAD_RIGHT (по умолчанию)
- STR_PAD_LEFT
- STR_PAD_BOTH

```
$str = "Aaaaa";
echo str_pad($str, 10);
// возвратит "Aaaaa"
echo str_pad($str, 10, "-=", STR_PAD_LEFT);
// возвратит "-=-=-Aaaaa"
echo str_pad($str, 10, "_", STR_PAD_BOTH)
// возвратит "_Aaaa_"
```

---

**chunk_split**
Возвращает фрагмент строки.

**Синтаксис :**
string chunk_split(string str [, int chunklen [, string end]])

Функция **chunk_split()** возвращает строку, в которой между каждым блоком строки *str* длиной *chunklen* (по умолчанию 76) вставляется последовательность разделителей *end* (по умолчанию: "\r\n").
Данная функция может быть полезна при конвертировании в формат "base64" для соответствия правилам RFC 2045.

```
// отформатируем $data, используя семантику RFC 2045
$str = chunk_split(base64_encode($data));
```

Эта функция работает значительно быстрее, чем **ereg_replace()**.

---

**strtok**
Возвращает строку по частям.

**Синтаксис :**
string strtok(string arg1, string arg2)

Фунция возвращает часть строки *arg1* до разделителя *arg2*. При последующих вызовах возвращается следующая часть до следующего разделителя, и так до конца строки. При первом вызове функция принимате два аргумента: исходную строку *arg1* и разделитель *arg2*. При каждом последующем вызове аргумент *arg1* указываеть не надо, иначе будет возвращаться первая часть строки. Когда возвращать больше нечего, функция вернет false. Если часть строки состоит из 0 или из пустой строки, то функция также вернет false.

```
$str="This is an example№string№ Aaa";
$tok = strtok($str, " ");
while($tok) {
  echo "$tok";
  $tok = strtok(" №");
};
// выведет: "This" "is" "an" "example" "string"
```

Надо заметить, что в качестве разделителей указывается последовательность символов, каждый из которых в отдельности может являться разделителем, но когда в строке последовательно встречаются два или более разделителей, функция возвращает пустую строку (что может прекратить цикл обработки, как в примере).

---

**explode**
Производит разделение строки в массив.

**Синтаксис :**
array explode(string seperator, string str [, int limit])

Функция **explode()** возвращает массив строк, каждая из которых соответствует фрагменту исходной строки *str*, находящемуся между разделителями, указанными аргументом *separator*.
Необязательный параметр *limit* указывает максимальное количество элементов в массиве. Оставшаяся неразделенная часть будет содержаться в последнем элементе.

```
$str = "Path1 Path2 Path3  Path4";
$str_exp = explode(" ", $str);
// теперь $str_exp = array([0] => Path1, [1] => Path2,
//        [2] => Path3, [3] => '', [4] => Path4)
```

---

**implode**
Производит объединение массива в строку.

**Синтаксис :**
string implode(string glue, array pieces)

Функция **implode()** возвращает строку, которая содержит последовательно все элементы массива, заданного в параметре *pieces*, между которыми вставляется значение, указанное в параметре *glue*.

```
$str = implode(":", $arr);
```

---

**join**
Производит объединение массива в строку.

**Синтаксис :**
string join(string glue, array pieces)

То же, что и **implode()**.

---

## Строковые функции : Работа с блоками текста

**str_replace**
Заменяет в исходной строке одни подстроки на другие.

**Синтаксис :**
string str_replace(string from, string to, string str)

Эта функция заменяет в строке *str* все вхождения подстроки *from* (с учетом регистра) на *to* и возвращает результат. Исходная строка, переданная третьим параметром, при этом не меняется.
также эта функция может работать с двоичными строками.

---

**substr_replace**
Заменяет в исходной строке одни подстроки на другие.

**Синтаксис :**
string substr_replace(string str, string replacement, int start [, int lemgth])

Эта функция возвращает строку *str*, в которой часть от символа с позицией *start* и

длиной *length* (или до конца, если аргумент длины не указан) заменяется строкой *replacement*.

Если значение *start* положительно, отсчет производится от начала строки *str*, иначе - от конца (-1 - последний символ строки).

Если значение *length* неотрицательно, тогда оно указывает длину заменяемого фрагмента. Если оно отрицательно, то это число символов от конца строки *str* до последнего символа заменяемого фрагмента (со знаком минус).

---

**wordwrap**
Разбивает исходный текст на строки с определенными завершающими символами.

**Синтаксис :**
string wordwrap(string str [, int width [, string break [, int cut]]])

Эта функция разбивает блок текста *str* на несколько строк, завершаемых символами *break*, так, чтобы на одной строке было не более *width* букв. Разбиение происходит по границе слова, так что текст остается читаемым.

---

**strtr**
Комплексная замена в строке.

**Синтаксис :**
string strtr(string str, string from, string to)
string strtr(string str, array from)

В первом случае функция **strtr()** возвращает строку *str*, у которой каждый символ, присутствующий в строку *from*, заменяется на корреспондирующий из строки *to*. В случае, если строки *from* и *to* различной длины, то лишние конечные символы длинной строки игнорируются.

Во втором случае функция **strtr()** возвращает строку, в которой фрагменты строки *str* заменяются на соответствующие индексам значения элементов массива *from*. При этом функция пытается заменить сначало наибольшие фрагменты исходной строки и не выполняет замену в уже модифицированных частях строки. Таким бразом, теперь мы можем выполнить несколько замен сразу:

```
$Subs = array(
  "<name>" => "Larry",
  "<time>" => date("d.m.Y")
);
$st="Привет, <name>! Сейчас <time";
echo strtr($st,$Subs);
```

А вот как можно "отменить" действие функции **HtmlSpecialChars()**:

```
$Trans=array_flip(get_html_translation_table());
$st=strtr($st, $Trans);
```

В результате мы из строки, в которой все спецсимволы заменены на их HTML-эквиваленты, получим исходную строку.

---

**stripslashes**
Удаление обратных слешей.

**Синтаксис :**
string stripslashes(string str);

Заменяет в строке *str* некоторые предваренные слешем символы на их однокодовые эквиваленты. Это относится к следующим символам: ", ", \.

**stripcslashes**
Преобразование специальных символов в их двоичное представление.

**Синтаксис :**
string stripcslashes(string str);

Возвращает строку, в которой те специальные символы, которые закомментированы (для визуального отображения) обратным слешем, преобразуются в их естественное двоичное представление. Распознаются C-подобные записи, например: \n, \r ..., восьмеричные и шестнадцатеричные последовательности.

---

**addslashes**
Добавление слешей перед специальными символами строки.

**Синтаксис :**
string addslashes(string str);

Вставляет слеши только перед следующими символами:", " и \. Функцию очень удобно использовать при вызове **eval()**.

---

**addcslashes**
Форматирование строки слешами в C-представление.

**Синтаксис :**
string addcslashes(string str, string charlist);

Функция **addcslashes()** возвращает строку *str*, в которую вставлены символы обратного слеша "\" перед перечисленными в строке-списке *charlist* символами. Это позволяет преобразовать непечатные символы в их визуальное C-представление.

---

**quotemeta**
Цитирование метасимволов.

**Синтаксис :**
string quotemeta(string str);

Возвращает строку, в которую добавлены обратные слеши "\" перед каждым из следующих символов:
. \\ + * ? [ ^ ] ( $ )

Может использоваться для подготовки шаблонов в регулярных выражениях.

---

**strrev**
Производит реверс строки.

**Синтаксис :**
string strrev(string str)

функция **strrev()** возвращает строку *str* "задом наперед".

---

## Строковые функции : Функции для преобразования символов

**nl2br**
Заменяет символы перевода строки.

**Синтаксис :**
string nl2br(string string)

Заменяет в строке все символы новой строки *n* на <*br*>*n* и возвращает результат. Исходная строка не изменяется. Обратите внимание на то, что символы *r*, которые присутствуют в конце строки текстовых файлов Windows, этой функцией никак не учитываются, а потому остаются на старом месте.

---

**strip_tags**
Удаляет из строки теги.

**Синтаксис :**
string strip_tags(string str [, string allowable_tags])

Эта функция удаляет из строки все HTML- и PHP-теги и возвращает результат. Незавершенные или фиктивные теги вызывают ошибку.
В параметре *allowable_tags* можно передать теги, которые не следует удалять из строки. Они должны перечисляться вплотную друг к другу.

```
$st="
<b>Жирный текст</b>
<tt>Моноширный текст</tt>
<a href=http://spravkaweb.ru>Ссылка</a>";
echo "Исходный текст: $st";
echo "<hr>После удаления тегов: ".striptags($st,"<a><b>").
     "<hr>";
```

Запустив этот пример, мы сможем заметить, что теги <a> и <b> не были удалены (ровно как и их парные закрывающие), в то время как <tt> исчез.

---

**get_meta_tags**
Функция ищет и обрабатывает все теги <META>.

**Синтаксис :**
array get_meta_tags(string filename, int use_include_path)

Функция открывает файл и ищет в нем все теги <META> до тех пор, пока не встретится закрывающий тег </head>.
Если очередной тег <META> имеет вид:
<meta name="название" content="содержимое">
то пара *название*=>*содержимое* добавляется в результирующий массив, который под конец и возвращается.
Спецсимволы в значении атрибута *filename* заменяются на знак подчеркивания "_", а алфавитные символы преобразуются в нижний регистр.
Функция удобно использовать для быстрого получения всех метатегов из указанного файла.
Если необязательный параметр *use_include_path* установлен, то поиск файла осуществляется не только в текщем каталоге, но и во всех тех, которые назначены для поиска инструкциями **include** и **require**.

---

**get_html_translation_table**
Функция возвращает таблицу трансляции, которая используется функциями htmlspecialchars() и htmlentities().

**Синтаксис :**
string get_html_translation_table(int table [, int quote_style])

В этой функции аргумент *table* указывает, какую таблицу трансляции необходимо получить: HTML_SPECIALCHARS для функции **htmlspecialchars()** или HTML_ENTITIES для функции **htmlentities()**. Описание необязательного параметра *quote_style* приведено в функции **htmlspecialchars()**.

```
$trans = get_html_translation_table(HTML_ENTITIES);
$str = "<A & B>";
$encoded = strtr($str, $strans);
// $encoded = "&amplt; A &amp; B &gt;"
```

Иногда удобно использовать функцию **array_flip()** для изменения направления транслитерации.

```
$trans = array_flip($trans);
$original = strtr($encoded, $trans);
```

---

**htmlspecialchars**
Производит преобразование спецсимволов в HTML-представление.

**Синтаксис :**
string HtmlSpecialChars(string str [, int quote_style]);

Основное назначение этой функции - гарантировать, что в выводимой строке ни один участок не будет воспринят как тэг.
Заменяет в строке некоторые символы (такие как амперсанд, кавычки и знаки "больше" и "меньше") на их HTML-эквиваленты,так, чтобы они выглядели на странице "самими собой". Самое типичное применение этой функции - формирование параметра *value* в различных элементах формы, чтобы не было никаких проблем с кавычками, или же вывод сообщения в гостевой книге, если вставлять теги пользователю запрещено. При помощи необязательного атрибута *quote_style* можно указать, что делать с кавычками:

- ENT_COMPAT (по умолчанию) - разрешить трансляцию только двойных кавычек
- ENT_QUOTES - разрешить трансляцию любых кавычек
- ENT_NOQUOTES - запретить трансляцию любых кавычек

```
$str = htmlspecialchars("<a href=index.php>Главная</a>", ENT_QUOTES);
```

---

**htmlentities**
Производит конвертацию символов, имеющих HTML-представление.

**Синтаксис :**
string htmlentities(string str [, int quote_style]);

Эта функция похожа на **htmlspecialchars()**, но только в ней производиться не выборочная трансляция, а полная - для всех символов, которые могут иметь эквивалентные HTML-представления.
При помощи необязательного атрибута *quote_style* можно указать, что делать с кавычками:

- ENT_COMPAT (по умолчанию) - разрешить трансляцию только двойных кавычек
- ENT_QUOTES - разрешить трансляцию любых кавычек
- ENT_NOQUOTES - запретить трансляцию любых кавычек

---

**hebrev**
Конвертация логического текста Hebrew в отображаемый.

**Синтаксис :**
string hebrev(string hebrew_text [, int max_chars_per_line]);

Необязательный аргумент *max_chars_per_line* указывает число символов на строку вывода. Функция пытается избежать разрыва слова.

---

**hebrevc**
Аналог функции hebrev с расстоновкой переносов.

**Синтаксис :**
string hebrevc(string hebrew_text [, int max_chars_per_line]);

Функция **hebrevc()** сходна с **hebrev()** с тем отличием, что она преобразует символы перевода строк "\n" в "<br>\n". необязательный аргумент *max_chars_per_line* указывает число символов на строку вывода. Функция пытается избежать разрыва слов.

---

**quoted_printable_decode**
Преобразование цитированной строки в 8-битную.

**Синтаксис :**
string quoted_printable_decode(string str);

---

## Строковые функции : Функции изменения регистра

**strtolower**
Производит преобразование символов строки в нижний регистр.

**Синтаксис :**
string strtolower(string str);

Преобразует строку в нижний регистр. Возвращает результат перевода.
Надо заметить, что при неправильной настройке локали функция будет выдавать, мягко говоря, странные результаты при работе с буквами кириллицы.

```
$str = "HeLLo World";
$str = strtolower($str);
echo $str;
// выведет hello world
```

---

**strtoupper**
Производит преобразование заданной строки в верхний регистр.

**Синтаксис :**
string strtoupper(string str);

Переводит строку в верхний регистр. Возвращает результат преобразования. Эта функция хорошо работает с английскими буквами, но с русскими может чудить.

```
$str = "Hello World";
$str = strtoupper($str);
echo $str;
// выведет HELLO WORLD
```

---

**ucfirst**
Производит преобразование первого символа строки в верхний регистр.

**Синтаксис :**
string ucfirst(string str);

Возвращает строку, у которой первый символ заглавный.
Символы кириллицы могут быть неправильно конвертированы.

```
$str = "hello world";
$str = ucfirst($str);
echo $str;
// выведет Hello world
```

---

**ucwords**
Производит преобразование первого символа каждого слова строки в верхний регистр.

**Синтаксис :**
string ucwords(string str);

Возвращает строку, у которой первый символ каждого слова в строке заглавный. Под словом здесь понимается участок строки, которому предшествует пробельный символ: пробел, переход на новую строку, прогонка страницы, возврат каретки, горизонтальная и вертикальная табуляция.
Символы кириллицы могут быть неправильно конвертированы.

```
$str = "hello world";
$str = ucfirst($str);
echo $str;
// выведет Hello World
```

---

## Строковые функции : Установка локали (локальных настроек)

**setlocale**
Установка региональных настроек.

**Синтаксис :**
string SetLocale(string category, string locale);

Функция **setlocale** устанавливает текущую локаль, с которой будут работать функции преобразования регистра символов, вывода даты-времени и т.д. Вообще говоря, для каждой категории функций локаль определяется отдельно и выглядит по-разному. То, какую именно категорию функций затронет вызов **setlocale()**, задается в параметре *category*. Он может принимать следующие строкрвые значения:

- **LC_CTYPE** - активизирует указанную локаль для функций перевода в верхний/нижний регистры;
- **LC_NUMERIC** - активизирует локаль для функций форматирования дробных чисел - а именно, задает разделитель целой и дробной части в числах;
- **LC_TIME** - задает формат вывода даты и времени по умолчанию;
- **LC_ALL** - устанавливает все вышеперечисленные режимы.

Теперь поговорим о параметре *locale*. Как известно, каждая локаль, установленная в системе, имеет свое уникальное имя, по которому к ней можно обратиться. Именно оно и фиксируется в этом параметре. Однако, есть два важных исключения из этого правила. Во-первых, если величина *locale* равна пустой строке "", то устанавливается та локаль, которая указана в глобальной переменной окружения с именем, совпадающем с именем категории *category* (или LANG - она практически всегда присутствует в Unix). Во-вторых, если в этом параметре передается 0, то новая локаль не устанавливается, а просто возвращается имя текущей локали для указанного режима.

```
setlocale("LC_CTYPE","ru_SU.KOI*-R");
// Здесь вызов устанавливает таблицу замены
// регистра букв в соответствии с кодировкой KOI8-R.
```

---

## Строковые функции : Преобразование кодировок

**convert_cyr_string**
Преобразует строку из одной кодировки кириллицы в другую.

**Синтаксис :**
string convert_cyr_string(string str, string from, string to);

Функция переводит строку *str* из кодировки *from* в кодировку *to*. Конечно, это имеет смысл только для строк, содержащих "русские" буквы, т.к. латиница во всех кодировках выглядит одинаково. Разумеется, кодировка *from* должна совпадать с истинной кодировкой строки, иначе результат получится неверным. Значения *from* и *to* - одиночный символ, определяющий кодировку:

- k - koi8-r
- w - windows-1251
- i - iso8859-5
- a - x-cp866
- d - x-cp866
- m - x-mac-cyrillic

Функция работает достаточно быстро, так что ее вполне можно применять для перекодировки писем в нужную форму перед их отправкой по электронной почте.

---

**bin2hex**
Производит преобразование символьных данных в шестнадцатеричный вид.

**Синтаксис :**
string bin2hex(string str)

Функция **bin2hex()** возвращает стороковое шестнадцатеричное представление символьно-байтовых данных, содержащихся в строке *str*. Конвертация производиться побайтово, старший полубайт указывается первым.

---

## Строковые функции : Функции форматных преобразований

**parse_url**
Обрабатывает URL и возвращает его компоненты.

**Синтаксис :**
array parse_url(string url);

Эта функция возвращает ассоциативный массив, включающий множество различных существующих компонентов URL. Они включают "scheme", "host", "port", "user", "pass", "path", "query" и "fragment".

---

**parse_str**
Заносит строки URL в переменные.

**Синтаксис :**
void parse_str(string str [, array arr]);

Функция **parse_str()** интерпретирует строку *str*, как если бы эта строка содержала в себе переменные и их значения и передавалась бы в URL. Наша функция устанавливает для этих переменных значения.
Если задан второй необязательный параметр, то значения, найденные при помощи функции **parse_str()**, сохраняться не в глобальных переменных, а в элементах указанного массива.

```
$str =
"name[]=Vasia&name[]=Pupkin&id=12645&mail=vasia@mail.ru&url=www.vasia.ru";
parse_str($str);
parse_str($str, $arr);
echo $id;        // выведет 1264
echo $name[0];   // выведет Vasia
echo $name[1];   // выведет Pupkin
print_r($arr);
// выведет
Array
(
    [name] => Array
        (
            [0] => Vasia
            [1] => Pupkin
        )

    [id] => 12645
    [mail] => vasia@mail.ru
    [url] => www.vasia.ru
)
```

---

**rawurlencode**
Кодирование URL.

**Синтаксис :**
string RawUrlEncode(string str);

Функция **RawUrlEncode()** возвращает строку, в которой все не алфавитно-цифровые символы (за исключением дефиса "-", знака подчеркивания "_" и точки ".") заменены последовательностями: знак процента (%), за которым следуют две шестнадцатеричные цифры, которые обозначают код символа. Это кодирование нужно для того, чтобы буквенные символы не обрабатывались в качестве разделителей строки URL и не искажались при передаче в сетях.

```
echo "<A href=ftp://user:".rawurlencode($mypasswd).
      "@ftp.my.com/x.txt>"; // передача пароля в гиперссылке
```

---

**rawurldecode**
Производит декодирование URL.

**Синтаксис :**
string rawurldecode(string str);

Эта функция возвращает строку, в которой последовательности с знаком процента (%) и следующими за ним двумя шестнадцатеричных цисла преобразует в симолы, соответствующими этому коду. Аналогична **urldecode()**, но не воспринимает + как пробел.

```
$str="foo%20bar%40baz";
echo rawurldecode($str);
// выведет foo bar@baz
```

---

**base64_encode**
Кодирует данные в кодировке MIME base64.

**Синтаксис :**
string base64_encode(string data);

base64_encode() возвращает data закодированные в кодировке base64. Эта кодировка разработана для того, чтобы передовать двоичные данные через транспортные слои, которые не содержат восьмой бит, такие как почтовые тела. Данные в кодировке Base64 занимают примерно на 33% больше места, чем оригинал.

---

**base64_decode**
Декодирует данные, закодированные в кодировке MIME base64.

**Синтаксис :**
string base64_decode(string encoded_data);

base64_decode() декодирует encoded_data и возвращает оригинал данных. Возвращаемые данные могут быть двоичными.

---

## Строковые функции : Функции URL

**number_format**
Форматирование числа.

**Синтаксис :**
number_format($number, $decimals, $dec_point=".", $thousands_sep=",");

Эта функция форматирует число с плавающей точкой с разделением его на триады с указанной точностью. Она может быть вызвана с двумя или четырьмя аргументами, но не с тремя! Параметр *$decimals* задает, сколько цифр после запятой должно быть у числа в выходной строке. Параметр *$dec_point* представляет собой разделитель целой и дробной частей, а параметр *$thousands_sep* - разделитель триад в числе (если указать на его месте пустую строку, то триады не отделяются друг от друга).

---

## Строковые функции : Работа с бинарными данными

**pack**
Пакетирование данных в двоичную строку.

**Синтаксис :**
string pack(string format [,mixed $args, ...]);

Функция **pack()** упаковывает заданные аргументы в бинарную строку, которая затем и возвращается. Формат параметров, а также их количество, задается при помощи строки *$format*, которая представляет собой набор однобуквенных спецификаторов форматирования - наподобие тех, которые указываются в **sprintf()**, но только без знака %. После каждого спецификатора может стоять число, которое отмечает, сколько информации будет обработано данным спецификатором. А именно, для форматов a,A,h и H число задает, какое количество символов будет помещено в бинарную строку из тех, что находится в очередном параметре-строке при вызове функции (то есть, определяет размер

поля для вывода строки). В случае @ оно определяет абсолютную позицию, в которую будут помещены следующие данные. Для всех остальных спецификаторов следующие за ними числа задают количество аргументов, на которые распространяется действие данного формата. Вместо числа можно указать *, в этом случае подразумевается, что спецификатор действует на все оставшиеся данные.

Вот полный список спецификаторов формата:

- a - строка, свободные места в поле заполняются символом с кодом 0;
- A - строка, свободные места заполняются пробелами;
- h - шестнадцатиричная строка, младшие разряды в начале;
- H - шестнадцатиричная строка, старшие разряды в начале;
- c - знаковый байт (символ);
- C - беззнаковый байт;
- s - знаковое короткое целое (16 битовб порядок байтов определяется архитектурой процессора);
- S - беззнаковое короткое число;
- n - беззнаковое целое (16 битов, старшие разряды в конце);
- v - беззнаковое целое (16 битов, младшие разряды в конце);
- i - знаковое целое (размер и порядок байтов определяется архитектурой);
- I - беззнаковое целое;
- l - знаковое длинное целое (32 бита, порядок знаков определяется архитектурой);
- L - беззнаковое длинное целое;
- N - беззнаковое длинное целое (32 бита, старшие разряды в конце);
- V - беззнаковое целое (32 бита, младшие разряды в конце);
- f - число с плавающей точкой (зависит от архитектуры);
- d - число с плавающей точкой двойной точности (зависит от архитектуры);
- x - символ с нулевым кодом;
- X - возврат назад на 1 байт;
- @ - заполнение нулевым кодом до заданной абсолютной позиции.

```
// Целое, целое, все остальное – сиволы
$bindata = pack("nvc*", 0x1234, 0x5678, 65, 66);
```

После выполнения приведенного кода в строке *$bindata* будет содержаться 6 байтов в такой последовательности:
0x12, 0x34, 0x78, 0x56, 0x41, 0x42 (в шестнадцатиричной системе счисления).

---

**unpack**
Распаковывает данные из двоичной строки.

**Синтаксис :**
array unpack(string format, string data);

Распаковывает данные из двоичной строки в массив согласно формату.
Возвращает массив, содержащий распакованные элементы.

```
$array = unpack("c2chars/nint", $binarydata);
```

Возникающий в результате массив будет содержать "chars1", "chars2" и "int".

---

## Строковые функции : Строковые суммы и хеш-функции

**strlen**
Возвращает длину строки.

**Синтаксис :**

int strlen(string str)

Возвращает просто длину строки, т.е., сколько символов содержится в *str*. Строка может содержать любые символы, в том числе и с нулевым кодом. Функция **strlen()** будет правильно работать и с такими строками.

---

**count_chars**
Возвращает информацию о символах строки.

**Синтаксис :**
mixed count_chars(string str [, int mode])

Функция **count_chars()** подсчитывает частоту встречаемости каждого байта (0-255) в строке *str* и возвращает в массиве результат согласно необязательному аргументу *mode*. *mode* может принимать следующие значения:
- 0 (по умолчанию)- массив с байтами в качестве индексов и частотой повторения в качестве значений элемента массива
- 1 - похож на 0, но отсутствующие в строке *str* байты не возвращаются
- 2 - похож на 0, но возвращаются только те байты, которые отсутствуют
- 3 - возвращается строка, состоящая из всех обнаруженных символов
- 4 - возвращается строка, состоящая из всех отсутствующих символов

---

**md5**
Получение строки-хеша MD5.

**Синтаксис :**
string md5(string str);

Возвращает хеш-код строки *str*, основанный на алгоритме корпорации RSA Data Security под названием "MD5 Message-Digest Algorithm". Хеш-код - это просто строка, практически уникальная для каждой из строк *str*. То есть вероятность того, что две разные строки, переданные в *str*, дадут нам одинаковый хеш-код, стремиться к нулю.
В то же время, если длина строки *str* может достигать нескольких тысяч символов, то ее MD5-код занимает максимум 32 символа.

---

**crc32**
Получение полиминала строки crc32.

**Синтаксис :**
int crc32(string str);

Функция **crc32()** вычисляет 32-битную контрольную сумму строки *str*. То есть, результат ее работы - 32-битное (4-байтовое) целое число.
Обычно эту функцию используют для проверки целостности переданных данных. Эта функция работает гораздо быстрее **md5()**, но в то же время выдает гораздо менее надежные "хеш-коды" для строки. Так что, теперь, чтобы получить методом случайного подбора для двух разных строк одинаковые "хеш-коды", вам потребуется не триллион лет работы самого мощного компьютера, а всег лишь год-другой.

---

**crypt**
Производит симметричное шифрование.

**Синтаксис :**
string crypt(string str [,string salt]);

В аргументе *str* задается строка, которую надо зашифровать.
Хеш-код для одной и той же строки, но с различными значениями *salt* (Кстати, это должна быть обязательно двухсимвольная строка) дает разные результаты. Если параметр *salt* пропущен, PHP сгенерирует его случайным образом.
В системах, которые поддерживают несколько алгоритмов шифрования, следующие константы устанавливаются равными 1 или 0, в зависимости от того, поддерживается ли данный алгоритм или нет:

- CRYPT_STD_DES - стандартное 2-байтовое DES-шифрование (SALT=2)
- CRYPT_EXT_DES - расширенное 9-байтовое DES-шифрование (SALT=9)
- CRYPT_MD5 - 12-байтовое MD5-шифрование (SALT начинается с $1$)
- CRYPT_BLOWFISH - расширенное 12-байтовое DES-шифрование (SALT начинается с $2$)

Т.к. данная функция использует односторонний алгоритм шифрования, то функции дешифрования не имеется.

---

**metaphone**
Производит вычисление метафон-хеша.

**Синтаксис :**
string metaphone(string str);

Данная функция схожа по действию с **soundex()**, вычисляет код произношения слова, переданного в строке *str*, но с повышенной точностью вычисления, т.к. использует правила произношения английского языка.
Возвращаемое строковое значение может быть переменной длины.

---

**soundex**
Вычисления хеша сходности произношения.

**Синтаксис :**
string soundex(string str);

Функция **soundex()** используется для проверки правописания, когда приблизительно известно как звучит слово, но не известно, как оно пишется, и имеется словарь (база данных), относительно которого можно осуществить проверку.
Возвращается строка из 4 символов: первая буква слова и 3 цифры.

```
soundex("Euler") == soundex("Ellery") == "E460";
soundex("Gauss") == soundex("Ghosh") == "G200";
soundex("Hilbert") == soundex("Heilbronn") == "H416";
soundex("Knuth") == soundex("Kant") == "K530";
soundex("Lloyd") == soundex("Ladd") == "L300";
soundex("Lukasiewicz") == soundex("Lissajous") == "L222";
```

---

## Символические ссылки. Жесткие ссылки

### Немного теории

В системах Unix довольно часто возникает необходимость иметь для одного и того же файла или каталога разные имена. При этом одно из имен логично назвать

основным, а все другие - его псевдонимами. В терминологии Unix такие всевдонимы называются *символическими ссылками*.

Символическая ссылка - это просто бинарный файл специального вида, который содержит ссылку на основной файл. При обращении к такому файлу (например, открытию его на чтение) система "соображает", к какому объекту на самом деле запрашивается доступ, и прозрачно его обеспечивает. Это означает, что мы можем использовать символические ссылки точно так же, как и обычные файлы. Однака иногда нужно бывает работать со ссылкой именно как со ссылкой, а не как с файлом. Для этого и существуют перечисленные ниже функции PHP.

---

**Жесткие ссылки**

Создание символической ссылки - не единственный способ задать для одного файла несколько имен. Главный недостаток символических ссылок - существование основного имени файла, на которое все и ссылаются. Попробуйте удалить этот файл - и вся "паутина" ссылок, если таковая имелась, развалиться на куски. Есть и другой недостаток: открытие файла, на который указывает ссылка, происходит несколько медленнее, т.к. системе нужно проанализировать содержимое ссылки и установить связь с "настоящим" файлом. Особенно это чувствуется, если одна ссылка указывает на другую, а та на третью и т.д. уровней на 10.

*Жесткие ссылки* позволяют вам иметь для одного файла несколько совершенно равноправных имен, причем доступ по ним осуществляется одинаково быстро. При этом, если одно из таких имен будет удалено, то сам файл удалиться только в том случае, если данное имя было последним, и других имен у файла нет.

Зарегестрировать новое имя у файла (то есть создать для него жесткую ссылку) можно с помощью функции **link()**. Ее синтаксис полностью идеентичен функции **symlink()**, да и работает она по тем же правилам, за исключением того, что создает не символическую, а жесткую ссылку.

---

**readlink**
Возвращает имя основного файла.

**Синтаксис :**
string readlink(string $linkname)

Возвращает имя основного файла, с которым связан его синоним *$linkname*. Это бывает полезно, если вы хотите узнать основное имя файла, чтобы, например, удалить сам файл, а не ссылку на него. В случае ошибки функция возвращает значение "ложь".

---

**symlink**
Создает символическую ссылку.

**Синтаксис :**
bool symlink(string $target, string $link)

Эта функция создает символическую ссылку с именем *$link* на объект (файл или каталог), заданную в *$target*. В случае "провала" функция возвращает false.

---

**lstat**
Функция собирает вместе всю информацию, выдаваемую операционной системой для указанной ссылки, и возвращает ее в виде массива.

**Синтаксис :**
array lstat(string $filename)

Функция полностью аналогична вызову **stat()**, за исключением того, что если *$filename* задает не файл, а символическую ссылку, будет возвращена информация именно об этой ссылке (а не о файле, на который она указывает, как это делает **stat()**).

---

**linkinfo**
Функция возвращает значение поля "устройство" из результата, выдаваемого функцией **lstat()**.

**Синтаксис :**
int linkinfo(string $linkname)

Ее обычно задействуют, если хотят определить, существует ли еще объект, на который указывает символическая ссылка в *$linkname*.

---

## Функции даты и времени

**checkdate**
Проверяет правильность даты/времени.

**Синтаксис :**
int checkdate(int month, int day, int year);

Функция **checkdate()** проверяет правильность даты, заданной в ее аргументах. Возвращает true если дата, указанная как "month, day, year" (месяц, число, год), правильна, иначе false. Дата считается правильной, если:
- год между 1 и 32767 включительно
- месяц между 1 и 12 включительно
- день находится в диапазоне разрешенных дней данного месяца. Високосные годы учитываются.

```
$month=1;
$day=10;
$year=2002;
if(checkdate($month,$day,$year)) echo "Такой день есть!";
else echo "Такого дня нет!";
```

Выведет: Такой день есть!

```
$month=13;
$day=10;
$year=2002;
if(checkdate($month,$day,$year)) echo "Такой день есть!";
else echo "Такого дня нет!";
```

Выведет: Такого дня нет!

---

**date**
Формат локального времени/даты.

**Синтаксис :**
string date(string format [, int timestamp]);

Эта функция возвращает строку, содержащую дату и время, отформатированную согласно строке *format* и используя временную метку *timestamp* или текущее локальное время, если не задана временная метка.
В фоматной строке должны использоваться следующие символы:

- a - "до" и "после" полудня: "am" или "pm"
- A - "До" и "После" полудня: "AM" или "PM"
- d - день месяца, 2 цифры (на первом месте ноль) (от 01 до 31)
- D - день недели, текстовый, 3 буквы; т.е. "Fri"
- j - день месяца, 1-2 цифры без начальных нулей (от 1 до 31)
- F - месяц, текстовый, длинный; т.е. "January"
- h - час, 12-часовой формат (от 01 до 12)
- H - час, 24-часовой формат (от 00 до 23)
- g - час, 12-часовой формат без нулей (от 1 до 12)
- G - час, 24-часовой формат без нулей (от 0 до 23)
- i - минуты (от 00 до 59)
- I(большая i) - 1, если действует переход на летнее время, иначе 0
- L - 0, если год не високосный, или 1 в противном случае
- B - Swatch Internet time
- T - временная зона компьютера, например: MDT (доступна не всегда)
- l (строчная "L") - день недели, текстовый, длинный; т.е. "Friday"
- m - месяц, две цифры с нулями (от 01 до 12)
- n - месяц, одна-две цифры без нулей (от 1 до 12)
- M - трехбуквенное английское сокращение месяца; т.е. "Jan"
- t - число дней в указанном месяце (от 28 до 31)
- s - секунды (от 0 до 59)
- S - англоязычный порядковый суффикс числа из двух букв, текстовый, т.е. "th", "nd"
- U - целое число секунд, прошедших с момента начала эпохи UNIX (доступно не всегда)
- Y - год, цифровой, 4 цифры (1999)
- y - год, цифровой, 2 цифры (99)
- w - порядковое число дня в неделе,(от 0-воскресенье до 6-суббота)
- z - порядковое число дня в году (от 0 до 365)
- Z - смешение временной зоны в секунадах (от -43200 до 43200)

Все остальные символы в строковом аргументе format возвращаются в результирующей строке "как есть".
Формат "Z" всегда возвращает 0 при использовании с функцией gmdate().

```
echo date("Сегодня d.m.Y");
//Сегодня 31.01.2002
echo date( "l dS of F Y h:i:s A" );
// Thursday 31st of January 2002 12:51:19 PM
echo "July 1, 2000 is on a " . date("l", mktime(0,0,0,7,1,2000));
// July 1, 2000 is on a Saturday
```

Функции date() и mktime() возможно использовать вместе для того, чтобы найти даты в будущем или прошлом.

```
$tomorrow  = mktime(0,0,0,date("m")  ,date("d")+1,date("Y"));
$lastmonth = mktime(0,0,0,date("m")-1,date("d"),  date("Y"));
$nextyear  = mktime(0,0,0,date("m"),  date("d"),   date("Y")+1);
```

---

**localtime**
Получает информацию о дате/времени.

**Синтаксис :**
array localtime([int timestamp [, bool is_associative]]);

Первый необязательный аргумент этой функции задает метку времени Unix. В случае, если он не указан, то используется текущее время.

Если второй необязательный параметр равен нулю (по умолчанию), то возвращаемый массив будет численно индексирован; в противном случае возвращается ассоциативный массив, где элементы имеют следующие значения:

- ([1])"tm_sec" - секунды
- ([2])"tm_min" - минуты
- ([3])"tm_hours" - часы
- ([4])"tm_mday" - день месяца
- ([5])"tm_mon" - месяц в году
- ([6])"tm_year" - год, цифровой
- ([7])"tm_wday" - день недели
- ([8])"tm_yday" - день в году
- ([9])"tm_isdst" - активен ли переход на летнее время

---

**gettimeofday**
Получени даты системным вызовом.

**Синтаксис :**
array gettimeofday();

Эта функция возвращает ассоциативный массив, который содержит дату, возвращенную системным вызовом. Функция является интерфейсом системной функции gettimeofday(2).
Возвращаемый ассоциативный массив содержит следующие элементы:
- "sec" - секунды
- "usec" - микросекунды
- "minuteswest" - смещение к западу от Гринвича, в минутах
- "dsttime" - тип dst коррекции (переход на летнее время)

---

**strftime**
Форматирует время согласно локальным установкам.

**Синтаксис :**
string strftime(string format [, int timestamp]);

Возвращает строку, отформатированную согласно данной форматной строке *format* и используя данную временную метку *timestamp* или текущее локальное время, если метка не задана.
Функцией **setlocale()** можно выставить язык, на котором будут выводиться названия месяцев и дней.
В форматной строке следует использовать следующие спецификаторы преобразований:

- %a - сокращенное название дня недели по умолчанию (Wed);
- %A - полное название дня недели по умолчанию (Wednesday);
- %b - сокращенное название месяца по умолчанию (Apr);
- %B - полное название месяца по умолчанию (April);
- %c - предпочтительное представление даты и времени (06/19/02 15:45:11);
- %C - номер века (год, деленный на 100 и без дробной части, от 00 до 99);
- %d - день месяца как десятичное число (в диапазоне от 0 до 31);
- %D - эквивалент %m/%d/%y;
- %e - число месяца (вместо незначащего нуля ставится пробел) (от 1 до 31);
- %h - аналог %b;
- %H - час как десятичное число в 24-часовом формате (в диапазоне от 00 до 23);
- %I - час как десятичное число в 12-часовом формате (в диапазоне от 01 до 12);
- %j - номер дня в году как десятичное число (в диапазоне от 001 до 366);

- %m - номер месяца как десятичное число (в диапазоне от 1 до 12);
- %M - минуты как десятичное число;
- %n - символ перехода на новую строку;
- %p - "am" или "pm" (до и после полудня) согласно текущему времени;
- %r - время в 12-часовом формате (a.m. или p.m.);
- %R - время в 24-часовом формате;
- %S - секунды как десятичное число;
- %t - символ табуляции;
- %T - текущее время, эквивалентно %H:%M:$S;
- %u - число дня в неделе (от 1 до 7) (понедельник - 1);
- %U - номер недели в году как десятичное число, начиная с первого Воскресенья в качестве первого дня первой недели;
- %V - номер недели в году по стандарту ISO 8601:1988 (от 1 до 53), где первая неделя - та, в которой насчитывается больше 3-х дней в текущем году;
- %W - номер недели в году как десятичное число, начиная с первого понедельника в качестве первого дня первой недели;
- %w - номер дня в неделе (от 0 до 6) (воскресенье - 0);
- %x - представление даты в системном формате без указания времени (06/13/02);
- %X - представление времени в системном формате без указания даты (15:34:54);
- %y - год как десятичное число без столетия (в диапазоне от 00 до 99);
- %Y - год как десятичное число, включая столетие;
- %Z - временная зона или название или сокращение;
- %% - символ "%".

```
setlocale ("LC_TIME", "C");
print(strftime("%A in Finnish is "));
setlocale ("LC_TIME", "fi");
print(strftime("%A, in French "));
setlocale ("LC_TIME", "fr");
print(strftime("%A and in German "));
setlocale ("LC_TIME", "de");
print(strftime("A.\n"));
```

Форматирует локальное время согласно установкам locale.

---

**getdate**
Получает информацию о дате/времени.

**Синтаксис :**
array getdate(int timestamp);

Возвращает ассоциативный массив, содержащий информацию о дате со следующими элементами:
- "seconds" - секунды
- "minutes" - минуты
- "hours" - часы
- "mday" - день месяца
- "wday" - день недели, цифровой
- "mon" - месяц, цифровой
- "year" - год, цифровой
- "yday" - день года, цифровой; т.е. "299"
- "weekday" - день недели, текстовый, полный; т.е. "Friday"
- "month" - месяц, текстовый, полный; т.е. "January"
- 0 - "UNIX timestamp", полученный аргумент.

```
print_r(getdate(time()));
```

Приведенный пример выведет следующее:

```
Array
(
    [seconds] => 23
    [minutes] => 44
    [hours] => 22
    [mday] => 15
    [wday] => 0
    [mon] => 8
    [year] => 2004
    [yday] => 227
    [weekday] => Sunday
    [month] => August
    [0] => 1092595463
)
```

---

**gmdate**
Получение даты в форматированной строке для времени GMT.

**Синтаксис :**
string gmdate(string format, int timestamp);

Аналогична функции date() за исключением того, что время возвращается в Гринвичском формате Greenwich Mean Time (GMT). Например, при запуске в Финляндии (GMT +0200), первая строка ниже напечатает "Jan 01 1998 00:00:00", в то время как вторая строка напечатает "Dec 31 1997 22:00:00".

```
echo date( "M d Y H:i:s",mktime(0,0,0,1,1,1998) );
echo gmdate( "M d Y H:i:s",mktime(0,0,0,1,1,1998) );
```

---

**gmstrftime**
Форматирование локальных времени/даты.

**Синтаксис :**
string gmstrftime(string format, int timestamp);

Действие этой функции аналогично действию функции strftime() с тем отличием, что возвращается время по Гринвичу (GMT).
Например, при запуске в зоне (GMT -0500) первая строка бедт следующей: "Dec 31 1998 20:00:00", а вторая: " Jan 01 1999 01:00:00".

```
setlocale("LC_TIME", "en_US");
echo strftime("%b %d %Y %H:%M:%S", mktime(20,0,0,12,31,98))."\n";
echo gmstrftime("%b %d %Y %H:%M:%S", mktime(20,0,0,12,31,98))."\n";
```

---

**mktime**
Получает временную метку UNIX для даты.

**Синтаксис :**
int mktime([int hour] [,int minute] [,int second] [,int month] [,int day] [,int year] [, int is_dst]);

Возвращает временную метку Unix согласно данным аргументам. Эта временная метка является целым числом, равным количеству секунд между эпохой Unix (1 Января 1970) и указанным временем.

Все парамеры это функции необязательны, но пропускать их можно только справо налево. Если какие-то параметры не заданы, на их место подставляются значения, соответствующие текущей дате.
Аргумент *is_dst*, указывает, осуществлен ли переход на летнее время (1) или нет (0); если не известно, то аргумент - (-1)

Функция возвращает *timestamp*, соответствующий указанной дате.

Правильность даты, переданной в параметрах, не проверяется. В случае некорректной даты ничего особенного не происходит - функция "делает вид", что ее не касается, и формирует соответствующий *timestamp*.

```
echo date( "M-d-Y", mktime(0,0,0,12,32,1997) ); // правильная дата
echo date( "M-d-Y", mktime(0,0,0,13,1,1997) );  // неправильная дата
echo date( "M-d-Y", mktime(0,0,0,1,1,1998) );   // неправильная дата
```

Выводит три одинаковых числа

---

**gmmktime**
Аналог функции time() для времени GMT.

**Синтаксис :**
int gmmktime(int hour, int minute, int second, int month, int day, int year [, int is_dst]);

Идентична mktime() за исключением того, что передаваемые параметры передают время по Гринвичу (GMT).

---

**time**
Получение времени в секунадх.

**Синтаксис :**
int time();

Возвращает текущее время, измеренное в числе секунд с эпохи Unix (1 Января 1970 00:00:00 GMT).
Этот формат данных принят в Unix как стандартный (называется "UNIX timestamp"): в частности, время последнего изменения файлов указывается именно в таком формате. Вообще говоря, почти все функции по работе со временеи имеют дело именно с таким его представлением (которое называется *timestamp*). То есть представление " количество секунд с 1 января 1970 года" весьма универсально и, что главное, - удобно.

```
echo time();
```

---

**microtime**
Возвращает текущую временную метку UNIX в микросекундах.

**Синтаксис :**
string microtime();

Возвращает строку "msec sec" где sec текущее время, измеренное в числе секунд с эпохи Unix (0:00:00 1 Января, 1970 GMT), а msec - это часть в микросекундах. Эти функции доступны только в операционных системах, поддерживающих системный вызов gettimeofday().
Но дело в том, что милисекунды в различных ОС выглядят по разному. Например в Unix это действительно число микросекунд, а в Windows - непонятное значение.

```
echo microtime(); // в Windows выведет что-то типа 0.53033200 1012468870
```

---

**strtotime**
Лексическое преобразование строки времени в Unix timestamp.

**Синтаксис :**
int strtotime(string time [, int now]);

В аргументе *time* функция получает дату в англоязычном формате, а затем преобразует ее в целочисленный формат Unix timestamp.

```
echo strtotime("now")."\n";
echo strtotime("10 September 2002")."\n";
echo strtotime("+2 day")."\n";
echo strtotime("+3 week")."\n";
echo strtotime("+1 week 2 days 4 hours 34 seconds")."\n";
```

---

## Логические функции определения типа переменной

**is_scalar**
Проверяет, является ли переменная простой.

**Синтаксис :**
bool is_scalar(mixed var)

Возвращает true, если *var* имеет скалярный тип (чила, строки, логические значения), но не комплексный (массивы или объекты).

---

**is_null**
Проверяет, является ли переменная пустой.

**Синтаксис :**
bool is_null(mixed var)

Возвращает true, если *var* не существует (или ей явно присвоено пустое значение null). Функция эквивалентна выражению:
(var===null или is_set(var)

---

**is_numeric**
Проверяет, является ли переменная числовой.

**Синтаксис :**
bool is_numeric(mixed var)

Возвращает true, если *var* имеет численный тип (integer, double), или строковой с числовым значением.

---

**is_bool**
Проверяет, является ли переменная логической.

**Синтаксис :**
bool is_bool(mixed var)

Возвращает true, если *var* имеет тип логического значения (TRUE или FALSE), а иначе - false.

---

**is_int**
Определяет, является ли переменная типа integer(целочисленной).

**Синтаксис :**
bool is_int(mixed var);

Возвращает true, если var имеет целочисленный тип.

---

**is_integer**
Определяет, является ли переменная типа integer.

**Синтаксис :**
bool is_integer(mixed var);

Возвращает true, если var является типа integer (целочисленной), или false в обратном случае.

---

**is_long**
Определяет, является ли переменная типа integer(целочисленной).

**Синтаксис :**
bool is_long(mixed var);

Возвращает true, если var имеет целочисленнй тип (integer, long), или false в обратном случае.

---

**is_real**
Определяет, является ли переменная типа real (дробной).

**Синтаксис :**
bool is_real(mixed var);

Возвращает true, если var типа real (дробная), или false в обратном случае.

---

**is_float**
Определяет, является ли переменная типа float (дробная).

**Синтаксис :**
bool is_float(mixed var);

Возвращает true, если var типа float (дробная), или false в обратном случае.

---

**is_double**
Определяет, является ли переменная типа double (дробная).

**Синтаксис :**
bool is_double(mixed var);

Возвращает true, если var является типа double (дробной), или false в обратном случае.

---

**is_string**
Определяет, является ли переменная строкой.

**Синтаксис :**
bool is_string(mixed var);

Возвращает true, если var - это строка, или false в обратном случае.

---

**is_array**
Определяет, является ли переменная массивом.

**Синтаксис :**
bool is_array(mixed var);

Возвращает true, если var - это массив, или false в обратном случае.

---

**is_object**
Определяет, является ли переменная объектом.

**Синтаксис :**
bool is_object(mixed var);

Возвращает true, если var - это объект (object), или false в обратном случае.

---

**is_resource**
Определяет, является ли переменная указателем на ресурс.

**Синтаксис :**
bool is_resource(mixed var);

Возвращает true, если var указывает на ресурс, выделенный и возвращенный предназначенной для этого функцией.

Ресурсы - это объекты, подобные файлам или результатам запросов к базам данных, выделяемые и освобождаемые внутренними функциями PHP. Когда программе больше не требуется какой-либо ресурс, хорошей практикой программирования является его явное освобождение предназначенными для этого функциями. Но в большинстве случаев интерпретатор PHP самостоятельно, по мере необходимости освобождает ненужные ресурсы (обычно при завершении сценария).

---

**get_resource_type**
Определение типа дескриптора ресурса.

**Синтаксис :**
string get_resource_type(resource $handle);

Эта функция возвращает строку, содержащую описание типа ресурса. Если в аргументе передается неверный указатель на ресурс, то возникает ошибка.

```
$c = mysql_connect();
echo get_resource_type($c)."\n";
// выведет: mysql link

$file = fopen("filename.txt","w");
echo get_resource_type($file)."\n";
// выведет: file

$doc = new_xmldoc("1.0");
echo get_resource_type($doc)."\n";
// выведет: domxml document
```

---

## Функции переменных

**gettype**
Получает тип переменной.

**Синтаксис :**
string gettype(mixed var);
Возвращает тип переменной PHP var.

Возможные значения для возвращаемой строки:
"integer"
"double"
"string"
"array"
"object"
"unknown type"

---

**intval**
Возвращает целочисленное значение переменной.

**Синтаксис :**
int intval(mixed var, int [base]);

Возвращает целочисленное значение переменной var, используя указанное основание для перевода (по умолчанию 10). var может быть скалярного типа. Вы не можете использовать функцию intval() для массивов или объектов.

---

**doubleval**
Получает значение переменной в формате double.

**Синтаксис :**
double doubleval(mixed var);

Возвращает double (с плавающей точкой) значение переменной var.
var может быть скалярного типа. Вы не можете использовать doubleval() на массивах и объектах.

---

**empty**
Определяет, есть ли у переменной какое либо значение.

**Синтаксис :**
int empty(mixed var);

Возвращает false, если var существует и имеет не пустое или не нулевое значение; true в обратном случае.
Фактически функция проверяет, имеет ли переменнная значение, которое может быть приравнено к 0, то есть: (var==0)

```
$var=0;
if(empty($var)) {
  echo "$var - либо 0, либо не имеет значения";
  if(!isset($var)) {
    echo "$var не имеет значения";
  };
};
```

Заметьте, что эту функцию бесполезно использовать с аргументом, отличным от переменной, например выражение empty (Addslashes($name)) бессмысленно, т.к. здесь проверяется значение, возвращаемое функцией. Единственное, что может выявить в данном варианте функция **empty()**, это имеет ли выражение значение, эквивалентное true (не равное нулю), а это может быть проверено и без использования функции.

---

**isset**
Определяет, существует ли переменная.

**Синтаксис :**
int isset(mixed var);

Возвращает true, если var существует; false в обратном случае.
Фактически функция проверяет, имеет ли переменная значение, идентичное null, то есть: (var===null). Отметьте различие между равенством и идентичностью. Если переменная была удалена функцией unset(), она больше не будет определяться функцией isset().

```
$a = "test";
echo isset($a); // true
unset($a);
echo isset($a); // false
```

---

**settype**
Устанавливает тип переменной.

**Синтаксис :**
int settype(string var, string type);

Устанавливает тип переменнойvar на type.
Возможные значения type :
"integer"
"double""
"string"
"array""
"object"
Возвращает true, при успехе; false в обратном случае.

---

**strval**
Получает строковое значение переменной.

**Синтаксис :**
string strval(mixed var);

Получает строковое значение var.
var может быть любого скалярного типа. Вы не можете использовать strval() на массивах или объектах.

---

**unset**
Удаляет указанную переменную.

**Синтаксис :**
int unset(mixed var);

unset() уничтожает указанную переменную и возвращает true.
**Пример :**
unset( $foo );
unset( $bar["quux"] );

---

## Функции для работы с функциями

**get_defined_functions**
Возвращает перечень всех доступных функций.

**Синтаксис:**
array get_defined_functions()

Функция **get_defined_functions()** Возвращает многомерный массив, который содержит имена всех доступных сценарию функций.

```
print_r(get_defined_functions);
```

---

**function_exists**
Проверяет существование функции.

**Синтаксис:**
bool function_exists(string fuction_name)

Функция **function_exists()** возвращает значение true, если функция с именем *function_name* имеется в сценарии. В противном случае возвращает false.

```
if(function_exists("imagecreate")) {
  echo "Библиотека работы с графикой доступна!
";
} else {
  echo "Библиотека работы с графикой недоступна!
";
};
```

---

**call_user_func**
Производит косвенный выбор функции.

**Синтаксис:**
mixed call_user_func(string function_name [, mixed parameter [, mixed ...]])

Функция **call_user_func()** вызывает функцию *function_name* и передает ей все остальные параметры *parameter ...*

```
<?php
function myFunc($str) {
  echo $str;
};
call_user_func("myFunc", "Hello World");
?>
```

---

**create_function**
Динамическое создание функции.

**Синтаксис :**
string create_function(string args, string code)

Функция **create_function()** создает анонимную функцию и возвращает имя, созданное для этой функции. Аргументы функции, перечисленные в аргументе *args*, обычно передаются в одинарных кавычках. Также передается тело функции в аргументе *code*. Это необходимо для того, чтобы недопустить замену интерпретатором переменных на значения. Если всеже ограничивать двойными кавычками, то необходимо предварять указание переменных слешем :\$var.

Обычно возвращаемые функцией имена содержат префикс *lambda_*.

При помощи этой функции можно создавать функции на основе информации, полученной во время исполнения сценария.

```
$func = create_function('$a,$b',
          'return "$a*$b = ".$a*$b);');
echo "Имя новой функции: $func<br>";
echo $func(2,3);
```

Данный пример выведет следующее:

```
Имя новой функции: lambda_1
6
```

**func_get_arg**
Получение аргумента функции.

**Синтаксис :**
mixed func_get_arg(int arg_num);

Функция **func_get_arg()** возвращает указанный в *arg_num* агрумен, который был передан в текущую пользовательскую функцию в качестве параметра. Перечисление переданных в функцию параметров начинается с нуля.

В случае, если эта функция вызывается вне определения функции, то она выдает предупреждение об ошибке. Также предупреждение будет выдаваться при попытке узнать параметр, которого не существует в списке аргументов (функция возвратит false). Для того, чтобы функция работала корректно, необходимо заранее узнать общее количество параметров, которое передается в пользовательскую функцию с помощью функции **func_num_args()**.

```
<?php
function func() {
  $num_args=func_num_args();
  echo "Число аргументов у этой функции: $num_args<br>";
  for($i=0;$i<$num_args;$i++)
    echo "$i агрумент: ".func_get_arg($i)."<br>";
};

func("2",1,"tree");
?>
```

**func_get_args**
Получение аргументов функции в массиве.

**Синтаксис :**
array func_get_args();

Функция **func_get_args()** возвращает в массиве список аргументов, с которым была вызвана текущая пользовательская функция. Если функция **func_get_args()** вызывается вне определения пользовательской функции, то выдается предупреждение об ошибке.

```
<?php
function func() {
  $num_args = func_num_args();
  echo "Число аргументов у этой функции: $num_args<br>";
  $func_list = func_get_args();
  for($i=0;$i<$num_args;$i++)
    echo "$i агрумент номер $i: ".$func_list[$i]."<br>";
};

func("2",1,"tree");
?>
```

**func_num_args**
Возвращает число полученных аргументов в пользовательской функции.

**Синтаксис :**
int func_num_args();

Функция **func_num_args()** возвращает число аргументов, которые были переданы в текущую пользовательскую функцию.

Обычно эту функцию используют совместно с функциями **func_get_arg()** и **func_get_args()** в пользовательских функциях, которые могут принимать неопределенное количество параметров.

```
<?php
function func() {
  $num_args = func_num_args();
  echo "Число аргументов у этой функции: $num_args<br>";
  $func_list = func_get_args();
  for($i=0;$i<$num_args;$i++)
    echo "$i агрумент номер $i: ".$func_list[$i]."<br>";
};

func("2",1,"tree");
?>
```

---

## Календарные функции

**JDToGregorian**
Преобразование дневного Юлианского счета в Грегорианскую дату.

**Синтаксис :**
string jdtogregorian(int julianday);

Преобразование дневного Юлианского счета в Грегорианскую в формате "месяц/день/год"

---

**GregorianToJD**
Преобразовывает Грегорианскую дату на Дневный Юлианский счет.

**Синтаксис :**
int gregoriantojd(int month, int day, int year);

Правильный дипазон для Грегорианского календаря 4714 д.э. до 9999 н.э.

Хотя это программное обеспечение может оперировать даты в обратном порядке до 4714 д.э., такое использование может быть бесполезно и не значительно. Грегорианский календарь не был учрежден до Октября 15, 1582 (или Октябрь 5, 1582 в Юлианском календаре). Некоторые страны еще очень долго не принимали его. Например, Великобритания преобразовалась в 1752, СССР в 1918 и Греции в 1923. Наиболее Европейские страны использовали Юлианский Календарь До Грегорианского.

**Пример :**

```
<?php
$jd = GregorianToJD(10,11,1970);
echo("$jd\n");
$gregorian = JDToGregorian($jd);
echo("$gregorian\n");
?>
```

---

**JDToJulian**
Преобразовывает дату Юлианского календаря на дневный Юлианский счет.

**Синтаксис :**
string jdtojulian(int julianday);

Преобразование дневного Юлианского счета в строку, содержащую дату Юлианского Календаря в формате "месяц/день/год".

**JulianToJD**
Преобразовывает дату Юлианского Календаря на дневной Юлианский счет.

**Синтаксис :**
int juliantojd(int month, int day, int year);

Правильный дипазон для Юлианского календаря 4713 д.э. до 9999 н.э.

Хотя это программное обеспечение может оперировать даты в обратном порядке до 4713 д.э. такое использование может быть бесполезно и не значительно. Календарь был создан в 46 д.э., но подробные не стабилизировался до по крайней мере 8 н.э., и возможно познее в 4-ом столетии. Также, начало года различно от одной культуры к другой - не все соглашаются, что Январь это первый месяц.

---

**JDToJewish**
Преобразовывает дневной Юлианский счет в Еврейский календарь.

**Синтаксис :**
string jdtojewish(int julianday);

---

**JewishToJD**
Преобразовывает дату в Еврейском Календаре на дневной Юлианский счет.

**Синтаксис :**
int jewishtojd(int month, int day, int year);

Еврейский календарь использовался в течение нескольких тысячалетий, но в течение начального периода не было никакой формулы, чтобы определить начало месяца. Новый месяц начинался когда замечено полнолуние.

---

**JDToFrench**
Преобразовывает дневной Юлианский счет в Французский Республиканский календарь.

**Синтаксис :**
string jdtofrench(int month, int day, int year);

Преобразовывает дневной Юлианский счет в Французский Республиканский календарь.

---

**FrenchToJD**
Преобразовывает дату и Французского Республиканского календаря в дневной Юлианский счет.

**Синтаксис :**
int frenchtojd(int month, int day, int year);

Эта программа преобразовывают даты в начиная с 1 до 14 (Грегорианские даты 22 Сентября 1792 до 22 1806 Сентября). Это покрывает тот период когда календарь использовался.

---

**JDMonthName**
Возвращает название месяца.

**Синтаксис :**

string jdmonthname(int julianday, int mode);

Возвращает строку с названием месяца. main сообщает функции, в какой календарь нужно преобразовать дневнрй Юлианский счет на, и какой тип месячных имен должен быть возвращен.

**Календарные способы**

| | |
|---|---|
| 0 | Грегорианский - apreviated |
| 1 | Грегорианский |
| 2 | Юлианский - apreviated |
| 3 | Юлианский |
| 4 | Еврейский |
| 5 | Французский Республиканский |

**JDDayOfWeek**
Возвращает день недели.

**Синтаксис :**
mixed jddayofweek(int julianday, int mode);

Возвращает день недели. Может возвратить строку(string) или int в зависимости от способа(mode).

**Календарные недельные пути**

| Способ | Значение |
|---|---|
| 0 | возвращает дневной номер как int (0=воскресенье, 1=понедельник, и т.п.) |
| 1 | возврат string содержание дня недели (английское-грегорианское) |
| 2 | возвращает string содержа abreviated дени недели (английский-грегорианский) |

## Работа с файлами : Открытие файла

**fopen**
Открывает файл и привязывает его к дескриптору.

**Синтаксис :**
int fopen(string $filename, string $mode, bool $use_include_path=false)

Открывает файл с именем *$filename* в режиме *$mode* и возвращает дескриптор открытого файла. Если операция "провалилась", то функция возвращает false. Необязательный параметр *use_include_path* говорит о том, что, если задано относительное имя файла, его следует искать также и всписке путей, используемом инструкциями **include** и **require**. Обычно этот параметр не используют.
Параметр *$mode* может принимать следующие значения:

r
: - Файл открывается только для чтения. Если файла не существует, вызов регистрирует ошибку. После удачного открытия указатель файла устанавливается на его первый байт, т.е. на начало.

r+
: - Файл открывается одновременно на чтение и запись. Указатель текущей позиции устанавливается на ее первый байт. Если файла не существует,

возвращает false. Если в момент записи указатель файла установлен где-то в середине файла, то данные запишутся прямо поверх уже имеющихся, а не раздвинут их, при необходимости увеличив размер файла.

w
- Создает новый пустой файл. Если на момент вызова уже был файл с таким именем, то он предварительно уничтожается. В случае неверно заданного имени файла вызов "проваливается".

w+
- Аналогичен r+, но если файл изначально не существовало, создает его. После этого с файлом можно работать как в режиме чтения, так и записи. Если файл существовал до момента вызова, его содержимое удаляется.

a
- Открывает существующий файл в режиме записи, и при этом сдвигает указатель текузей позиции за последний байт файла. Как водится, вызов неуспешен в случае отсутствия файла.

a+
- Открывает файл в режиме чтения и записи, указатель файла устанавливается на конец файла, при этом содержимое файла не уничтожается. Отличается от a тем, что если файла изначально не существовало, то он создается. Этот режим полезен, если вам нужно что-то дописать в файл, но вы не знаете, создан ли уже такой файл.

Но это еще не полное описание параметра *$mode*. Дело в том, что в конце любой из строк r,w,a,r+,w+ и a+ может находиться еще один необязательный символ - b или t. Если указан b (или не указан вообще никакой), то файл открывается в режиме бинарного чтения/записи. Если же это t, то для файла устанавливается режим трансляции символа перевода строки, т.е. он воспринимается как текстовой.

---

**tmpfile**
Создает новый временный файл с уникальным именем и открывает его на чтение и запись.

**Синтаксис :**
int tmpfile()

В дальнейшем вся работа должна вестись с возвращенным файловым дескриптором, потому что имя файла недоступно.

Пространство, занимаемое временным файлом, автоматически освобождается при его закрытии и при завершении работы программы.

---

## Работа с файлами : Закрытие файла

**fclose**
Закрывает файл, открытый предварительно функцией **fopen()**.

**Синтаксис :**
int fclose(int $fp)

Возвращает false, если файл закрыть не удалось (например, что-то с ним случилось или же разорвалась связь с удаленным хостом). В противном случае возвращает значение "истина".

Всегда нужно закрывать FTP- и HTTP-соединения, потому что в противном случае

"беспризорный" файл приведет к неоправданному простою канала и излишней загрузке сервера. Кроме того, успешно закрыв соединение, вы будете уверены в том, что все данные были доставлены без ошибок.

---

## Работа с файлами : Чтение и запись

**fread**
Читает из открытого файла определенное количество символов.

**Синтаксис :**
string fread(int $f, int $numbytes)

Читает из файла *$f $numbytes* символов и возвращает строку этих символов. После чтения указатель файла продвигается к следующему после прочитанного блока позициям. Если *$numbytes* больше, чем можно прочитать из файла, возвращается то, что удалось считать. Этот прием можно использовать, если вам нужно считать в строку файл целиком. Для этого просто задайте в *$numbytes* очень большое число. Но если вы заботитесь об экономии памяти в системе, так поступать не рекомендуется.

---

**fwrite**
Запись в файл.

**Синтаксис :**
int fwrite(int $f, string $str)

Записывает в файл *$f* все содержимое строки *$str*. Эта функция составляет пару для **fread()**, действуя "в обратном направлении".

При работе с текстовыми файлами (то есть когда указан символ t в режиме открытия файла) все \n авоматически преобразуются в тот разделитель строк, который принят в вашей операционной системе.

---

**fgets**
Читает из файла одну строку, заканчивающуюся символом новой строки \n.

**Синтаксис :**
string fgets(int $f, int $length)

Этот символ также считывается и включается в результат. Если строка в файле занимает больше *$length-1* байтов, то возвращаются только ее *$length-1* символов. Функция полезна, если вы открыли файл и хотите "пройтись" по всем ее строкам. Однако даже в этом случае (и быстрее) будет воспользоваться функцией **File()**. Стоит также заметить, что эта функция (как и функция **fread()**) в случае текстового режима в Windows заботиться о преобразовании пар \r\n в один символ \n.

---

**fputs**
Полный аналог **fwrite()**.

**Синтаксис :**
int fputs(int $f, string $str)

---

**fgetcsv**
Функция для работы с одним из форматов файлов, в котором может сохранять

данные Excel.

**Синтаксис :**
list fgetcsv(int $f, int $length, char $delim=",")

Функция читает строку из файла, заданного дескриптором *$f*, и разбивает ее по символу *$delim*. Параметр *$delim* должен обязательно быть строкой из одного символа, в противном случае принимается во внимание только первый символ этой строки. Функция возвращает получившийся список или false, если строки кончились. Параметр *$length* задает максимальную длину строки точно так же, как это делается в **fgets()**. Пустые строки в файле не игнорируются, а возвращаются как список из одного элемента - пустой строки.
**Пример:**

```
$f=fopen("file.csv","r") or die("Ошибка");
for($i=0; $data=fgetscsv($f, 1000, ";"); $i++) {
  $num = count($data);
  if($num==1 && $data[0]==="") continue;
  echo "<h3>Строка номер $i ($num полей):</h3>";
  for($c=0; $c<$num; $c++)
    print "[$c]: $data[$c]<br>";
}
fclose($f);
```

---

## Работа с файлами : Положение указателя текущей позиции

**feof**
Указатель конца файла.

**Синтаксис :**
int feof(int $f)

Возвращает true, если достигнут конец файла (то есть если указатель файла установлен за концом файла).
**Пример:**

```
$f=fopen("myfile.txt","r");
while(!feof($f))
{  $str=fgets($f);
   // Обрабатываем очередную строку $str

}
fclose($f);
```

---

**fseek**
Устанавливает указатель файла на определенную позицию.

**Синтаксис :**
int fseek(int $f, int $offset, int $whence=SEEK_SET)

Устанавливает указатель файла на байт со смещением *$offset* (от начала файла, от его конца или от текущей позиции, в зависимости от параметра *$whence*). Это может и не сработать, если дескриптор *$f* ассоциирован не с обычным локальным файлом, а с соединением HTTP или FTP.

Параметр *$whence* задает с какого места отсчитывается смещение *$offset*. В PHP для этого существуют три константы, равные, соответственно, 0, 1 и 2:

SEEK_SET
- устанавливает позицию начиная с начала файла;
SEEK_CUR
- отсчитывает позицию относительно текущей позиции;
SEEK_END
- отсчитывает позицию относительно конца файла;
В случае использования последних двух констант параметр *$offset* вполне может быть отрицательным (а при применении *SEEK_END* он будет отрицательным наверняка). В случае успешного завершения эта функция возвращает 0, а в случае неудачи -1.

---

**ftell**
Возвращает положение указателя файла.

**Синтаксис :**
int ftell(int $f)

---

## Работа с файлами : Функции для определения типов файлов

**file_exists**
Проверяет существование вызываемого файла.

**Синтаксис :**
bool file_exists(string filename)

Возвращает true, если файл с именем *filename* существует на момент вызова. Следует использовать эту функцию с осторожностью. Например, следующий код никуда не годится с точки зрения безопасности:

```
$fname="/etc/passwd";
if(!file_exists($fname))
  $f=fopen($fname,"w");
else
  $f=fopen($fname,"r");
```

Дело в том, что между вызовом **file_exists()** и открытием файла в режиме w проходит некоторое время, в течение которого другой процесс может вклиниться и подменить используемый нами файл. Данная проблема выходит на передний план при написании сценария счетчика.

Функция не работает с удаленными файлами, файл должен находиться в доступной для сервера файловой системе.
Результаты функции кэшируются, см. функцию clearstatcache().

---

**filetype**
Возвращает тип файла.

**Синтаксис :**
string filetype(string filename)

Возвращает строку, которая описывает тип файла с именем *filename*. Если такого файла не существует, возвращает false.
После вызова строка будет содержать одно из следующих значений:
*file* - обычный файл;
*dir* - каталог;
*link* - символическая ссылка;
*fifo* - fifo-канал;

*block* - блочно-ориентированное устройство;
*char* - символьно-ориентированное устройство;
*unknown* - неизвестный тип файла;

---

**is_file**
Проверка существования обычного файла.

**Синтаксис :**
bool is_file(string filename)

Возвращает true, если *filename* - обычный файл.

---

**is_dir**
Проверка существования каталога.

**Синтаксис :**
bool is_dir(string filename)

Возвращает true, если каталог *filename* существует.

---

**is_link**
Проверка существования символической ссылки на файл.

**Синтаксис :**
bool is_link(string filename)

Возвращает true, если *filename* - символическая ссылка.

Функция не работает под Windows.

---

**is_readable**
Проверка существования файла, доступного для чтения.

**Синтаксис :**
bool is_readable(string filename)

Возвращает true, если файл может быть открыт для чтения.
Обычно PHP осуществляет доступ к файлу с привелегиями пользователя, запускающего web-сервер (часто "nobody"). Соображения безопасности должны приниматься в расчет.

---

**is_writeable**
Проверка существования файла, доступного для записи.

**Синтаксис :**
bool is_writeable(string filename)

Возвращает true, если в файл можно писать.
Обычно PHP осуществляет доступ к файлу с привелегиями пользователя, запускающего web-сервер (часто "nobody"). Соображения безопасности должны приниматься в расчет.

---

**is_executable**
Проверка существования запускаемого файла.

**Синтаксис :**
bool is_executable(string filename)

Возвращает true, если файл *filename* - исполняемый.

---

**is_uploaded_file**
Проверка существования файла, загруженного методом HTTP POST.

**Синтаксис :**
bool is_uploaded_file(string filename)

Возвращает true, если файл с именем *filename* был загружен на сервер посредством HTTP POST.
Часто это полезно, чтобы убедиться, что пользователи из злого умысла не пытались заставить сценарий работать с теми файлами, с которыми им работать не следует, например: /etc/passwd.

---

## Работа с файлами : Определение параметров файла

**stat**
Функция собирает вместе всю информацию, выдаваемую операционной системой для указанного файла, и возвращает ее в виде массива.

**Синтаксис :**
array stat(string $filename)

Этот массив всегда содержит следующие элементы с указанными ключами:
```
0  - устройство;
1  - Номер узла inode;
2  - атрибуты защиты файла;
3  - число синонимов ("жестких" ссылок) файла;
4  - идентификатор uid владельца;
5  - идентификатор gid группы;
6  - тип устройства;
7  - размер файла в байтах;
8  - время последнего доступа в секундах, прошедших с 1 января 1970 года;
9  - время последней модификации содержимого файла;
10 - время последнего изменения атрибутов файла;
11 - размер блока;
12 - число занятых блоков;
```
Этот массив помещает информацию, которая доступна в системах Unix. Под Windows многие поля могут быть пусты.

Если *$filename* задает не имя файла, а имя символической ссылки, то всетаки будет возвращена информация о том файле, на который ссылается эта ссылка (а не о ссылке).

---

**fileatime**
Возвращает время последнего доступа к файлу.

**Синтаксис :**
int fileatime(string filename)

Время выражается в количестве секунд, прошедших с 1 января 1970 года (Unix timestamp). Если файл не обнаружен, возвращает false.
Атрибут времени последнего доступа к файлу изменяется каждый раз, когда данные файла читаются. Так как это сильно снижает производительность при интенсивной работе с файлами и каталогами, часто изменение этого атрибута в операционных системах блокируют, и тогда функция бесполезна.

**filemtime**
Возвращает время последнего изменения файла или false в случае отсутствия файла.

**Синтаксис :**
int filemtime(string $filename)

---

**filectime**
Возвращает время создания файла.

**Синтаксис :**
int filectime(string $filename)

---

**filesize**
Возвращает размер файла в байтах или false, если файла не существует.

**Синтаксис :**
int filesize(string $filename)

---

**touch**
Устанавливает время модификации.

**Синтаксис :**
int touch(string $filename [, int $timestamp])

Устанавливает время модификации указанного файла *$filename* равным *$timestamp* (в секундах, прошедших с 1 января 1970 года). Если второй параметр не указан, то подразумевается текущее время. Вслучае ошибки возвращает false.

Если файл с указанным именем не существует, он создается пустым.

---

## Работа с файлами : Функции для работы с именами файлов

**basename**
Выделяет имя файла из пути.

**Синтаксис :**
string basename(string $path)

Выделяет основное имя из пути *$path*
**Примеры:**
```
echo basename("/home/somebody/somefile.txt");  // выводит "somefile.txt"
echo basename("/");                            // ничего не выводит
echo basename("/.");                           // выводит "."
echo basename("/./");                          // также выводит "."
```
Функция **basename()** не проверяет существование файла. Она просто берет часть строки после самого правого слеша и возвращает ее.
Эта функция правильно обрабатывает как прямые, так и обратные слеши под Windows.

---

**dirname**
Выделяет имя каталога.

**Синтаксис :**
string dirname(string $path)

Возвращает имя каталога, выделенное из пути *$path*. Функция довольно "разумна" и умеет выделять нетривиальные ситуации, которые описаны в примерах:

```
echo dirname("/home/file.txt"); // выводит "/home"
echo dirname("../file.txt");    // выводит ".."
echo dirname("/file.txt");      // выводит "/" под Unix,
                                //         "\" под Windows
echo dirname("/");              // то же самое
echo dirname("file.txt");       // выводит "."
```

Если функции **dirname()** передать просто имя файла, она вернет ".", что означает "текущий каталог".

---

**tempnam**
Генерирует уникальное имя файла в определенном каталоге.

**Синтаксис :**
string tempnam(string $dir, string $prefix)

Генерирует имя файла в каталоге *$dir* с префиксом *$prefix* в имени, причем так, чтобы созданный под этим именем в будущем файл был уникален.
Для этого к строке *$prefix* присоединяется некое случайное число.
Например, вызов tempnam("/tmp","temp") может возвратить /tmp/temp3a6b243c.
Если такое имя нужно создать в текущем каталоге, передайте $dir="."

---

**realpath**
Преобразует относительный путь в абсолютный.

**Синтаксис :**
string realpath(string $path)

Преобразует относительный путь *$path* в абсолбтный, т.е. начинающийся от корня.
**Пример:**

```
echo realpath("../t.php"); // например, /home/t.php
echo realpath(".");        // выводит имя текущего каталога
```

Файл, который указан в параметре *$path*, должен существовать, иначе функция фозвратит false.

---

## Работа с файлами : Функции манипулирования целыми файлами

**copy**
Копирует файл.

**Синтаксис :**
bool copy(string $src, string $dst)

Копирует файл с именем *$src* в файл с именем *$dst*. При этом, если файл *$dst* на момент вызова существовал, осуществляется его перезапись.
Функция возвращает true, если копирование прошло успешно, а в случае провала - false.

Функция не выполняет переименования файла, если его новое имя расположено

в другой файловой системе (на другой смонтированной системе в Unix или на другом диске в Windows).

---

**unlink**
Удаление файла.

**Синтаксис :**
bool unlink(string $filename)

Удаляет файл с именем *$filename*. В случае неудачи возвращает false, иначе - true.

Надо заметить, что файл удаляется только в том случае, если число "жестких" ссылок на него стало равным 0. Правда, эта схема специфична для Unix-систем.

---

**file**
Считывает файл и разбивает его по строкам.

**Синтаксис :**
list file(string $filename)

Считывает файл с именем *$filename* целиком и возвращает массив-список, каждый элемент которого соответствует строке в прочитанном файле. Неудобство этой функции состоит в том, что символы конца строки (обычно \n), не вырезаются из строк файла, а также не транслируются, как это делается для текстовых файлов.

---

## Работа с файлами : Другие функции

**ftruncate**
Усекает файл.

**Синтаксис :**
bool ftruncate(int $f, int $newsize)

Эта функция усекает открытый файл *$f* до размера *$newsize*. Разумеется, файл должен быть открыт в режиме, разрешающим запись.
Например, следующий код очищает весь файл:

```
ftruncate($f,0);
```

---

**fflush**
Немедленная запись всех изменений в файле.

**Синтаксис :**
void fflush(int $f)

Заставляет PHP немедленно записать на диск все изменения, которые производились до этого с открытым файлом *$f*. Что это за изменения? Дело в том, что для повышения производительности все операции записи в файл буферизируются: например, вызов fputs($f, "Это строка!") не приводит к непосредственной записи данных на диск - сначало они попадают во внутренний буфер (обычно размером 8К). Как только буфер заполняется, его содержимое отправляется на диск, а сам он очищается, и все повторяется вновь. Особенный

выигрыш от буферзации чувствуется в сетевых операциях, когда просто глупо отправлять данные маленькими порциями.

---

**set_file_buffer**
Устанавливает размер буфера.

**Синтаксис :**
int set_file_buffer(int $f, int $size)

Эта функция устанавливает размер буфера, о котором говорилось **выше**, для указанного открытого файла *$f*.
Чаще всего она используется так:

set_file_buffer($f,0);

Приведенный код отключает буферизацию для указанного файла, так что теперь все данные, записываемые в файл, немедленно отправляются на диск или в сеть.

---

**flock**
Блокирование файла.

**Синтаксис :**
bool flock(int $f, int $operation [, int $wouldblock])

Функция устанавливает для указанного открытого дескриптора файла *$f* режим блокировки, который бы хотел получить текущий процесс. Этот режим задается аргументом *$operation* и может быть одной из следующих констант:

- LOCK_SH (или 1) - разделяемая блокировка;
- LOCK_EX (или 2) - исключительная блокировка;
- LOCK_UN (или 3) - снять блокировку;
- LOCK_NB (или 4) - эту константу нужно прибавить к одной из предыдущих,
  если вы не хотите, чтобы программа подвисала на **flock()** в ожидании своей очереди, а сразу возвращала управление.

В случае, если был затребован режим без ожидания, и блокировка не была успешно установлена, в необязятельный параметр-переменную *$wouldblock* будет записано значение истина true.
В случае ошибки функция возвращает false, а в случае успешного завершения - true.

---

## Функции для работы с каталогами : Манипулирование каталогами

**mkdir**
Создание каталога.

**Синтаксис :**
bool mkdir(string $name, int $perms)

Создает каталог с именем *$name* и правами доступа *perms*. Права доступа для каталогов указываются точно так же, как и для файлов. Чаще всего значение *$perms* устанавливают равным 0770 (предваряющий ноль обязателен - он указывает PHP на то, что это - восьмеричная константа, а не десятичное число).

**Пример:**
```
mkdir("my_directory",0755);
     // создает подкаталог в текущем каталоге
mkdir("/data");
     // создает подкаталог data в корневом каталоге
```
В случае успеха функция возвращает true, иначе - false.

---

**rmdir**
Удаление каталога.

**Синтаксис :**
bool rmdir(string $name)

Удаляет каталог с именем *$name*.
Каталог должен быть пустым, а его атрибуты должны позволять это.
В случае успеха функция возвращает true, иначе - false.

---

**chdir**
Смена текущего каталога.

**Синтаксис :**
int chdir(string $directory);

Изменяет текущий PHP каталог на directory. Возвращает FALSE если не может изменить, TRUE если смена произошла. Параметр *$directory* может определять и относительный путь, задающийся от текущего каталога.

**Примеры:**
```
chdir("/tmp/data"); // переходим по абсолютному пути
chdir("./js"); // переходим в подкаталог текущего каталога
chdir(".."); // переходим в родительский каталог
chdir("~/data"); // переходим в /home/пользователь/data (для Unix)
```

---

**getcwd**
Полный путь.

**Синтаксис :**
string getcwd()

Данная функция возвращает текущую директорию, относительно которой проводятся файловые операции, т.е. возвращает полный путь к текущему каталогу, начиная от "корня" (/). Если такой путь не можнет быть отслежен, вызов "проваливается" и возвращается false.

---

**diskfreespace**
Определяет свободное пространство в каталоге

**Синтаксис :**
float diskfreespace (string directory);

Данная функция возвращает в байтах свободное пространство в каталоге *directory*, то есть в соответствующей ей файловой системе или на разделе диска.

**Пример:**
```
$diskspace=diskfreespace("/");
// Тем самым мы определили свободное место в корневой директории "/"
```

---

# Функции для работы с каталогами : Работа с записями

**dir**
Класс каталога (псевдо-объектно ориентированный механизм).

**Синтаксис :**
new dir(string directory);

Псевдо-объектно ориентированный механизм для получения списка файлов каталога. Открывает каталог из directory.
После этого становятся доступны два свойства объекта: дескриптор каталога handle и строка path, указывающая, какой каталог в настоящий момент используется. Эти свойства доступны, если только каталог был открыт. Свойство handle может быть использован вместе с другими функциями работы с каталогом типа readdir(), rewinddir() и closedir().
Для класса доступны три метода: чтение, возврат к началу и закрытие (read, rewind и close соответственно).

**Пример :**
```
$d = dir("/etc");
echo "Handle: ".$d->handle."<br>\n";
echo "Path: ".$d->path."<br>\n";
while($entry=$d->read()) { // Последовательно выводить
  echo $entry."<br>\n";    // имя каждого файла,
}                          // имеющегося в каталоге
$d->close();
```

---

**closedir**
Закрыть дескриптор(handle) каталога.

**Синтаксис :**
void closedir(int dir_handle);

Закрывает поток каталога, обозначенный как dir_handle. Поток предварительно должен быть открыт финкцией opendir().

---

**opendir**
Открыть дескриптор каталога.

**Синтаксис :**
int opendir(string path);

Возвращает дескриптор открытого каталога *path*, который в последующем используется в функциях closedir(), readdir(), и rewinddir().

---

**readdir**
Получение имени следующего файла в списке каталога.

**Синтаксис :**
string readdir(int dir_handle);

Возвращает имя следующего файла из каталога. Имена файлов возвращаются в виде неупорядоченной последовательности.

**Пример:**
```
<?php
  $handle=opendir(".");
```

```
  echo "Directory handle: $handle\n";
  echo "Files:\n";
  while ($file = readdir($handle)) {
    echo "$file\n";
  }
  closedir($handle);
?>
```

Следует отметить, что функция также возвращает значения "." и "..". Если эти значения не требуются, то их можно исключить следующим образом:

```
<?php
$handle=opendir(".");
while($file=readdir($handle)) {
  if($file != "." && $file != "..") {
    echo "Имя файла: $file<br>";
  };
};
closedir($handle);
?>
```

---

**rewinddir**
Реинициализация дескриптора каталога.

**Синтаксис :**
void rewinddir(int dir_handle);

После вызова этой функции функция readdir() с аргументом *dir_handle* будет возвращать имена файлов с начала в списке каталога.

---

## FTP : Работа с FTP-сервером

**ftp_connect**
Производит подключение к FTP-серверу.

**Синтаксис :**
int ftp_connect(string host [, int port])

В функции **ftp_connect()** аргумент *host* указывает имя сервера, к которому производится подключение, а необязательный аргумент *port* указывает какой порт нужно использовать (по умолчанию это 21).
Функция возвращает дескриптор потока FTP или false в случае ошибки.

---

**ftp_pasv**
Производит переключение пассиваного режима.

**Синтаксис :**
int ftp_pasv(int ftp_stream, int pasv)

Функция **ftp_pasv()** производит переключение режима подключения в пассивный, в случае, если аргумент *pasv* равен true. Если false - то режим подключения будет активный.
В пассивном режиме передача данных инициируется клиентом, а в активном - сервером (это бывает необходимо при блокировке портов у клиента).
Функция возварщает true или false при ошибке.

---

**ftp_login**
Производит вход на сервер FTP.

**Синтаксис :**
int ftp_login(int ftp_stream, string username, string password)

Функция **ftp_login()** производит регистрацию в системе под именем *username* с паролем *password*. Возвращает true или false при ошибке.

---

**ftp_quit**
Производит завершение сеанса FTP.

**Синтаксис :**
int ftp_quit(int ftp_stream)

---

**ftp_pwd**
Производит определение текущего каталога.

**Синтаксис :**
int ftp_pwd(int ftp_stream)

Эта функция возвращает текущий каталог FTP-сервера или false при ошибке.

---

**ftp_cdup**
Производит переход в корневой каталог.

**Синтаксис :**
int ftp_cdup(int ftp_stream)

Функция возвращает true или false при ошибке.

---

**ftp_chdir**
Производит переход в каталог.

**Синтаксис :**
int ftp_chdir(int ftp_stream, string directory)

Функция возвращает true или false при ошибке.

---

**ftp_mkdir**
Производит создание каталога.

**Синтаксис :**
int ftp_mkdir(int ftp_stream, string directory)

Функция возвращает имя созданного каталога или false при ошибке.

---

**ftp_rmdir**
Производит удаление каталога.

**Синтаксис :**
int ftp_rmdir(int ftp_stream, string directory)

Функция true или false при ошибке.

---

**ftp_nlist**
Производит получение листинга каталога.

**Синтаксис :**

int ftp_nlist(int ftp_stream, string directory)

Функция **ftp_nlist()** возвращает массив файловых имен или false при ошибке.

---

**ftp_rawlist**
Получение подробного листинга каталога.

**Синтаксис :**
int ftp_rawlist(int ftp_stream, string directory)

Функция **ftp_rawlist()** выполняет FTP-команду LIST, и возвращате его результаты в массиве, где каждый элемент соответствует строке текста "как есть". Идентификатор типа системы, возвращаемый **ftp_systype()**, может быть использован для определения того, как следует интерпретировать результаты.

---

**ftp_systype**
Возвращает системный идентификатор типа FTP-сервера.

**Синтаксис :**
int ftp_systype(int ftp_stream)

Функция возвращает строковое значение или false в случае ошибки.

---

## FTP : Работа файлами

**ftp_get**
Производит загрузку с FTP-сервера.

**Синтаксис :**
int ftp_get(int ftp_stream, string local_file, string remote_file, int mode)

Функция **ftp_get()** загружает файл под названием *remote_file* с FTP-сервера и локально сохраняет его под именем *local_file*. Параметр *mode* устанавливает режим передачи файла и может принимать значения FTP_ASCII(текстовой) или FTP_BINARY(бинарный, двоичный).
Функция возвращает true или false при ошибке.

---

**ftp_fget**
Производит загрузку и запись файла.

**Синтаксис :**
int ftp_fget(int ftp_stream, string fp, string remote_file, int mode)

Функция **ftp_fget()** загружает файл под названием *remote_file* с FTP-сервера и сохраняет его в файле, который имеет дескриптор *fp*. Параметр *mode* устанавливает режим передачи файла и может принимать значения FTP_ASCII(текстовой) или FTP_BINARY(бинарный, двоичный).
Функция возвращает true или false при ошибке.

---

**ftp_put**
Производит загрузку файла на FTP-сервер.

**Синтаксис :**
int ftp_put(int ftp_stream, string remote_file, string local_file, int mode)

Функция **ftp_put()** загружает файл на FTP-сервер под именем *remote_file*. Параметр *mode* устанавливает режим передачи файла и может принимать значения FTP_ASCII(текстовой) или FTP_BINARY(бинарный, двоичный). Функция возвращает true или false при ошибке.

```
$upload = ftp_put($ftp_id, "C:\\file.txt", "/file.txt", FTP_ASCII);
```

---

**ftp_fput**
Производит чтение и загрузку файла на FTP-сервер.

**Синтаксис :**
int ftp_fput(int ftp_stream, string remote_file, string fp, int mode)

Функция **ftp_fput()** читает открытый файл с дескриптором *fp* до конца и загружает этот файл на FTP-сервер под именем *remote_file*. Параметр *mode* устанавливает режим передачи файла и может принимать значения FTP_ASCII(текстовой) или FTP_BINARY(бинарный, двоичный).
Функция возвращает true или false при ошибке.

---

**ftp_size**
Определяет размер файла.

**Синтаксис :**
int ftp_size(int ftp_stream, string remote_file)

Функция **ftp_size()** возвращает размер файла, заданного в параметре *remote_file*, в байтах или -1 при ошибке.
Не все серверы поддерживают эту возможность.

---

**ftp_mdtm**
Возвращает время последней модификации файла.

**Синтаксис :**
int ftp_mdtm(int ftp_stream, string remote_file)

Функция **ftp_mdtm()** возвращает время, последней модификации, представленное в формате Unix, или -1 при ошибке.
Данная функция не работает с каталогами.

---

**ftp_rename**
Производит переименование файла.

**Синтаксис :**
int ftp_rename(int ftp_stream, string from, string to)

Функция **ftp_rename()** переименовывает файл *from* в *to*.
Функция возвращает true или false в случае ошибки.

---

**ftp_delete**
Производит удаление файла с сервера.

**Синтаксис :**
int ftp_delete(int ftp_stream, string path)

Функция **ftp_delete()** удаляет файл, имя которого задано в параметре *path*.
Функция возвращает true или false в случае ошибки.

---

**ftp_site**
Производит выполнение команды SITE на сервере.

**Синтаксис :**
int ftp_site(int ftp_stream, string cmd)

Функция **ftp_site()** посылает серверу команду cmd.
Т.к. команды SITE не стандартизированы, они могут различаться. Обычно они полезны для изменения прав доступа к файлам и групповой принадлежности.
Функция возвращает true или false в случае ошибки.

---

## Функции IMAP

Для того, чтобы эти функции заработали вы должны скомпилировать PHP с флагом --with-imap.
Этот флаг требут,чтобы была установлена библиотека c-client. Последнюю версию можно получить по адресу ftp://ftp.cac.washington.edu/imap/.
Затем скопируйте c-client/c-client.a в /usr/local/lib или какую либо другую директорию, прописанную в пути, затемскопируйте c-client/rfc822.h, mail.h и linkage.h в /usr/local/include или другую директорию с include-файлами.

Не смотря на имя модуля, имеющиеся в нем функции позволяют выполнять также много других полезных операций, выходящих за рамки простого использования протокола IMAP. Лежащая в основе библиотека C-клиента также поддерживает NNTP, POP3 и методы доступа к локальным почтовым ящикам.

---

**imap_append**
Добавляет текстовое сообщение в указанный почтовый ящик.

**Синтаксис :**
int imap_append(int imap_stream, string mbox, string message, stringflags);

Возвращает true в случае успеха или false иначе.
imap_append() добавляет текстовое сообщение в указанный почтовый ящик mbox. Если указаны необязательные флаги, также записывает в почтовый ящик и флаги. При общении с сервером Cyrus IMAP нужно использовать в качестве ограничителей строки "\r\n" вместо "\n",иначе действие не выполнится.

---

**imap_base64**
Декодирует текст, закодированный с помощью BASE64.

**Синтаксис :**
string imap_base64(string text);

Функция imap_base64() декодирует текст в формате BASE-64. Декодированное сообщение возвращается как строка.

---

**imap_body**
Читает тело сообщения.

**Синтаксис :**
string imap_body(int imap_stream, int msg_number, int flags);

Функция imap_body() возвращает тело сообщения, имеющего номер п/п msg_number в текущем почтовом ящике.

Необязательные флаги это битовые маски из
FT_UID - Номер сообщения msgno является UID-ом сообщения
FT_PEEK - Не устанавливать флаг \Seen если он еще не установлен.
FT_INTERNAL - Возвращаемая строка записана во внутреннем формате и не может быть приведена к канонической форме с CRLF.

---

**imap_check**
Проверяет текущий почтовый ящик.

**Синтаксис :**
array imap_check(int imap_stream);

Возвращает информацию о текущем почтовом ящике. В случае неуспеха возвращает FALSE.
Функция imap_check() проверяет статус текущего почтового ящика на сервере и возвращает информацию в объекте со следующими свойствами :
Date : дата сообщения
Driver : драйвер
Mailbox : название почтового ящика
Nmsgs : количество сообщений
Recent : количество недавно пришедших сообщений

---

**imap_close**
Закрывает поток IMAP.

**Синтаксис :**
int imap_close(int imap_stream, int flags);

Закрывает поток imap. Необязательный флаг CL_EXPUNGE заставляет стереть помеченные на удаление сообщенияпри закрытии.

---

**imap_createmailbox**
Создает новый почтовый ящик.

**Синтаксис :**
int imap_createmailbox(int imap_stream, string mbox);

imap_createmailbox() создает новый почтовый ящик указанный в mbox.Возвращает true в случае успеха и false при ошибке.

---

**imap_delete**
Помечает сообщение из текущего почтового ящика на удаление.

**Синтаксис :**
int imap_delete(int imap_stream, int msg_number);

Возвращает true.
Возвращает true.Функция imap_delete() помечает сообщение, указанное через msg_number на удаление. Настоящее удаление сообщенийосуществляется функцией imap_expunge().

---

**imap_deletemailbox**
Удаляет почтовый ящик.

**Синтаксис :**
int imap_deletemailbox(int imap_stream, string mbox);

Возвращает true в случае успеха и false иначе.

---

**imap_expunge**
Удаляет все сообщения, помеченные на удаление.

**Синтаксис :**
int imap_expunge(int imap_stream);

imap_expunge() удаляет все сообщения помеченные на удаление с помощью imap_delete().
Возвращает true.

---

**imap_fetchbody**
Извлекает простую секцию тела сообщения.

**Синтаксис :**
string imap_fetchbody(int imap_stream, int msg_number, int part_number, flags flags);

Эта функция заставляет извлечь подробную секцию указанного сообщения как текстовую строку.Секция это строка целых чисел, разделенных точками, которые указывают на части тела сообщения в списке частей согласно спецификации IMAP4. Части тела не декодируются этой функцией.
Необязательным параметром к imap_fetchbody () является битовая маска из
FT_UID - msgono является
UID-ом FT_PEEK - не устанавливать флаг \Seen если он не установлен
FT_UID - возвращаемая строка записана во внутреннем формате, которое не может быть канонизированна с помощью CRLF

---

**imap_fetchstructure**
Читает структуру простого сообщения.

**Синтаксис :**
array imap_fetchstructure(int imap_stream, int msg_number);

array imap_fetchstructure(int imap_stream, int msg_number);Эта функция заставляет извлечь всю информацию о структуре сообщения с номером msg_number. Возвращаемая величинаявляется объектом со следующими элементами :
type, encoding, ifsubtype, subtype, ifdescription, description, ifid, id, lines, bytes, ifparameters тип, кодировка, подтип интерфейса, подтип, описание интерфейса, описание, идентификатор интерфейса, строки, байты, параметры интерфейса
Также функция возвращает массив объектов под названием parameters[]. Этот объект имеет следующиесвойства :
attribute, value
атрибут, величина
В случае сообщения из нескольких частей, функция также возвращает массив объектов всех свойств под название parts[].

---

**imap_header**
Читает заголовок сообщения.

**Синтаксис :**
object imap_header(int imap_stream, int msg_number, int fromlength, int subjectlength, int defaulthost);

Эта функция возвращает объект различных элементов заголовка
remail,date,Date,subject,Subject,in_reply_to,message_id,newsgroups,
followup_to,references
toaddress (полная строка To: строка длиной до 1024 символов)
to[] (возвращает массив объектов из строки To, содержит:)
personal
adl
mailbox
host
fromaddress (полная строка From: строка длиной до 1024 символов)
from[] (возвращает массив объектов из строки From, содержит:)
personal
adl
mailbox
host
ccaddress (полная строка Cc: строка длиной до 1024 символов)
cc[] (возвращает массив объектов из строки Cc, содержит)
personal
adl
mailbox
host
bccaddress (полная строка Bcc: строка длиной до 1024 символов)
bcc[] (возвращает массив объектов из строки Bcc, содержит:)
personal
adl
mailbox
host
reply_toaddress (полная строка Reply_to: строка длиной до 1024 символов)
reply_to[] (возвращает массив объектов из строки Reply_to, содержит:)
personal
adl
mailbox
host
senderaddress (полная строка Sender: строка длиной до 1024 символов)
sender[] (возвращает массив объектов из строки Sender, содержит:)
personal
adl
mailbox
host
return_path (полная строка Return-path: строка длиной до 1024 символов)
return_path[] (возвращает массив объектов из строки Return_path, содержит:)
personal
adl
mailbox
host
udate ( дата сообщения в формате времени unix)
fetchfrom (строка From, отформатированная до fromlength символов)
fetchsubject (строка Subject, отформатированная до subjectlength символов)

---

**imap_headers**
Возвращает заголовки всех сообщений в почтовом ящике.

**Синтаксис :**
array imap_headers(int imap_stream);

Возвращает строковый массив из информации по заголокам. Один элемент массива на сообщение.

**imap_listmailbox**
Читает список почтовых ящиков.

**Синтаксис :**
array imap_listmailbox(int imap_stream, string ref, string pat);

Возвращает массив, содержащий названия почтовых ящиков.

---

**imap_listsubscribed**
Перечисляет все подписанные ящики.

**Синтаксис :**
array imap_listsubscribed(int imap_stream, string ref, string pattern);

Возвращает массив всех почтовых ящиков на которые Вы подписаны. Аргументы ref и pattern указывают начальноеместорасположение откуда начинать поиск и шаблон, которому должны удовлетворять названия почтовых ящиков.

---

**imap_mail_copy**
Копирует указанные сообщения в почтовый ящик.

**Синтаксис :**
int imap_mail_copy(int imap_stream, string msglist, string mbox, int flags);

Возвращает true в случае успеха и false иначе.
Копирует почтовые сообщения указанные с помощью msglist в почтовый ящик mbox. msglist - это диапазон, а не простономера сообщений.
Флаги - это битовые маски из
CP_UID - номера в последовательности содержат
UID-ы CP_MOVE - после копирования удалить сообщения из текущего почтового ящика

---

**imap_mail_move**
Переносит указанные сообщения в почтовый ящик.

**Синтаксис :**
int imap_mail_move(int imap_stream, string msglist, string mbox);

Переносит почтовые сообщения указанные с помощью msglist в почтовый ящик mbox. msglist - это диапазон, а не простономера сообщений.
Возвращает true в случае успеха и false иначе.

---

**imap_num_msg**
Выдает количество сообщений в текущем почтовом ящике.

**Синтаксис :**
int imap_num_msg(void);

Возвращает количество сообщений в текущем почтовом ящике.

---

**imap_num_recent**
Возвращает количество недавно пришедших сообщений в текущем почтовом ящике.

**Синтаксис :**
int imap_num_recent(int imap_stream);

**imap_open**
Подключение к серверу (открытие почтового ящика).

**Синтаксис :**
int imap_open(string mailbox, string username, string password [, int flags]);

Функция **imap_open()** возвращает дескриптор почтового ящика IMAP (дескриптор подключения к серверу IMAP) или false при ошибке.
Эта функция может быть использована для открытия потоков к POP3 и NNTP серверам, но в этом случае некоторые функции будут недоступны.

Аргумент *mailbox* - задает имя сервера и путь к почтовому ящику. Имя сервера следует заключать в фигурные скобки "{" и "}", внутри которых должно содержаться: имя сервера (или его IP-адрес), возможно указание протокола (который начинается со слеша "/") и номера порта Для того, чтобы присоединиться к серверу IMAP на 143-й порт на локальной машине сделайте следующее:
$mbox = imap_open("{localhost:143}INBOX","user_id","password");
Для того, чтобы подсоединиться к POP3-серверу на 110-й порт на локальном сервере используйте:
$mbox = imap_open("{localhost/pop3:110}INBOX","user_id","password");
Для того, чтобы подсоединиться к NNTP-серверу на 119-й порт на локальном сервере используйте:
$nntp = imap_open("{localhost/nntp:119}comp.test","","");
Для того, чтобы подсоединиться к удаленному серверу замените "localhost" на имя или IP-адрес сервера к которомуВы хотите подсоединиться.
Опции - битовая маска из
OP_READONLY - Открыть почтовый ящик в режим "только чтение"
OP_ANONYMOUS - Не использовать или не обновлять .newsrc при использовании новостей
OP_HALFOPEN - Для IMAP и NNTP устанавливает соединение, но не открывает почтовый ящик
CL_EXPUNGE - Автоматически очищать почтовый ящик при закрытии

---

**imap_ping**
Проверяет поток IMAP на работоспособность.

**Синтаксис :**
int imap_ping(int imap_stream);

Возвращает true если поток еще работоспособен и false иначе.
Функция imap_ping() проверяет поток на работоспособность. Он может также проверять новую почту; это предпочтительный метод для периодической проверки новой почты и "живучести" удаленных серверов.

---

**imap_renamemailbox**
Переименовывает старый почтовый ящик в новый.

**Синтаксис :**
int imap_renamemailbox(int imap_stream, string old_mbox, string new_mbox);

Эта функция переименовывает старый почтовый ящик в новый.
Возвращает true в случае успеха и false иначе.

---

**imap_reopen**
Заново открывает поток IMAP на новый почтовый ящик.

**Синтаксис :**
int imap_reopen(string imap_stream, string mailbox, string [flags]);

Возвращает true в случае успеха и false иначе.
Эта функция заново открывает указанный поток на новый ящик.
Опции - битовая маска из
OP_READONLY - Открыть почтовый ящик в режиме только чтение
OP_ANONYMOUS - Не использовать или не обновлять .newsrc при работе с новостями
OP_HALFOPEN - Для IMAP и NNTP устанавливает связь но не открывает почтовый ящик
CL_EXPUNGE - Очищает почтовый ящик при закрытии

---

**imap_subscribe**
Подписывает на почтовый ящик.

**Синтаксис :**
int imap_subscribe(int imap_stream, string mbox);

Возвращает true в случае успеха и false иначе.

---

**imap_undelete**
Снимает отметку с сообщения помеченного на удаление.

**Синтаксис :**
int imap_undelete(int imap_stream, int msg_number);

Эта функция снимает отметку с сообщения помеченного на удаление функцией imap_delete().
Возвращает true в случае успеха и false иначе.

---

**imap_unsubscribe**
Снимает подписку с почтового ящика.

**Синтаксис :**
int imap_unsubscribe(int imap_stream, string mbox);

Возвращает true в случае успеха и false иначе.

---

**imap_qprint**
Конвертирует строку формата quoted-printable в 8-битовую строку.

**Синтаксис :**
string imap_qprint(string string);

Возвращает 8-битовую (бинарную) строку.

---

**imap_8bit**
Конвертирует 8-битовую строку в формат quoted-printable.

**Синтаксис :**
string imap_8bit(string string);

Возвращает строку в формате quoted-printable.

---

**imap_binary**
Конвертирует 8-битную строку в формат base64.

**Синтаксис :**
string imap_binary(string string);

Возвращает строку в формате base64.

---

**imap_scanmailbox**
Читает список почтовых ящиков, проводит поиск в названиях ящиков.

**Синтаксис :**
array imap_scanmailbox(int imap_stream, string string);

Возвращает массив, содержащий имена почтовых ящиков, которые имеют строку string в названии.

---

**imap_mailboxmsginfo**
Получает информацию о текущем почтовом ящике.

**Синтаксис :**
array imap_mailboxmsginfo(int imap_stream);

Возвращает информацию о текущем почтовом ящике. FALSE в случае неудачи.
Функция imap_mailboxmsginfo() проверяет статус текущего почтового ящика на сервере и возвращает информацию в объекте со следующими свойствами:
Date : дата сообщения
Driver : драйвер
Mailbox : название почтового ящика
Nmsgs : количество сообщений
Recent : количество недавно пришедших сообщений
Unread : количество непрочитанных сообщений
Size : размер почтового ящика

---

**imap_rfc822_write_address**
Возвращает правильно отформатированный email адрес.

**Синтаксис :**
string imap_rfc822_write_address(string mailbox, string host, string personal);

Возвращает правильно отформатированный email адрес по данному почтовому ящику, хосту и персональной информации.

---

**imap_rfc822_parse_adrlist**
Проводит разбор адресной строки.

**Синтаксис :**
string imap_rfc822_parse_adrlist(string address, string default_host);

Эта функция разбирает адресную строку и для каждого адреса возвращает массив объектов.
Есть 4 типа объектов:
mailbox - название почтового ящика (имя пользователя)
host - название хоста
personal - личное имя
adl - путь к домену-источнику

---

**imap_setflag_full**
Устанавливает флаги на сообщения.

**Синтаксис :**
string imap_setflag_full(int stream, string sequence, string flag, string options);

Эта функция заставляет добавить указанный флаг к набору флагов сообщения в указанной последовательности.
options - это битовая маска из ST_UID
Аргументы последовательности содержат UIDы вместо номеров

---

**imap_clearflag_full**
Очищает флаги сообщения.

**Синтаксис :**
string imap_clearflag_full(int stream, string sequence, string flag, string options);

Эта функция заставляет удалить флаги из набора флагов сообщения в указанной последовательности.
options - это битовая маска из ST_UID
Аргументы последовательности содержат UIDы вместо номеров

---

**imap_sort**
Сортирует сообщения в текущем почтовом ящике.

**Синтаксис :**
string imap_sort(int stream, int criteria, int reverse, int options);

Возвращает массив номеров сообщений рассортированных по данному параметру
Rev должен быть равен 1 если нужна сортировка в обратном порядке
Критерии сортировки (должен быть указан только один): SORTDATE - по дате сообщения
SORTARRIVAL - по дате поступления
SORTFROM - по полю From
SORTSUBJECT - по теме сообщения
SORTTO - по полю To
SORTCC - по полю cc
SORTSIZE - по размеру
опции - битовая маска из
SE_UID - Возвратить UIDы вместо номеров последовательности
SE_NOPREFETCH - Не извелекать заранее найденные сообщения

---

**imap_fetchheader**
Возвращает заголовок сообщения.

**Синтаксис :**
string imap_fetchheader(int imap_stream, int msgno, int flags);

Эта функция заставляет извлечь полный, неотфильтрованный заголовок указанного сообщения в формате RFC 822 как текстовую строку.
**Опции:**
FT_UID msgno является UID-ом
FT_INTERNAL Возвращаемая строка записана во внутреннем формате без каких-либо попыток канонизировать ее с помощью CRLF
FT_PREFETCHTEXT RFC822. Текст должен быть предварительно разобран. Это поможет избежать эстренных задержек если требуется извлечь полный текст сообщения (например, в операции "сохранить в локальном файле")

**imap_uid**
Эта функция возвращает UID по данному номеру сообщения в последовательности.

**Синтаксис :**
string imap_uid(string mailbox, int msgno);

---

## Функции SNMP

**snmpget**
Получает объект SNMP.

**Синтаксис :**
int snmpget(string hostname, string community, string object_id);

Возвращает значание SNMP объекта при успехе и false при ошибке.
Функция snmpget() используется для чтения значения SNMP объекта, указанного в object_id.
SNMP агент определяется именем хоста hostname и группа чтения определяется параметром community.
snmpget("127.0.0.1", "public", "system.SysContact.0")

---

**snmpwalk**
Получает все SNMP объекты у агента.

**Синтаксис :**
array snmpwalk(string hostname, string community, string object_id);

Возвращает массив значений SNMP объектов начиная с object_id и false при ошибке.
Функция snmpwalk() используется для чтения всех значений у SNMP агента, определяемого параметром hostname.
Community определяет группу чтения для агента.
Нулевой object_id берется как корень дерева SNMP объектов и все объекты под этим деревом возвращаются как массив.
Если object_id указан, то возвращаются все SNMP объекты ниже этого объекта.
$a = snmpwalk("127.0.0.1", "public", "");
Указанный выше вызов функции вернет все SNMP объекты из SNMP агента, напущенного на локальном хосте.
По всем значениям можно пройти с помощью цикла :

```
for($i=0; $i<count($a); $i++) {
        echo $a[$i];
}
```

---

## Функции Vmailmgr

Эти функции требуют пакетов QMAIL (www.qmail.org) и vmailmgr Bruce Guenter http://www.qcc.sk.ca/~bguenter/distrib/vmailmgr/
Для всех функций следующие две переменные определяются как: строка vdomain - имя домена вашего виртуального домена (vdomain.com) , строка basepwd - пароль для "real" пользователя, который поддерживает виртуальных пользователей.
Только до 8 символов распознаются в пароле для виртуальных пользователей.

Возвращается статус для всех функциональных значений ответа в response.h
O ok
1 плохой
2 ошибка
3 ошибка соединения

```
<?php
dl("php3_vmailmgr.so"); //load the shared library
$vdomain="vdomain.com";
$basepwd="password";
?>
```

---

**vm_adduser**
Добавляет нового виртуального пользователя с паролем.

**Синтаксис :**
int vm_adduser(string vdomain, string basepwd, string newusername, string newuserpassword);

Добавляет нового виртуального пользователя с паролем. newusername - это имя почтового login-a и newuserpassword - это пароль для это пользователя.

---

**vm_addalias**
Добавляет новый псевдоним для виртуального пользователя.

**Синтаксис :**
int vm_addalias(string vdomain, string basepwd, string username, string alias);

Добавляет псевдоним виртуальному пользователю. username - это имя почтового login-a и alias - это псевдоним для этого пользователя.

---

**vm_passwd**
Изменяет пароль виртуальных пользователей.

**Синтаксис :**
int vm_passwd(string vdomain, string username, string password, string newpassword);

Изменяет пароль виртуальных пользователей. username - это имя почтового login-a, password - старый пароль пользователя, и newpassword - новый пароль.

---

**vm_delalias**
Удаляет всевдоним.

**Синтаксис :**
int vm_delalias(string vdomain, string basepwd, string alias);

---

**vm_deluser**
Удаляет псевдоним виртуального пользователя.

**Синтаксис :**
int vm_deluser (string vdomain, string username);

---

## Сетевые функции

**ip2long**
Производит конвертацию строки адреса IPv4 в число.

**Синтаксис :**
int ip2long(string ip_address);

Функция **ip2long()** возвращает четырехбайтовое численное представление адреса IP v4 из строки (числа, разделенные точками, например: "127.0.0.1").

```
// получить IP адрес хоста
$ip=gethostbyname("www.php.net");
echo "Следующие URL эквивалентны:<br>";
echo "http://www.php.net/, http://".$ip.
     "/, и http://".ip2long($ip)."/<br>";
```

---

**long2ip**
Производит конвертацию числа в строку адреса IP v4.

**Синтаксис :**
string long2ip(int proper_address);

Функция **long2ip()** возвращает строковое представление IP-адреса (в формате: "aaa.bbb.ccc.ddd") из численного представления.

---

**gethostbyaddr**
Возвращает имя хоста, который соответствует заданному IP-адресу.

**Синтаксис :**
string gethostbyaddr(string ip_address);

Функция **gethostbyaddr()** возвращает доменное имя хоста, заданного своим IP-адресом. В аргументе указывается адрес IP в строковом формате. В случае ошибки возвращает *ip_address*.
Надо отметить, что функция не гарантирует, что полученное имя на самом деле будет соответствовать действительности. Она лишь опрашивает хост по адресу *ip_address* и просит его сообщить свое имя. Владелец хоста, таким образом, может передавать все, что ему заблагорассудится.

```
echo gethostbyaddr("127.0.0.1");
```

---

**gethostbyname**
Возвращает IP-адрес хоста.

**Синтаксис :**
string gethostbyname(string hostname);

Функция **gethostbyname()** получает в параметрах доменное имя хоста и возвращает его IP-адрес. Если адрес определить не удалось, функция возвращает *hostname*.

---

**gethostbynamel**
Возвращает список IP-адресов хоста.

**Синтаксис :**
array gethostbynamel(string hostname);

Одному доменному имени может соответствовать сразу несколько IP-адресов, и в случае сильной загруженности серверов DNS-сервер сам выбирает, по какому IP-адресу перенаправить запрос. Он выбирает тот адрес, который использовался наиболее редко.
Функция **gethostbynamel()** возвращает не один, а все IP-адреса хоста с именем

*hostname*.

Стоит заметить, что в Интернете существует множество виртуальных хостов, которые имеют различные доменные имена, но один и тот же IP-адресс. Таким образом, если следующая последовательность команд для существующего хоста с IP-адресом *ip* всегда печатает этот же адрес:

```
$host = gethostbyaddr($ip);
echo gethostbyname($host);
```

то аналогичная последовательность для домена с DNS-именем *$host*, наоборот, может напечатать не то же имя, а другое:

```
$ip = gethostbyname($host);
echo gethostbyaddr($ip);
```

---

**getprotobyname**
Производит определение номера порта, используемого протоколом.

**Синтаксис :**
int getprotobyname(string name);

---

**getprotobynumber**
Производит определение протокола порта.

**Синтаксис :**
string getprotobynumber(int number);

---

**getservbyname**
Производит определение протокола интернет-службы.

**Синтаксис :**
int getservbyname(string service, string protocol);

Эта функция возвращает номер порта, который использует служба **service**.
В аргументе **protocol** указывается тип протокола - TCP или UDP.

```
echo getservbyname("HTTP", "TCP"); // может вывести 80
```

---

**getservbyport**
Производит определение интернет-службы, которая использует заданный порт.

**Синтаксис :**
string getservbyport(int port, string protocol);

Здесь в аргументе **protocol** нужно указать тип протокола - TCP либо UDP.

```
echo getservbyport(21, "TCP"); // выведет: ftp
echo getservbyport(23, "TCP"); // выведет: telnet
```

---

**checkdnsrr**
Производит проверку записи DNS.

**Синтаксис :**
int checkdnsrr(string host [, string type]);

Эта функция отправляет запрос DNS-серверу для поиска записей, которые имеются для хоста **host**. Если были найдены записи типа *type*, то функция возвращает true. В противном случае и при ошибке - false.

Аргумент *type*может принимать значения:

- A

- MX (по умолчанию)
- NS
- SOA
- PTR
- CNAME
- ANY

Аргумент *host* может указываться строкой в формате IP с разделением точками, либо быть именем хоста.

---

**getmxrr**
Производит получение MX записи для интернет-хоста.

**Синтаксис :**
int getmxrr(string hostname, array mxhosts [, array weight]);

Функция **getmxrr()** инициирует поиск в базе данных DNS записи MX (почтовый сервер домена) для хоста *hostname*.
Если запись найдена, возвращает true, если нет - то false.

Список записей MX заносится в массив *mxhosts*. Если указан массив *weight*, он заполняется дополнительной информацией о записях.

---

## Отслеживание и обработка ошибок : Введение

PHP имеет следующие типы ошибок и предупреждений:

| Значение | Константа | Описание |
|---|---|---|
| 1 | E_ERROR | Фатальная ошибка времени исполнения. |
| 2 | E_WARNING | Предупреждение времени исполнения. |
| 4 | E_PARSE | Сообщение интерпретации времени исполнения. |
| 8 | E_NOTICE | Простое сообщение времени исполнения. |
| 16 | E_CORE_ERROR | Фатальная ошибка при инициализации PHP. |
| 32 | E_CORE_WARNING | Предупреждение инициализации. |
| 64 | E_COMPILE_ERROR | Фатальная ошибка компиляции. |
| 128 | E_COMPILE_WARNING | Предупреждение компиляции. |
| 256 | E_USER_ERROR | Ошибки, определяемые пользователем. |
| 512 | E_USER_WARNING | Предупреждения, определяемые пользователем. |
| 1024 | E_USER_NOTICE | Сообщения, определяемые пользователем. |
| 2047 | E_ALL | Все перечисленные сообщения. |

Указанные значения в виде чисел или констант можно комбинировать, формируя битовую маску ошибок, о которых необходимо сообщать в ходе исполнения сценария. Для комбинирования используются битовые операторы, но в конфигурационном файле php.ini распознаются только "|", "~", "!" и "&".

В PHP 4 по умолчанию разрешены сообщения вида E_ALL & ~E_NOTICE, то есть

сообщаться должно все, кроме обычных сообщений. Можно переопределить эту установку параметром файла конфигурации **error_reporting()** (ее также можно указывать в файлах конфигурации сервера Apache).

Если при вызове функции перед ее именем указать символ "@", то в случае возникновения ошибки в этой функции сообщение о нем выдаваться не будет.

В настоящее время оператор игнорирования ошибок блокирует даже выдачу сообщений о критических ошибках, при возникновении которых сценарий досрочно завершается.

Если разрешен параметр конфигурации track_errors, то сообщение об ошибке сохраняется в глобальной переменной $php_errormsg.

```
<?
// определенный пользователем обработчик ошибок
function userErrorHandler($errno,$errmsg,$filename,$linenum,$vars) {
  // время возникновения ошибки
  $dt=date("Y-m-d H:i:s (T)");
  $errortype = array(
    1    => "Error",
    2    => "Warning",
    4    => "Parsing Error",
    8    => "Notice",
    16   => "Core Error",
    32   => "Core Warning",
    64   => "Compile Error",
    128  => "Compile Warning",
    256  => "User Error",
    512  => "User Warning",
    1024 => "User Notice"
    );

  $err.="время ($dt), номер ошибки ($errno), ";
  $err.="тип ошибки (".$errortype[$errno]."): ";
  $err.="\"$errmsg\".файл \"$filename\", строка (";
  $err.=$linenum.")\n";

  $user_errors=array(E_USER_ERROR, E_USER_WARNING, E_USER_NOTICE);
  if(in_array($errno, $user_errors))
  // выдать сообщение для ошибок пользователя
  echo $err;

  // сохранить событие ошибки в системном журнале
  error_log($err, 3, "/usr/local/php4/error.log");
}

// установить уровень контроля ошибок и обработчик
error_reporting(0); // не выводить сообщения PHP
$old_error_handler=set_error_handler("userErrorHAndler");

// неопределенная константа вызывает предупреждение
$t=_NOT_DEFINED_CONSTANT;

trigger_error("Моя ошибка", E_USER_ERROR);
trigger_error("Мое предупреждение", E_USER_WARNING);

?>
```

---

## Отслеживание и обработка ошибок : Функции обработки ошибок

**error_log**
Посылка сообщения об ошибке.

**Синтаксис :**
int error_log(string message, int message_type [, string destination [, string extra_headers]])

Сообщение, посылаемое этой функцией, может быть направлено в журнал системных сообщений web-сервера, прот TCP или в файл.
В первом аргументе *message* указывается само содержание сообщения. Во втором аргументе *message_type* - куда оно должно быть направлено.
Назначение обозначается следующими значениями:

- 0 - Сообщение заносится в системный журнал событий (файл) согласно установке параметра конфигурации **error_log**.
- 1 - Сообщение отправляется по электронной почте, по адресу, указанному в аргументе *destination*. Это единственный тип сообщения, использующий четвертый параметр *extra_headers*, в котором можно указать дополнительные заголовки (как в функции **mail()**).
- 2 - Сообщение посылается через подключение отладки. Это возможно только в случае, если параметр удаленной отладки был разрешен в файле конфигурации. Для этого также должен быть определен адрес хоста (имя или его IP адрес) и порт сокета, который будет принимать сообщения отладки. Это можно указать в аргументе *destination* или параметрах конфигурации.
- 3 - *message* добавляется в конец файла *destination*.

```
if(!Ora_London($username, $password)) {
  error_log("Сервер Oracle недоступен!", 0);
};

if(!($foo = allocate_new_foo()) {
  error_log("Нельзя выделить FOO!", 1, "operator@mydomain.ru");
}

// other ways of calling error_log():
error_log("У нас ошибка!", 2, "127.0.0.1:7000");
error_log("У нас ошибка!", 2, "localhost");
error_log("У нас ошибка!", 3, "/var/tmp/my-errors.log");
```

---

**error_reporting**
Установка видов сообщаемых ошибок.

**Синтаксис :**
int error_reporting([int level])

Функция **error_reporting()** возвращает предыдущую установку типа сообщаемых ошибок. Если указан аргумент, то заново переопределяет ее. В аргументе можно указывать константу, число или битовую маску. Старайтесь использовать константы вместо численных значений, чтобы сохранить совместимость с будущими версиями PHP.

```
error_reporting(2039); // в PHP эквивалент E_ALL ^ E_NOTICE
error_reporting(E_ALL ^ E_NOTICE); // установка по умолчанию
error_reporting(0); // отключить сообщения об ошибках
// общие ошибки выполнения
error_reporting(E_ERROR | E_WARNING | E_PARSE);
// также сообщать о неизвестных переменных
error_reporting(E_ERROR | E_WARNING | E_PARSE | E_NOTICE);
error_reporting(E_ALL); // сообщать все ошибки
```

---

**restore_error_handler**
Восстановление предыдущего обработчика ошибок.

**Синтаксис :**
void restore_error_handler()

Эта функция устанавливает в качестве функции обработчика ошибок ту, которая была таковой до последнего вызова функции **set_error_handler()**. Предыдущим обработчиком может быть ранее установленный пользовательский обработчик или встроенный обработчик PHP.

---

**trigger_error**
Генерация ошибки.

**Синтаксис :**
void trigger_error(string error_msg [, int error_type])

Явно вызывает функцию, установленную для обработки ошибок, и обычно используется в паре с обработчиком ошибок. Функция способна генерировать только пользовательские типы ошибок ( семейство констант E_USER), и по умолчанию, если не указан тип ошибки *error_type*, он считается E_USER_NOTICE.

Возможно конструировать сложные конструкции генерации и обработки ошибок и исключительных ситуаций.

```
if(assert($divisor == 0))
  trigger_error ("Нельзя делить на 0 ", E_USER_ERROR);
```

---

**user_error**
Синоним функции **trigger_error()**.

**Синтаксис :**
void user_error(string error_msg [, int error_type])

---

## Отслеживание и обработка ошибок : Установка пользовательского обработчика ошибок

**set_error_handler**
Установка пользовательского обработчика ошибок.

**Синтаксис :**
string set_error_handler(string error_handler)

Функция возвращает имя функции, ранее определенной в качестве обработчика ошибок (или FALSE при ошибке), и устанавливает, в качестве нового обработчика, функцию с указанным в аргументе *error_handler* именем.

Обычно пользовательский обработчик ошибок работает в паре с функцией **trigger_error()**, генерирующей ошибку. Это может быть использовано (подобно аналогичной конструкции работы с исключениями в C) для освобождения выделенных ресурсов (например, удаления созданных файлов), если сценарий не может нормально завершиться.

Функция, устанавливаемая в качестве обработчика ошибок, должна принимать пять параметров (три последних являются дополнительными и могут не обрабатываться):

- код ошибки
- строку, описывающую ошибку
- имя сценария, в котором произошла ошибка
- номер строки сценария, содержащей ошибку
- контекст (массив, содержащий значения переменных, в момент возникновения ошибки)

```
<?
// определить константы пользовательских ошибок
define(FATAL, E_USER_ERROR);
define(ERROR, E_USER_WARNING);
define(WARNING, E_USER_NOTICE);

// установить, какие ошибки должны обрабатываться в сценарии
error_reporting (FATAL | ERROR | WARNING);

// пользовательский обработчик ошибок
function myErrorHandler($errno,$errstr,$errfile,$errline) {
  switch ($errno) {
  case FATAL:
    echo "<b>Критическая ошибка</b> [$errno] $errstr<br>\n";
    echo "в строке: $errline файла:".$errfile;
    echo ", PHP ".PHP_VERSION." (".PHP_OS.")<br>\n";
    echo "Aborting...<br>\n";
    exit -1;
    break;
  case ERROR:
    echo "<b>Ошибка</b> [$errno] $errstr<br>\n";
    break;
  case WARNING:
    echo "<b>Предупреждение</b> [$errno] $errstr<br>\n";
    break;
  default:
    echo "Неизвестный тип ошибки: [$errno] $errstr<br>\n";
  }
}

// функция для проверки обработки ошибок
// (масштабирование массива
function scale_by_log($vect, $scale) {
  if(!is_numeric($scale) || $scale <= 0)
    trigger_error("вычислить log(x) для x <= 0 нельзя. ",
                  "(x = $scale)", FATAL);
  if(!is_array($vect)) {
    trigger_error("Требуется массив ", ERROR);
    return null;
  }
  for($i=0; $i<count($vect); $i++) {
    if(!is_numeric($vect[$i]))
      trigger_error("Элемент ($i) не число и
                     его значением считается 0", WARNING);
    $temp[$i]=log($scale)*$vect[$i];
  }
  return $temp;
}

// установить пользовательский обработчик ошибок
$old_error_handler=set_error_handler("myErrorHandler");

$a=array(2,3,"foo",5.5,43.3,21.11);
print_r($a);

$b=scale_by_log($a,M_PI); // здесь выдается предупреждение
echo "Массив, масштабированный на логарифм(Пи): ";
```

```
print_r($b);

$c=scale_by_log("not array",2,3); // здесь ошибка
var_dump($c);

$d=scale_by_log($a, -2.5); // здесь критическая ошибка

echo "Продолжение сценария...";
?>
```

При выполнении сценария вывод будет следующим:

```
Array
(
  [0] => 2
  [1] => 3
  [2] => foo
  [3] => 5.5
  [4] => 43.3
  [5] => 21.11
)
<b>Предупреждение</b> [1024] Элемент (2) не число,
  и его значением считается 0<br>
Массив, масштабированный на логарифм(Пи): Array
(
  [0] => 2.2894597716988
  [1] => 3.4341896575482
  [2] => 0
  [3] => 6.2960143721717
  [4] => 49.566804057279
  [5] => 24.165247890281
)
<b>Ошибка</b> [512] Требуется массив <br>
NULL
<b>Критическая ошибка</b> [256] вычислить log(x) для x <=0 нельзя,
  (x = -2.5)<br>
  в строке: 37, файла E:\www\exampl.php, PHP 4.0.5 (WINNT)<br>
Aborting...<br>
```

Не забывайте, что при установке пользовательского обработчика ошибок стандартный обработчик PHP не используется. Установки **error_reporting()** также не будут иметь эффекта, и пользовательский обработчик должен уметь обрабатывать все виды ошибок (значение **error_reporting()** можно выяснить и действовать соответсвенно). Заметьте, что код ошибки будет равен 0, если ошибка возникла в функции, вывод ошибок для которой был блокирован оператором "@".

Также помните, что завершать сценарий в обработчике необходимо явно (например, с помощью функции **die()**), если, конечно, в этом есть необходимость. Если обработчик ошибок завершается с помощью return, то выполнение сценария продолжается с того места, в котором возникла ошибка (то есть исполняются инструкции, которые следуют за той инструкцией, в которой возникла ошибка).

---

# Управление сессиями : Зачем нужны сессии.Механизм работы сессий

### Зачем нужны сессии

Сессия представляет собой механизм, позволяющий хранить некоторые данные, индивидуальные для каждого пользователя (например, его имя и номер счета), между запусками сценария.

В Web-программировании есть один класс задач, который может вызвать довольно много проблем, если писать сценарий "в лоб". Речь идет о слабой стороне CGI - невозможности запустить программу на длительное время, позволив ей при этом обмениваться данными с пользователями.

Представте, что мы пишем форму, но в ней такое большое число полей, что было бы глупо поместить их на одну страницу. Нам нужно разбить процесс заполнения формы на несколько этапов, или стадий, и представить их в виде отдельных HTML-документов.
Например, в первом документе с диалогом у пользователя может запрашиваться его имя и фамилия, во втором - данные о его месте жительства, и в третьем - номер кредитной карточки. В любой момент можно вернуться на шаг назад, чтобы исправить те или иные данные. Наконец, если все в порядке, накопленная информация обрабатывается - например, помещается в базу данных.

Реализация такой схемы оказывается для Web-приложений довольно нетривиальной проблемой. Действительно, нам придется хранить все ранее введенные данные в каком-нибудь хранилище, которое должно аннулироваться, если пользователь вдруг передумает и уйдет с сайта. Для этого можно использовать функции сериализации и файлы. Однако ими мы решаем только половину проблемы: нам нужно как то привязать конкретного пользователя к конкретному временному хранилищу. Действительно, предположим, мы этого не сделали. Тогда, если в момент заполнения какой-нибудь формы одним пользователем на сайт зайдет другой и тоже попытается ввести свои данные, получится белеберда.
Все эти проблемы решаются при помощи сессий.

**Механизм работы сессий**

Для начала должен существовать механизм, который бы позволил PHP идентифицировать каждого пользователя, запустившего сценарий. То есть при следующем запуске PHP нужно однозначно определить, кто его запустил: тот же человек, или другой. Делается это путем присвоения клиенту так называемого уникального *идентификатора сессии*. Чтобы этот идентификатор был доступен при каждом запуске сценария, PHP помещает его Cookies браузера. Теперь, зная идентификатор (дальше SID), PHP может определить, в каком же файле на диске храняться данные пользователя.

Немного о том, как сохранять переменную (обязательно глобальную) в сессии. Для этого мы должны ее зарегестрировать с помощью специальной функции. После регистрации мы можем быть уверены, что при следующем запуске сценария тем же пользователем она получит то же самое значение, которое было у нее при предыдущем завершении программы. Это произойдет потому, что при завершении сценария PHP автоматически сохраняет все переменные, зарегестрированные в сессии, во временное хранилище. Конечно, можно в любой момент аннулировать переменную - вычеркнуть ее из сессии, или же уничтожить вообще все данные сессии.

Где же находиться то промежуточное хранилище, которое использует PHP? Вообще говоря, вы вольны сами это задать, написав соответствующие функции и зарегестрировав их как обработчики сессии. Впрочем, делать это не обязательно: в PHP уже существуют обработчики по умолчанию, которые хранят данные в файлах. Если вы не собираетесь создавать что-то особенное, вам они вполне подойдут.

## Управление сессиями : Инициализация сессии и регистрация переменных

**session_start**
Эта функция инициализирует механизм сессий для текущего пользователя, запустившего сценарий.

**Синтаксис :**
void session_start()

- Если посетитель запускает программу впервые, у него устанавливается Cookies с уникальным идентификатором, и создается временное хранилище, ассоциированное с этим идентификатором.
- Определяется, какое хранилище связано с текущим идентификатором пользователя.
- Если в хранилище имеются какие-то переменные, их значения восстанавливаются. Точнее, создаются глобальные переменные, которые были сохранены в сессии при предыдущем завершении сценария.

Надо заметить, что если вы поставили в настройках PHP режим *session.auto_start=1*, то функция инициализации вызывается автоматически при запуске сценария. Так же надо следить за тем, чтобы до нашей функции не было никакого вывода в браузер - иначе PHP не сможет установить SID для пользователя.
Функция всегда возвращает true.

---

**session_register**
Указывает PHP на то, что ту или иную переменную нужно сохранить в сессии.

**Синтаксис :**
bool session_register(mixed name [, mixed name1, ...])

Функция принимает в параметрах одно или несколько имен переменных (имена задаются в скобках, без знака $ слева), регистрируют их в текущей запущенной сессии и возвращает true, если регистрация прошла успешно.

Повторная запись одной переменной в сессии не приведет к ошибке.

```
<?
session_start();
session_register("count");
$count=@$count+1;
?>
<h2>Счетчик</h2>
В текущей сессии работы с браузером вы открыли эту страницу
<?=$count?> раз(а). Закройте браузер, чтобы обнулить счетчик.
</body>
```

---

## Управление сессиями : Имя группы сессии

Надо отметить, что на одном и том же сайте могут существовать сразу несколько сценариев, которые нуждаются в услугах поддержки сессий PHP. Они "ничего не знают" друг о друге, поэтому временные хранилища для сессий должны выбираться не только на основе идентификатора пользователя, но и на основе того, какой из сценариев запросил обслуживание сессии.
Для наглядности рассмотрим пример:

Пусть разработчик А написал сценарий счетчика. Он использует переменную $count, и не имеет никаких проблем. До тех пор, пока разработчик В, ничего не знающий о сценарии А, не создал систему статистики, которая тоже использует сессии. Самое ужасное, что он также регистрирует переменную $count, не зная о том, что она уже занята. В результате, как всегда, страдает пользователь: запустив сначало сценарий разработчика В, а потом - А, он видит, что данные счетчиков перемешались.

Нам нужно как-то разграничить сессии, принадлежащие одному сценарию, от сессии, принадлежащих другому. К счастью, разработчики PHP предусмотрели такое положение вещей. Мы можем давать *группам сессии* непересекающиеся имена, и сценарий, знающий имя своей группы сессии, сможет получить к ней доступ. Вот теперь-то разработчики А и В могут оградить свои сценарии от проблем с пересечением имен переменных. Достаточно в первой программе указать PHP, что мы хотим использовать группу с именем, например, sesA, а во второй - sesB.

---

**session_name**
Эта функция устанавливает или возвращает имя группы сессии, которая будет использоваться PHP для хранения зарегестрированных переменных.

**Синтаксис :**
string session_name([string $newname])

Если $newname не задан, то возвращается текущее имя. Если же этот параметр указан, то имя группы будет изменено на $newname, при этом функция вернет предыдущее имя.

Отметим, что session_name() лишь сменяет имя текущей группы и сессии, но не создает новую сессию и временное хранилище. Это значит, что мы должны в большинстве случаев вызывать session_name(имя_группы) еще до ее инициализации - вызова **session_start()**, в противном случае мы получим совсем не то, что ожидали.

Если функция session_name() не была вызвана по инициализации, PHP будет использовать имя по умолчанию - **PHPSESID**.
**Пример:**
```
<?
session_name("CounterScript"
session_start();
session_register("count");
$count=@$count+1;
?>
В текущей сессии вы открыли эту страницу <?=$count?> раз(а).
```

---

## Управление сессиями : Идентификатор сессии

Итак, идентификатор сессии является именем временного хранилища, которое будет использовано для хранения данных сессии между запусками сценария. Один SID - одно хранилище. Нет SID, нет и хранилища, и наоботот.
так как же соотностится идентификатор и имя группы? Имя - это всего лишь собирательное название для нескольких сессий (то есть, для многих SID), запущенных разными пользователями. Один и тот же клиент никогда не будет иметь два различных SID в пределах одного имени группы. Но его браузер вполне может работать с несколькими SID, расположенными логически в разных

"пространствах имен".

Итак, все SID уникальны и однозначно определяют сессию на компьютере, выполняющем сценарий - независимо от имени сессии. Имя же задает пространство имен, в которое будут сгруппированны сессии, запущенные разными пользователями. Один клиент может иметь сразу несколько активных пространств имен (то есть несколько имен групп сессий).

---

**session_id**
Эта функция возвращает текущий идентификатор сессии SID.

**Синтаксис :**
string session_id([string $sid])

Если задан параметр $sid, то у активной сессии изменяется идентификатор на $sid.
Вызвав session_id() до **session_start()**, мы можем подключиться к любой (в том числе и к чужой) сессии на сервере, если знаем ее идентификатор. Мы можем также создать сессию с угодным нам идентификатором, при этом автоматически установив его в Cookies пользователя.

---

## Управление сессиями : Другие функции

**session_is_registered**
Проверяет, зарегестрирована или нет та или иная переменная.

**Синтаксис :**
bool session_is_registered(string $name)

Функция возвращает true, если переменная с именем $name была зарегестрирована в сессии, иначе возвращает false.

---

**session_unregister**
Отменяет регистрацию переменной.

**Синтаксис :**
bool session_unregister(string $name)

Эта функция отменяет регистрацию для переменной с именем $name для текущей сессии. Или иначе, при завершении сценария все будет выглядеть так, словно переменная с именем $name и небыла никогда зарегестрирована.
Возвращает true, если все прошло успешно, и false - в противном случае.
Отметим, что после вызова функции session_unregister() глобальная переменная, которая была "аннулирована", не уничтожается, а сохраняет свое значение.

---

**session_unset**
Отменяет регистрацию и уничтожает глобальные переменные.

**Синтаксис :**
void session_unset()

Эта функция, в отличии от **session_unregister()**, не только отменяет регистрацию переменных (всех переменных сессии, а не только какой-то одной), но и уничтожает глобальные переменные, которые были зарегестрированы в сессии.

**session_save_path**
Имя каталога, в котором будут храниться файлы с данными из сессий.

**Синтаксис :**
string session_save_path([string $path])

Эта функция возвращает имя каталога, в котором будут помещаться файлы - временные хранилища данных сессии. В случае, если указан параметр, активное имя каталога будет переустановлено на $path. При этом функция вернет предыдущий каталог.

## Управление сессиями : Обзор обработчиков

**handler_open**
Этот обработчик должен взять на себя всю работу по открытию базы данных для группы сессий с именем, которое было передано ей в параметрах.

**Синтаксис :**
bool handler_open(string $save_path, string $session_name)

Функция вызывается, когда вызывается **session_start()**. Обработчик должен взять на себя всю работу, связанную с открытием базы данных для группы сессий с именем $session_name. В параметре $save_path передается то, что было указано при вызове **session_save_path()** или же путь к файлам-хранилищам данных сессий по умолчанию.

**handler_close**
Этот обработчик вызывается, когда данные сессии записаны во временное хранилище и его нужно закрыть.

**Синтаксис :**
bool handler_close()

**handler_read**
Чтение данных сессии.

**Синтаксис :**
string handler_read(string $sid)

Этот обработчик вызывают, когда нужно прочитать данные сессии с идентификатором $sid из временного хранилища.
Возвращаемые данные представлены в следующем виде:

имя1=значение1;имя2=значение2;имя3=значение3;...

имяN задает имя очередной переменной, зарегестрированной в сессии, а занчениеN - результат вызова функции Serialize() для значения этой переменной. Например, наша запись может иметь следующий вид:

foo|i:1;count|i:10;

Она говорит о том, что из временного хранилища были прочитаны две целые переменные, первая из которых равна 1, а вторая - 10.

**handler_write**
Запись данных сессии.

**Синтаксис :**
string handler_write(string $sid, string $data)

Этот обработчик предназначен для записи данных сессии с идентификатором $sid во временное хранилище - например, открытое ранее обработчиком **handler_open()**. Параметр $data задается в точно таком же формате. Фактически, чаще всего действие этой функции сводится к записи в базу данных строки $data без каких-либо ее изменений.

---

**handler_gc**
Очищает временное хранилище данных через определенный промежуток времени.

**Синтаксис :**
bool handler_gc(int $maxlifetime)

Этот обработчик вызывается каждый раз при завершении работы сценария. Если пользователь окончательно покинул сервер, заначит, данные сессии во временном хранилище можно уничтожить. Этим и должна заниматься функция **handler_gc()**. Ей передается в параметрах то время (в секундах), по прошествии которого PHP принимает решение о необходимости удалить все ненужные данные.

---

**session_set_save_handler**
Регистрация обработчиков.

При описании обработчиков мы указывали их имена с префиксом handler. На самом деле, это совсем не является обязательным. Даже наоборот - вы можете давать такие имена своим обработчикам, какие только захотите.

Но возникает вопрос: как же тогда PHP их найдет? Вот для этого и существует функция регистрации обработчиков, которая говорит интерпретатору, какую функцию он должен вызывать при наступлении того или иного события.

**Синтаксис :**
void session_set_save_handler($open, $close, $read, $write, $destroy, $gc)

Эта функция регестрирует подпрограммы, имена которых переданы в ее параметрах, как обработчики текущих сессии. Параметр $open содержит имя функции, которая будет вызвана при инициализации сессии, а $close - функции, вызываемой при ее закрытии. В $read и $write нужно указать имена обработчиков, соответственно, для чтения и записи во временное хранилище. Функция с именем, заданным в $destroy, будет вызвана при уничтожении сессии. Наконец, обработчик, определяемый параметром $gc, используется как обработчик мусора.

Эту функцию можно вызывать только до инициализации сессии, в противном случае она просто игнорируется.

---

## Управление сессиями : Про сессии и Cookies

**Проблема: - отключены Cookies**
Бытует распространенное мнение, что сессии без Cookies не может существовать. Действительно, Cookies наиболее просто решает проблему идентификации пользователя, что необходимо для связи временного хранилища и данных сессии. Но вот что делать, если пользователь у себя в настройках отключил прием Cookies?
На этот случай разработчики PHP позаботились о передаче идентификаторов сессии не в Cookies, а каким-нибудь аналогичным путем, например через адресную строку броузера.
**Решение: - изменение гиперссылок и форм**
В PHP существует одна специальная константа с именем SID. Она всегда содержит имя группы сессии и ее идентификатор в формате *имя=идентификатор*. Именно в таком формате данные принимаются, когда они приходят из Cookies браузера. Таким образом, нам достаточно просто передать значение константы SID в сценарий, чтобы он "подумал", будто бы данные пришли из Cookies.
Вот пример использования сессий без Cookies:

```
<?
session_name("testses");
session_start();
session_register("i");
$i=@$i+1;
?>
<body>
Вы открыли эту страницу
<?=$i?> раз. При закрытии браузера счетчик обнулиться.<BR>
<A href=sesclick.php?<?=SID?>>Нажмите для записи в счетчик!</A>
</body>
```

Этот пример будет работать, если у пользователя действительно отключены Cookies. Если они включены, PHP просто не будет генерировать константу SID и задействует Cookies.

Но в приведенном способе есть одно неудобство, а именно, везде в участки кода нужно вставлять <?=SID?>, и если вы гдето пропустили, то программа может не работать!
К счастью, разработчики PHP учли эту возможность, и решили уберечь нас от этого. По-этому если в какой-нибудь гиперссылке вы по ошибке пропустите <?=SID?>, PHP вставит его автоматически. При этом не повредив остальные параметры, которые могут присутствовать в URL.

Для проверки можно использовать следующий пример:

```
<?session_start()?>
<body>
<A href="php.php">PHP</A>
<A href="php.php?ss=1">PHP - выражения</A>
```

Вот что получиться, при наведении мышки на эти ссылки:

http://www.spravkaweb.ru/php.php?PHPSESSID=81456f6a886f2104
http://www.spravkaweb.ru/php.php?ss=1&PHPSESSID=34f5d04a35601510f45

В PHP существует еще одна возможность использовать сессии с отключенными Cookies - добавление скрытх полей в формы, которые формируют сценарий, чтобы передать идентификатор сессии вызываемому документу.
Приведем пример, который выявляет эту возможность:

```
<?session_start()?>
<form action=act.php mathod=post>
</form>
```

А вот что получиться при просмотре нашей страницы в виде HTML:

```
<form action="act.php" method="post">
<INPUT TYPE=HIDDEN NAME="PHPSESSID" VALUE="0a561093f84d4321">
```

```
</form>
```
Из примера мы видим, что PHP добавил в форму скрытое поле с нужным именем и занчением.

---

## Работа с WWW : Установка заголовков ответа

**Header**
Вывод заголовка.

**Синтаксис :**
int Header(string $string)

Обычно функция Header() является одной из первых команд сценария. Она предназначена для установки заголовков ответа, которые будут переданы браузеру - по одному заголовку на вызов. Вызов Header() обязательно должен осуществляться до любого оператора вывода в сценарии - в противном случае вы получете предупреждение. Текст вне <? и ?> также рассматривается как оператор вывода.

**Пример:**

```
// перенаправляет браузер на сайт PHP
Header("Location: http://www.php.net");
// теперь принудительно завершаем сценарий, ввиду того, что после
// перенаправления больше делать нечего
exit;
```

См. также:
Запрет кэширования страниц при помощи Header().

---

## Работа с WWW : Получение заголовков запроса

**getallheaders**
Получение всех заголовков запроса.

**Синтаксис :**
array GetAllHeaders()

Функция GetAllHeaders() возвращает ассоцативный массив, содержащий данные о HTTP-заголовках запроса клиента, породившего запуск сценария. Ключи массива содержат названия заголовков, а значения - их величины.

```
$headers = GetAllHeaders();
foreach($headers as $header=>$value)
echo "$header: $value<br>\n";
```

Функция GetAllHeaders() поддерживается PHP только в том случае, если он установлен в виде модуля Apache. В противном случае этой функции просто не будет (да и не может быть, потому что обычный CGI-сценарий не имеет доступа к загаловкам запроса). В частности, в PHP для Windows (который чаще всего реализуют именно в виде сценария) функция GetAllHeaders() недоступна.

---

# Работа с WWW : Работа с Cookies

### Немного теории

Cookie - это именованная порция информации, которая может сохраняться прямо в настройках браузера пользователя между сеансами. Причина, по которой применяются Cookies - большое количество посетителей вашего сервера, а также нежелание иметь нечто подобное базе данных для хранения информации о каждом посетителе. Поиск в такой базе данных может очень и очень затянуться, и, в тоже время, нет никакого смысла централизованно хранить столь отрывочные сведения. Использование Cookies фактически перекладывает задачу на плечи браузера, решая одним махом как проблему быстродействия, так и проблему большого объема базы данных с информацией о пользователе.

Самый распространенный прием применения Cookies - логин и пароль пользователя, использующего некотовые защищенные ресурсы вашего сайта. Эти данные, конечно же, между открытиями страниц храняться в Cookies, для того чтобы пользователю не пришлось их каждый раз набирать вручную заново.

У каждого Cookies есть определенное время жизни, по истечении которого он автоматически уничтожается. Существуют также Cookies, которые "живут" только в течение текущего сеанса работы с браузером.

Каждый Cookie устанавливается сценарием на сервере. Для этого он должен послать браузеру специальный заголовок вида:

Set-cookie: данные

Сценарии с других серверов, а также расположенные в другом каталоге, не будут извещены о "чужих" Cookies. Для них их словно и нет. И это правильно с точки зрения безопасности - кто знает, насколько секретна может быть информация, сохраненная в Cookies?

---

### Получение Cookie

Предположим, сценарий отработал и установил какой-то Cookie, например, с именем Cook и значением Val. В следующий раз при запуске этого сценария (на самом деле, и всех других сценариев, расположенных на том же сервере в том же каталоге или ниже по дереву) ему передастся пара типа Cook=Val (через специальную переменную окружения). PHP это событие перехватит и автоматически создаст переменную $Cook со значением Val. То есть интерпретатор действует точно так же, как если бы значение нашего Cookie пришло откуда-то из формы. Та переменная, которую мы установили в прошлый раз, будет доступна и сейчас.

---

**setcookie**
Установка Cookie.

**Синтаксис :**
int setcookie(string $name [,string $value] [,int $expire] [,string $path] [,string $domain] [,book $secure])

Так как Cookie фактически представляет собой заголовок, установить его можно только перед первой командой вывода в сценарий.

Вызов setcookie() определяет новый Cookie, который тут же посылается браузеру

вместе с остальными заголовками. Все аргументы, кроме имени, необязательны. Если задан только параметр $name (имя Cookie), то Cookie с указанным именем у пользователя удаляется. Вы можете пропускать аргументы, которые не хотите задавать, пустыми строками "". Аргументы $expire и $secure не могут быть представлены строками, а потому вместо пустых строк здесь нужно использовать 0.
Параметр $expire задает timestamp, который, например, может быть сформирован функциями time() или mktime().
Параметр $secure говорит о том, что величина Cookie может передаваться только через безопасное HTTPS-соединение.

**Примеры:**

```
// Cookie на одну сессию, т.е. до закрытия браузера
SetCookie("TextCookie","value");

// Эти Cookies уничтожаются браузером через 1 час после установки
SetCookie("TextCookie",$val,time()+3600);
SetCookie("TextCookie",$val,time()+3600,"/~rasmus/",".utoronto.ca",1);
```

После вызова SetCookie() только что созданный Cookie сразу появляется среди глобальных переменных как переменная с заданным в параметре $name именем. Она появиться и при следующем запуске сценария - даже если SetCookie() в нем и не будет вызвана.

---

## Работа с WWW : SSI и функция virtual()

**Немного теории**

Для одного и того же документа в Apache нельзя применять два "обработчика". Иными словами, действует принцып: либо PHP, либо SSI. Однако в PHP имеются определенные средства для "эмуляцииquot; SSI-конструкции *include virtual*.

Конструкция *include virtual* загружает файл, URL которого указан у нее в параметрах, обрабатывает его нужным обработчиком и выводит в браузер. То ести все происходит так, будто указанный URL был затребован виртуальным браузером. Большинство SSI-файлов ограничиваются использованием этой возможности.

---

**virtual**
Имитация include virtual.

**Синтаксис :**
int virtual(string $url)

Функция virtual() представляет собой процедуру, которая может поддерживаться только в случае, если PHP установлен как модуль Apache. Она делает то же самое, что и SSI-инструкция <-- *#include virtual=...* -->. Иными словами, она генерирует новый запрос серверу, обрабатываемый им обычным образом, а затем выводит данные в стандартный поток вывода.

Чаше всего функция virtual() используется для запуска внешних CGI-сценариев, написанных на другом языке программирования, или же для обработки SSI-файлов более сложной структуры. В случае, если запускается сценарий, он должен генерировать правильные HTTP-заголовки, иначе будет выведено

сообщение об ошибке. Для включения обычных PHP-файлов с участками кода функция virtual() неприменима - это выполняет оператор **include**.

---

## Управление выводом : Введение

Данная группа функций позволяет управлять тем, как PHP при выполнении сценария выводит информацию. Это может быть полезно в различных ситуациях, в особенности при посылке браузеру HTML-заголовков (headers) после того, как сценарий начал выводить HTML-текст. (В обычном случае невозможно послать заголовок после того, как был начат вывод текста.)
Эти функции не воздействуют на заголовки, посланные функциями header() или setcookie(), а только на функции, подобные echo() и HTML-тексту между блоками PHP-кода.

```
<?php

ob_start();
echo "Hello\n"

setcookie("cookiename", "cookiedata");

ob_end_flush();

?>
```

В примере выше вывод командой echo() будет сохранен в буфере вывода до вызова функции ob_end_flush(). В то же время вызов setcookie() успешно сохраняет cookie, не вызывая ошибки.

---

## Управление выводом : Функции управления выводом

**ob_start**
Включение буферизации вывода.

**Синтаксис :**
void ob_start([string output_callback])

После вызова этой функции включается буферизация вывода и, пока она активна, никакие из выводящихся данных не будут посланы браузеру, а будут сохраняться во внутреннем буфере PHP.

Содержимое буфера может быть скопировано в строковую переменную функцией ob_get_contents(). Для вывода содержимого из буфера используется функция ob_end_flush(). Удалить содержимое буфера позволяет функция ob_end_clean().

В аргументе *output_callback* можно указать функцию, которая будет автоматически вызываться при выводе содержимого буфера. Обычно это используется для модификации содержимого буфера перед выводом (например, сжатия). Тогда при вызове функции ob_end_flush() в указанную функцию будет передаваться содержимое буфера, а то, что она возвратит, будет выведено (заметьте, сама функция не должна ничего выводить).

Буферизация может быть вложенной, и тогда она обрабатывается соответственно вложенности; и содержимое, выводимое из буфера нижнего уровня, будет включаться в буфер верхнего уровня. Не забывайте, что для вывода всего буферизованного содержимого необходимо вызывать функцию ob_end_flush() столько же раз, сколько была вызвана ob_start().

```
<?php
function c($str) { // получает содержимое буфера
  return nl2br($str); // возвращает содержимое буфера
}
function d($str) { // получает содержимое буфера
  return strtoupper($str); // возвращает содержимое буфера
}
?>
<?php
ob_start("c");
?>

Тут различный текст...

<?php
// преобразовывать текст длее в верхний регистр
ob_start("d");
?>

еще что-то...

<?php
ob_end_flush();
?>

.......

<?php
ob_end_flush();
?>
```

---

**ob_get_contents**
Получение содержимого буфера вывода.

**Синтаксис :**
string ob_get_contents()

Если буферизация неактивна, возвращается false.

---

**ob_get_length**
Получение длины данных в буфере вывода.

**Синтаксис :**
string ob_get_length()

Если буферизация неактивна, возвращается false.

---

**ob_end_flush**
Вывод содержимого буфера.

**Синтаксис :**
void ob_end_flush(void)

После вывода буфер текущего уровня очищается, поэтому вызывайте функцию ob_get_contents() заранее, если необходимо получить его содержимое.

---

**flush**
Вывод всего содержимого буфера.

**Синтаксис :**

void flush(void);

Функция воздействует только на буферизацию PHP и не может контролировать схему буферизации web-сервера или браузера.

Некоторые серверы, в особенности под Win32, буферизируют выводящиеся сценарием данные до того, как сценарий завершится и данные будут отосланы браузеру.

Браузер, в свою очередь, также может буферизировать получаемые данные до их отображения. Netscape, например, буферизирует текст до получения символа завершения строки или открывающего тега, а для таблиц - до получения тега </table> таблицы верхнего уровня.

---

**ob_end_clean**
Очистка буфера.

**Синтаксис :**
void ob_end_clean(void);

Вызов функции отключает буферизацию на текущем уровне.

---

**ob_implicit_flush**
Установление режима буферизации.

**Синтаксис :**
void ob_implicit_flush([int flag]);

Если в аргументе указано ненулевое значение или оно не указано, то при осуществлении каждой операции вывода будет неявно вызыватся функция flush().

Надо отметить, что часто эта функция работает курьезно; например, если в конце сценария вызвать функцию ob_end_clean(), то сценарий не выведет ничего, если вывод из буфера не производился явно другими функциями.

---

## Управление исполнением сценария PHP : Функции управления сценарием

**set_time_limit**
Установка предельного времени исполнения сценария.

**Синтаксис :**
void set_time_limit(int seconds)

При запуске сценария PHP запускает системный таймер, и если время (выделенное сценарию для выполнения) истекает, а сценарий еще не завершился, PHP принудительно завершает сценарий (генерируя фатальную ошибку исполнения). Это не допускпет скопления большого количества сценариев, расходующих ресурсы сервера, но, повидимому, "зависших" (например, если в них обнаружился бесконечный цикл или они пытаются дождаться подключения к неотвечающему серверу).

По умолчанию допустимое время исполнения сценария устанавливается в файле конфигурации параметром *max_execution_time* (обычно оно равно 30 с). Но для

текущего сценария это время можно изменить вызовом данной функции, указав время в секундах в ее аргументе. Если указывается значение 0, то тогда временное ограничение снимается.

Отсчет времени начинается от момента вызова функции. Например, если сценарий уже выполнялся в течении 15 секунд, а затем вызывается функция **set_time_limit(20)**, то общее максимальное время исполнения сценария становится равным 35 секундам.

Если сценарий выполняется в безопасном режиме (с установленным параметром safe mode), то тогда вызов этой функции игнорируется и используется значение из файла конфигурации.

---

**sleep**
Задержка выполнения сценария.

**Синтаксис :**
void sleep(int seconds);

Фукция **sleep()** выполняет задержку выполненя сценария в секундах (seconds).

---

**usleep**
Задержка выполнения сценария в микросекундах.

**Синтаксис :**
void usleep(int micro_seconds);

Задержка выполнения сценария в микросекундах (micro_seconds).
Эта функция не работает в Windows.

---

**die**
Вывод сообщения и завершение текущего сценария.

**Синтаксис :**
void die(string message);

Эта функция выводит сообщение и прекращает выполнение текущего скрипта. Не возвращает значение.

```
<?php
$filename = '/path/to/data-file';
$file = fopen($filename, 'r')
 or die "unable to open file ($filename)";
?>
```

---

**exit**
Завершает текущий сценарий.

**Синтаксис :**
void exit(void);

Эта функция завершает текущий сценарий. Не возвращает значение.

---

**assert**
Проверка истинности значения.

**Синтаксис :**
int assert(string|bool assertion);

В качестве аргумента функции может быть указано значение или строка, содержащая код PHP (как в функции eval()). Функция проверяет, является ли значение (или выражение) равным false, и, если это так, выполняет определенные действия.

Поведение функции определяется установками в файле конфигурации или при вызове функции assert_options().

Обычно эта функция используется исключительно в целях отладки, для проверки тех значений, которые всегда должны быть истинны (например: подключение модуля, свободное пространство на диске и т.д.).

В целом же выполнение сценария не должно зависеть от таких проверок, а использовать обычные проверки возвращаемых функциями значений.

```
<?php
function handler() {
  echo "\n* Failed * \n";
}

assert("\$a='1';");
echo "a: $a \n";
assert(0);
// завершать сценарий
echo assert_options(ASSERT_BAIL, 1);
// вызвать обработчик
assert_options(ASSERT_CALLBACK, "handler");
// не выдавать сообщений PHP
@assert(--$a);
// эта строка не будет выполнена
echo "\n ... \n"
```

Приведенный пример выведет:

a: 1
Warning: Assertion failed in file.php on line 20
0
* Failed *

---

**assert_options**
Определение параметров assert.

**Синтаксис :**
mixed assert_options(int parameter [, mixed value])

Эта функция позволяет определить поведение конструкции assert().
Возвращается предыдущее значение параметра (или значение false при ошибке), указанного в первом аргументе одной из следующих констант:

| Параметр | ini-параметр | Умолчание | Описание |
|---|---|---|---|
| ASSERT_ACTIVE | asser.active | 1 | Разрешить указание кода в assert(). |
| ASSERT_WARNING | assert.warning | 1 | Выдавать предупреждение PHP. |
| ASSERT_BAIL | assert.bail | 0 | Завершать выполнение, если "неистинно". |
| ASSERT_QUIET_EVAL | assert.quiet_eval | 0 | Не выдавать сообщений. |
| ASSERT_CALLBACK | assert_callback | (null) | Установить функцию в |

| | |
|---|---|
| | качестве обработчика "неистинных" assert(). |

Если значение необходимо переопределить, его указывают во втором аргументе.

**eval**
Производит выполнение строки содержащей PHP код.

**Синтаксис :**
void eval(string code_str);

Функция **eval()** производит выполнение строки, заданной в *code_str* содержащей PHP код. Кстати, это может пригодиться для сохранения кода в текстовом поле базы данных для более позднего выполнения. Не забывайте, что указанный в строке код должен быть синтаксически правильным (например, должны присутствовать точки с запятой после каждой команды и т.п.), в противном случае сценарий будет завершен в этой строке с ошибкой. Учитывайте также, что те значения переменных, которые будут установлены в данной строке, будут использоваться в оставшейся части сценария.
При изменении переменных значений в eval() эти переменные будут изменены и в основных данных.

Если в строке указан оператор **return**, то тогда выполнение указанного кода будет досрочно завершено и возвращенное значение можно будет получить как значение, возвращаемое самой функции.

```
<?php
$string = 'cup';
$name = 'coffee';
$str = 'This is a $string with my $name in it.
';
echo $str;
eval( "\$str = \"$str\";" );
echo $str;
?>
```

Результатом выполнения этого кода будет:

This is a $string with my $name in it.
This is a cup with my coffee in it.

## Управление исполнением сценария PHP : Статус подключений

Внутренне PHP имеет три статуса подключения:
- 0 - NORMAL;
- 1 - ABORTED (прервано пользователем);
- 2 - TIMEOUT (истеклож время ожидания ответа).

При норамальном выполнении сценария активно состояние NORMAL. Если во время загрузки страницы пользователь нажал кнопку STOP, активным становится состояние ABORTED. Если сценарий выполняется дольше отведенного ему времени, устанавливается флаг состояния TIMEOUT. Возможно определить, как должен вести себя сценарий в зависимости от этих условий.

Если требуется, чтобы сценарий продолжал свое выполнение при разрыве

соединения пользователем, нужно установить в файле конфигурации значение параметра ignore_user_abort = 1 (это также можно сделать в файлах конфигурации Apache). Можно также воспользоваться функцией ignore_user_abort(). В противном случае, сценарий завершается.

Чтобы игнорировать завершение сценария таймером, необходимо использовать функцию set_time_limit().

Если функцией register_shutdown_function() была установлена функция "запускаемая при завершении сценария", то, вне зависимости от статуса подключения, она будет исполнена перед тем, как сценарий завершится. И в "завершающей" функции можно будет выяснить (с помощью функции: connection_aborted(), connection_timeout() и connection_status()), был ли сценарий завершен нормально или досрочно.

---

**connection_aborted**
Определения разрыва подключения пользователем.

**Синтаксис :**
int connection_aborted(void);

Функция **connection_aborted()** возвращает true, если подключение было разорвано пользователем.

---

**connection_status**
Определения статуса подключения.

**Синтаксис :**
int connection_status(void);

Возвращает значение битового поля, позволяющее выяснить в "завершающей" функции, был ли сценарий завершен досрочно и причину этого. Например, если возвращается 3 (ABORTED | TIMEOUT), то это означает, что время выполнения истекло, а также то, что пользователь отказался от загрузки страницы.

Если возвращается 0 (то есть значение NORMAL), то это означает, что выполнение сценария не было прервано.

---

**connection_timeout**
Определения наступления тайм-аута.

**Синтаксис :**
int connection_timeout(void);

Возвращает true, если время выполнения сценария истекло.

---

**ignore_user_abort**
Прерывание сценария при разрыве подключения.

**Синтаксис :**
int ignore_user_abort([int setting]);

Аргументом *setting* можно указать, необходимо ли досрочно завершать выполнение сценария, если связь с клиентом потеряна. Если аргумент не указан, то возвращается текущая установка.

---

**register_shutdown_function**
Устанавливает функцию, которая будет выполнена при завершении.

**Синтаксис :**
int register_shutdown_function(string func);

Регистрирует функцию с именем *func* в качестве функции, запускаемой после завершения сценария.

Заметьте: так как после завершения функции никакие средства вывода недоступны, это делает для функции, зарегистрированной в качестве "завершающей", недоступными обычные средства отладки, такие команды как print или echo.

---

## Управление исполнением сценария PHP : Дополнительные функции

**get_browser**
Определение возможностей браузера.

**Синтаксис :**
object get_browser([string user_agent]);

Возвращаемая информация извлекается из файла browscap.ini. Для определения браузера используется значение переменной $HTTP_USER_AGENT или значение, содержащееся в аргументе *user_agent*.

Информация возвращается в виде свойств объекта и отражает возможности клиентского браузера (например, версию, поддерживает ли он javascript или cookies).

```
<?php
function list_array($array) {
  while (list ($key, $val) == each ($array)) {
    $str .= "<b>$key:</b> $val<br>\n";
  }
  return $str;
}

echo "$HTTP_USER_AGENT<hr>";
$bouser = get_browser();
echo list_array ((array) $browser);
?>
```

Содержимое возможного вывода:

```
Mozilla/4.5 [en] (X11: Linux 2.2.9 i586)<hr>
<b>browser_name_pattern:</b>Mozilla/4\.5.*<br>
<b>parent:</b>Netscape<br>
<b>platform:</b>Unknown<br>
<b>majorver:</b>4<br>
<b>minorver:</b>5<br>
<b>browser:</b>Netscape<br>
<b>version:</b>4<br>
<b>frames:</b>1<br>
<b>tables:</b>1<br>
<b>cookies:</b>1<br>
<b>backgroundsounds:</b> <br>
<b>vbscript:</b> <br>
<b>javascript:</b>1<br>
<b>javaapplets:</b>1<br>
```

```
<b>activexcontrols:</b> <br>
<b>beta:</b> <br>
<b>crawler:</b> <br>
<b>authenticodeupdate:</b> <br>
<b>msn:</b> <br>
```
Для того чтобы функция могла функционировать, следует правильно указать месторасположение файла browscan.ini в файле конфигурации.

---

**highlight_file**
Вывод содержимого файла с цветовой разметкой.

**Синтаксис :**
boolean highlight_file(string filename);

Имя или путь файла указывается в аргументе. Цвета выделения синтаксиса определяются в файле конфигурации PHP. Возвращает true или false при ошибке.

Напримет, чтобы заставить сервер Apache при получении запроса с URL, содержащего значение вида "http://имя.сервера/source/путь/к/файлу.php", выводит листинг файла "http://имя.сервера/source/путь/к/файлу.php", сделайте следующее.

- Добавьте в файл httpd.conf следующий фрагмент:

```
# Используем директиву "ForceType" чтобы указать,
# что значение source в URL - не каталог, а имя сценария PHP
<Location /source>
  ForceType application/x-httpd-php
</Location>
```

- Создайте в корневом web-каталоге следующий файл с именем *source*:

```
<HTML><HEAD>
<TITLE>Source Display</TITLE>
</HEAD>
<BODY bgcolor=#FFEEDD>
<?php
$script = getenv ("PATH_TRANSLATED");
if(!$script) {
  echo "<BR><B>ERROR: Укажите имя сценария</B><BR>";
} else {
  if(ereg("(\.php|\.inc)$",$script)) {
    echo "<H!>Листинг файла: $PATH_INFO</H1>\n<hr>\n";
    if(!@highlight_file($script))
      echo "Ошибка вывода файла";
  } else {
    echo "<H1>ERROR: Показываются только листинги PHP файлов </H1>";
  }
}
echo "<HR>Распечатано: ".date("Y/M/d H:i:s",time());
?>
</BODY>
</HTML>
```

---

**highlight_string**
Выделение строки цветом.

**Синтаксис :**
void highlight_string(string str);

Функция действует подобно **highlight_file()**, но использует не содержимое файла, а указанной строки.

---

**show_source**
Синоним функции highlight_file.

**Синтаксис :**
boolean show_source(string str);

---

**pack**
Пакетирование данных в двоичную строку.

**Синтаксис :**
string pack(string format [,mixed $args, ...]);

Функция **pack()** упаковывает заданные аргументы в бинарную строку, которая затем и возвращается. Формат параметров, а также их количество, задается при помощи строки *$format*, которая представляет собой набор однобуквенных спецификаторов форматирования - наподобие тех, которые указываются в **sprintf()**, но только без знака %. После каждого спецификатора может стоять число, которое отмечает, сколько информации будет обработано данным спецификатором. А именно, для форматов a,A,h и H число задает, какое количество символов будет помещено в бинарную строку из тех, что находится в очередном параметре-строке при вызове функции (то есть, определяет размер поля для вывода строки). В случае @ оно определяет абсолютную позицию, в которую будут помещены следующие данные. Для всех остальных спецификаторов следующие за ними числа задают количество аргументов, на которые распространяется действие данного формата. Вместо числа можно указать *, в этом случае подразумевается, что спецификатор действует на все оставшиеся данные.

Вот полный список спецификаторов формата:

- a - строка, свободные места в поле заполняются символом с кодом 0;
- A - строка, свободные места заполняются пробелами;
- h - шестнадцатиричная строка, младшие разряды в начале;
- H - шестнадцатиричная строка, старшие разряды в начале;
- c - знаковый байт (символ);
- C - беззнаковый байт;
- s - знаковое короткое целое (16 битовб порядок байтов определяется архитектурой процессора);
- S - беззнаковое короткое число;
- n - беззнаковое целое (16 битов, старшие разряды в конце);
- v - беззнаковое целое (16 битов, младшие разряды в конце);
- i - знаковое целое (размер и порядок байтов определяется архитектурой);
- I - беззнаковое целое;
- l - знаковое длинное целое (32 бита, порядок знаков определяется архитектурой);
- L - беззнаковое длинное целое;
- N - беззнаковое длинное целое (32 бита, старшие разряды в конце);
- V - беззнаковое целое (32 бита, младшие разряды в конце);
- f - число с плавающей точкой (зависит от архитектуры);
- d - число с плавающей точкой двойной точности (зависит от архитектуры);
- x - символ с нулевым кодом;
- X - возврат назад на 1 байт;
- @ - заполнение нулевым кодом до заданной абсолютной позиции.

```
// Целое, целое, все остальное - сиволы
$bindata = pack("nvc*", 0x1234, 0x5678, 65, 66);
```

После выполнения приведенного кода в строке *$bindata* будет содержаться 6 байтов в такой последовательности:
0x12, 0x34, 0x78, 0x56, 0x41, 0x42 (в шестнадцатиричной системе счисления).

---

**unpack**
Распаковывает данные из двоичной строки.

**Синтаксис :**
array unpack(string format, string data);

Распаковывает данные из двоичной строки в массив согласно формату.
Возвращает массив, содержащий распакованные элементы.

```
$array = unpack("c2chars/nint", $binarydata);
```

Возникающий в результате массив будет содержать "chars1", "chars2" и "int".

---

**iptcparse**
Анализирует двоичный IPTC блок на одиночные тэги.

**Синтаксис :**
array iptcparse(string iptcblock);

Эта функция анализирует двоичный блок IPTC на одиночные теги. Возвращает массив, использующий tagmarker как индекс и значение как значение.
Возвращает false при ошибке или если никаких IPTC данных не было найдено.

---

**leak**
Имитация утечки памяти.

**Синтаксис :**
void leak(int bytes);

**leak()** отсекает определенный объем памяти.
Это полезно при отладке диспетчера памяти, который автоматически очищает "отсеченную" память при выполнении запроса.

Размер блока памяти указывается в байтах аргументом *bytes*.

---

**serialize**
Генерирует удобохраниемое представление значения.

**Синтаксис :**
string serialize(mixed value);

serialize() возвращает строку состоящую из потока байтов при представлениизначения value, которое может где-нибудь сохранено.
Это полезно для сохранения или передачи значений PHP без потери их типа и структуры.

**Пример :**

```
//$session_data содержит многомерный массив
       с информацией о сессии
//текущего пользователя.  Мы используем
       serialize() для сохранения
//этого в базе данных в конце запроса.

$conn = odbc_connect("webdb", "php", "chicken");
$stmt = odbc_prepare($conn,
  "UPDATE sessions SET data = ? WHERE id = ?");
$sqldata = array(serialize($session_data),
```

```
  $PHP_AUTH_USER);
if (!odbc_execute($stmt, &$sqldata)) {
  $stmt = odbc_prepare($conn,
    INSERT INTO sessions (id, data) VALUES(?, ?)");
  if (!odbc_execute($stmt, &$sqldata)) {
      /* Что-то сделано неправильно. */
  }
}
```

---

**unserialize**
Создает PHP значение из сохраненного представления.

**Синтаксис :**
mixed unserialize(string str);

unserialize() берет одно сохраненное значение и преобразует обратно в PHP значение. Возвращает преобразованное значение, и может иметь тип: integer, double, string, array или object. Если был преобразован object, то методы не востановятся.

**Пример :**

```
// Здесь мы используем unserialize() для загрузки
        данных о сессии из базы данных
// в $session_data.  Этот пример дополняет
        описанный в месте
// с serialize() .

$conn = odbc_connect("webdb", "php", "chicken");
$stmt = odbc_prepare($conn,
  "SELECT data FROM sessions WHERE id = ?");
$sqldata = array($PHP_AUTH_USER);
if (!odbc_execute($stmt, &$sqldata) ||
  !odbc_fetch_into($stmt, &$tmp)) {
// Если сбой запуска или выборки ,
//        то инициализируем массив
    $session_data = array();
} else {
    // Мы должны иметь представление в $tmp[0].
    $session_data = unserialize($tmp[0]);
    if (!is_array($session_data)) {
// Что-то неправильно, инициализируем массив
        $session_data = array();
    }
}
```

---

**uniqid**
Генерирует уникальный идентификатор.

**Синтаксис :**
int uniqid(string prefix [, boolean lcg]);

Функция **uniqid()** возвращает уникальный идентификатор, основанный на текущем времени в микросекундах и имеющий префикс *prefix*.
Префикс может быть полезен, например, если Вы генерируете идентификаторы одновременно на отдельных хостах, которые, могли бы случилось, генерировали идентификатор в одной и той же микросекунде. Префикс может быть длиной до 114 символов.
Если в качестве его значения передается пустая строка, то длина сгенерированного идентификатора будет 13 символов (при lcg=true - 23 символа).

Если указан необязательный аргумент *lcg* со значением true, к концу идентификатора будет добавляться "комбинированный хеш энтропии LCG", делающий его значение более уникальным.

Принято также дообрабатывать полученное значение криптографическими методами (например, это часто делается в идентификаторах сессий).

```
// без случайной части
$token = md5(uniqid(""));
// посложнее
$better_token = md5(uniqid(rnad()));
```

Эти строки генерируют 32 байта (128-битное шестнадцатеричное число): они обладают максимальной унакальностью, которая только может потребоваться.

---

## Почтовые функции

**mail**
Отсылает почту.

**Синтаксис :** mail(*$to, $subject, $msg [,$headers]*);
Функция **mail()** посылает сообщение с телом *$msg* (это может быть "многострочная строка", т.е. переменная, содержащая несколько строк, разделенных символом перевода строки) по адресу *$to*. Можно задать сразу несколько получателей, разделив их адреса пробелами в параметре *$to*.

**Пример :**

```
mail("spravka_web@chat.ru spravka_web@hut.ru",
     "Мое сообщение",
     "Первая строка\nВторая строка\nТретья строка"
);
```

В случае, если указан четвертый параметр, переданная в нем строка вставляется между концом стандартных почтовых заголовков (таких как *To*, *Content-type* и т.д.) и началом текста письма. Обычно этот параметр используется для задания дополнительных заголовков письма.

**Пример :**

```
mail("spravka_web@chat.ru spravka_web@hut.ru",
     "Тема",
     "Тело письма",
     "From: webmaster@chat.ru\n".
     "Reply-To: webmaster@chat.ru\n".
     "X-Mailer: PHP/" . phpvarsion()
);
```

---

## Функции запуска программ

**escapeshellcmd**
Убирает shell метасимволы.

**Синтаксис :**
string escapeshellcmd(string command);

Убирает любые символы в строке, которые могут быть использованы в командном интерпретаторе как произвольные команды. Эту функцию нужно использовать, что бы убедится, что все ваши данные введены правильно, и эту функцию лучше всего вставлять в функции exec() или system(). Стандартное использование этой

функции выглядит так:
system(EscapeShellCmd($cmd))

---

**exec**
Запуск внешней программы.

**Синтаксис :**
string exec(string command [, string array [, int return_var]]);

Функция **exec()** скрыто от пользователя запускает программу из строки command, весь стандартный вывод отключен. Возвращает последнюю строку результата выполнения программы.
Если параметр array установлен, то указанный массив будет заполнен выводом из программы. Помните, если массив уже содержит данные, то **exec()** добавляет свои данные в конец массива. Для очистки массива можно использовать функцию **unset()**.
Если параметр *return_var* установлен наряду с параметром array, то в него записывается результат выполнения команды.

```
<?php
$se = "dir c:\\";
$s0 = exec($se, $sa, $sr);
echo "При запуске команды "$se" последняя выведенная строка была:\n",
       $s0,"\n Код возврата ($sr) \nA это все что было выведено: ";
print_r($sa);
?>
```

Если требуется запустить программу в фоновом режиме (на длительное время), то поток ее вывода должен быть перенаправлен в файл (или иной поток вывода); иначе по истечении допустимого времени исполнения сценария (ожидания завершения внешней программы) он будет принудительно завершен с ошибкой.

---

**system**
Запуск внешней программы с выводом результата.

**Синтаксис :**
string system(string command, int [return_var]);

это функция для запуска command и вывода результата. Если используется второй параметр, то в него записывается результат выполнения команды. Вызов System() также пробует автоматически вставить в буфер вывода web сервера после каждой строки вывода, если PHP запущен как модель сервера.

---

**passthru**
Запускает внешнюю программу и выводит данные напрямую.

**Синтаксис :**
string passthru(string command [, int return_var]);

Функция **passthru()** похожа на функцию **exec()** для запуска *command*. Если параметр return_var установлен, то результат Unix команды помещается здесь. Эта функция должна использоваться вместо **exec()** или **system()** тогда, когда вывод из Unix команды является двоичными данными, которые должны быть переданы непосредственно обратно в окно просмотра(browser). Это можно использовать, например, для запуска утилиты pbmplus для вывода непосредственно потока изображения. Установка типа image/gif и вызов программы pbmplus, чтобы вывести gif-рисунок, вы можете создавать PHP скрипты, которые выводят изображения непосредственно.

---

## Функции динамической загрузки

**dl**
Загрузка библиотеки расширения PHP во время выполнения.

**Синтаксис :**
int dl(string library);
```
dl("extensions/php_db.dll");
```
Загружает PHP расширение определенное в library.

---

**get_loaded_extensions**
Определения перечня загруженных модулей.

**Синтаксис :**
array get_loaded_extensions(void);

Возвращает массив, содержащий список имен модулей PHP, которые были прокомпилированны, загружены при старте PHP и загружены во время исполнения функцией dl().
```
print_r (get_loaded_extensions());
```
Выводит информацию, подобную следующей:
```
Array
(
    [0] => standard
    [1] => bcmath
    [2] => calendar
    [3] => ctype
    [4] => com
    [5] => ftp
    [6] => mysql
    [7] => odbc
    [8] => overload
    [9] => pcre
    [10] => session
    [11] => tokenizer
    [12] => xml
    [13] => wddx
    [14] => zlib
    [15] => exif
    [16] => gd
    [17] => zip
)
```

---

**extension_loaded**
Проверка загрузки модуля.

**Синтаксис :**
bool extension_loaded(string name);

Возвращает true, если указанный модуль name уже был загружен. Следует обращать внимание на то, как пишется имя модуля, и на регистр символов.

---

**get_extension_funcs**
Определение функций модуля.

**Синтаксис :**
array get_extension_funcs(string module_name);

Возвращает массив, содержащий перечисление имен функций, содержащихся в модуле module_name. Этот модуль должен быть предварительно загружен.

```
print_r(get_extension_funcs("xml"));
```

---

## Информационные функции

**phpinfo**
Выводит текущее состояние всех параметров PHP.

**Синтаксис :**
int phpinfo([int what])

Для сокращения объема выводимой информации можно указать один из следующих разделов what (если он не указывается, то подразумевается INFO_ALL):

- INFO_GENERAL
- INFO_CREDITS
- INFO_CONFIGURATION
- INFO_MODULES
- INFO_ENVIRONMENT
- INFO_VARIABLES
- INFO_LICENSE
- INFO_ALL

Эта функция,которая в общем-то не должна появляться в законченной программе, выводит в броузер большое количество различной информации, касающейся настроек PHP и параметров вызова сценария. Именно, в стандартный выходной поток (то есть в браузер пользователя) печатается:
- версия PHP;
- опции, которые были установлены при компиляции PHP;
- информация о дополнительных модулях;
- переменные окружения, в том числе и установленные сервером при получении запроса от пользователя на вызов сценария;
- версия операционной системы;
- состояние основных и локальных настроек интерпретатора;
- HTTP-заголовки;
- лицензия PHP.

Функция **phpinfo()** в основном применяется при первоначальной установке PHP для проверки его работоспособности (уж больно много она выдает информации). Проверить работу этой функции можно нажав эту ссылку.

---

**phpversion**
Возвращает текущую версию PHP.

**Синтаксис :**
string phpversion();

Возвращает строку, содержащую название версии интерпретатора PHP.

```
echo phpversion();
```

Вот что примерно должно получиться:

```
4.3.6
```

---

**phpcredits**
HTML-распечатка разработчиков PHP.

**Синтаксис :**
void phpcredits(inf flag);

Выводит информацию о создателях и их вкладе в разработку пакета PHP.
```
phpcredits(CREDITS_GENERAL);
```
Флаги можно комбинировать следующим образом:
```
phpcredits(CREDITS_GROUP + CREDITS_DOCS + CREDITS_FULLPAGE);
```
Далее приведу список доступных флагов:

- CREDITS_ALL - Полный HTML-листинг.
- CREDITS_DOCS - Список разработчиков документации.
- CREDITS_FULLPAGE - Обычно используется в комбинации с другими флагами.

Выбирает вариант, подготовленный к распечатке.
- CREDITS_GENERAL - Общая разработка языка PHP 4.0 и SAPI
- CREDITS_GROUP - Список разработчиков ядра.
- CREDITS_MODULES - Список модулей расширения и их авторов.
- CREDITS_SAPI - Список разработчиков PHP модуля API сервера.

---

**php_sapi_name**
Получение типа интерфейса между Web-сервером и PHP.

**Синтаксис :**
string php_sapi_name();

Возвращает строку, содержащую строчными буквами тип интерфейса. Для CGI PHP, это будет строка "cgi", для mod_php под Apache - "apache" и т.п.
```
$sapi_type = php_sapi_name();
if($sapi_type == "cgi")
  echo "Это CGI PHP\n";
else
  echo "Это не CGI PHP а $sapi_type";
```
Вот что получиться для нашего случая:
```
Это не CGI PHP а cgi-fcgi
```

---

**php_uname**
Определение операционной системы.

**Синтаксис :**
string php_uname();

Возвращает строку, содержащую название операционной системы, например "Windows NT MYCOMP 5.1 build 2600".
```
if(substr(php_uname(),0,7) != "Windows") {
  die("Этот сценарий должен выполняться в Windows.");
}
```

---

**ini_set**
Изменение параметра конфигурации.

**Синтаксис :**
string ini_set(string varname, string newvalue);

Устанавливает для указанного параметра varname значение newvalue. При успехе возвращает прежнее значение, при ошибке - false.

---

**ini_alter**
Тоже, что и ini_set().

**Синтаксис :**
string ini_alter(string varname, string newvalue);

---

**ini_get**
Эта функция получает значения параметров конфигурации.

**Синтаксис :**
string ini_get(string varname);

Возвращает текущее значение парамерта конфигурации, заданное в переменной varname.
Данная функция позволяет получить все доступные в PHP парамеры.
В случае ошибки возвращает false.

---

**ini_restore**
Производит восстановление парамерта конфигурации.

**Синтаксис :**
string ini_restore(string varname);

Устанавливает значение параметра конфигурации varname в первоначальное.

```
echo ini_set("precision",20).ini_get("precision").
      ini_restore("precision").ini_get("precision");
// Выведет 14 20 14
```

---

**get_cfg_var**
Получает значения параметра непосредственно из файла php.ini.

**Синтаксис :**
string get_cfg_var(string varname);

Надо отметить, что в отличии от функции ini_get(), которая возвращает текущее значение параметра, функция get_cfg_var() возвращает значение параметра, которое установлено в файле конфигурации php.ini. Также эта функция не возвращает другие параметры (например, из конфигурации самого сервера).

---

**getenv**
Функция возвращает значение переменной окружения.

**Синтаксис :**
string getenv(string varname);

```
$ip = getenv("REMOTE_ADDR");
echo "Ваш IP-адрес: $ip";
```

Вот что получиться в результате работы:

```
Ваш IP-адрес: 127.0.0.1
```

Список переменных окружения можно посмотреть в Приложения->Переменные окружения, или при помощи функции phpinfo().
Эта функция не работает в модуле PHP ISAPI.

---

**putenv**
Устанавливает переменную окружения.

**Синтаксис :**
void putenv(string setting);

```
putenv("UNIQID=$uniquid");
```

---

**get_magic_quotes_gpc**
Получает текущее значение параметра magic_quotes_gpc.

**Синтаксис :**
long get_magic_quotes_gpc();

Эта функция возвратит 0 для Off и 1 для On.

---

**get_magic_quotes_runtime**
Предназначена для получения текущего значения патаметра magic_quotes_runtime.

**Синтаксис :**
long get_magic_quotes_runtime();

Эта функция возвратит 0 для Off и 1 для On.

---

**set_magic_quotes_runtime**
Предназначена для установки текущего значения патаметра magic_quotes_runtime.

**Синтаксис :**
long set_magic_quotes_runtime(int new_setting);

Для установки magic_quotes_runtime в Off задайте параметр new_setting равным 0, а для установки в On равным 1.

---

**php_logo_guid**
Функция получения GUID логотипа PHP.

**Синтаксис :**
string php_logo_guid();

Строка
```
echo php_logo_guid();
```
возвратит
```
PHPE9568F34-D428-11d2-A769-00AA001ACF42
```

---

**zend_logo_guid**
Функция получения GUID логотипа Zend.

**Синтаксис :**
string zend_logo_guid();

Строка
```
echo zend_logo_guid();
```
возвратит
```
PHPE9568F35-D428-11d2-A769-00AA001ACF42
```

---

## База данных MySQL : Работа с базами данных

**mysql_connect**
Устанавливает сетевое соединение с базой данных MySQL.

**Синтаксис :**
int mysql_connect([string $hostname[:port][:/path/to/socket][, [,string $username [,string $password]]])

Функция *mysql_connect()* устанавливает сетевое соединение с базой данных MySQL, расположенной на хосте *$hostname*, и возвращает идентификатор открытого соединения. Вся дальнейшая работа ведется именно с этим идентификатором. При регистрации указывается имя пользователя *$username* и пароль *$password*. Строка *$hostname* также может включать в себя номер порта в виде "hostname:port"или путь к сокету для локальной машины в системах Unix - ":/path/to/socket" (если сервер MySQL настроен не на станадартный, а на какой-то другой порт).
При ошибке выдается предупреждение. Выдачу сообщения об ошибке можно блокировать, указав перед именем функции оператор "@".
При следующем запуске функции с теми же самыми аргументами второе соединение не будет открыто, а функция возвратит идентификатор уже существующего.
В конце сценария обычно принято закрывать подключения функцией **mysql_close()**, но этого можно не делать, т.к. PHP автоматически закрывает все (неустойчивые) подключения при завершении сценария.

```
<?php
   $conn = mysql_connect ("localhost", "username", "pass")
        or die ("Соединение не установлено!");
   print ("Соединение установлено!");
   mysql_close($conn);
?>
```

---

**mysql_pconnect**
Устанавливает устойчивое сетевое соединение с базой данных MySQL.

**Синтаксис :**
int mysql_pconnect([string $hostname[:port][:/path/to/socket][, [,string $username [,string $password]]])

Функция **mysql_pconnect()** устанавливает устойчивое сетевое соединение с базой данных MySQL, расположенной на хосте *$hostname*, и возвращает идентификатор открытого соединения. Вся дальнейшая работа ведется именно с этим идентификатором. При регистрации указывается имя пользователя *$username* и пароль *$password*. Строка *$hostname* также может включать в себя номер порта в виде "hostname:port"или путь к сокету для локальной машины в системах Unix - ":/path/to/socket" (если сервер MySQL настроен не на станадартный, а на какой-то другой порт).
При ошибке выдается предупреждение. Выдачу сообщения об ошибке можно блокировать, указав перед именем функции оператор "@".
При следующем запуске функции с теми же самыми аргументами второе соединение не будет открыто, а функция возвратит идентификатор уже существующего.
**mysql_pconnect()** действует аналогично **mysql_connect**, но с двумя отличиями:
- Перед подключением функция пытается проверить, имеется ли уже открытое подключение. Если есть, то возвращается идентификатор вместо создания нового подключения.
- При завершении сценария подключение не закрывается, а остается действующим для дальнейшего использования, т.е. функция **mysql_close()** не может закрыть подключения, созданное с помощью **mysql_pconnect()**.

---

**mysql_close**
Закрывает установленное ранее соединение с базой данных.

**Синтаксис :**
int mysql_close ([int link_identifier])

Закрывает соединение с MySQL серевером с идентификатором *link_identifier*, или последнее открытое соединение, если используется без идентификатора.
Возвращает true при удачном закрытии или false при ошибке.
Использование этой функции не обязательно, т.к. PHP автоматически закрывает все неустойчивые подключения при завершении работы сценария.
Подключения, установленные функцией **mysql_pconnect()**, не закрываются.

```
<?php
   $conn = mysql_connect ("localhost", "username", "pass")
        or die ("Соединение не установлено!");
   print ("Соединение установлено!");
   mysql_close($conn);
?>
```

---

**mysql_change_user**
Изменяет параметры подключения.

**Синтаксис :**
int mysql_change_user(string user, string password [, string database [, int link_identifier]])

Если не указывается БД или подключение, то используется последняя активная БД.
Если авторизация не произошла, то параметры подключения не изменяются.
Работает с MySQL 3.23.3 и выше.

---

**mysql_list_dbs**
Возвращает список БД на сервере.

**Синтаксис :**
int mysql_list_dbs([int link_identifier])

Возврашает набор записей, содержащий список БД на сервере.

```
$bd=mysql_connect("localhost", "name", "pass");
$bd_list=mysql_list_dbs($bd);
while($row=mysql_fetch_object($bd_list)) {
  echo $row->Database."\n";
}
```

Надо отметить, что список баз данных можно получить не имея привелегий, т.е. не укзывая пароль доступа.

---

**mysql_db_name**
Возвращает имя базы данных из списка.

**Синтаксис :**
int mysql_db_name(int result, int row [, mixed field])

Параметр *result* задает дескриптор набора записей, полученных при помощи функции **mysql_list_dbs()**. Аргумент *row* указывает номер записи.
В случает ошибки данная функция возвращает false.

```
mysql_connect("localhost", "username", "pass");
$db_list=mysql_list();
for($i=0;$i<($cnt=mysql_num_rows($db_list));$i++) {
  echo mysql_db_name($db_list,$i)."\n";
}
```

Ранее функция называлась **mysql_dbname()**.

---

**mysql_select_db**
Выбор одной базы данных MySQL.

**Синтаксис :**
int mysql_select_db (string database_name [, int link_identifier])

Возвращает true при удачном закрытии или false при ошибке.
Если Вы планируете открывать только одно соединение с базой данных за все время работы сценария, то можете не сохранять возвращенное значение, а также не указывать идентификатор при вызове всех остальных функций.
До того как послать первый запрос серверу MySQL, необходимо указать, с какой базой данных мы собираемся работать. Для этого и предназначена данная функция. Она уведомляет, что в дальнейших операциях с соединением *link_identifier* (или с последним открытым соединением, если указанный параметр не задан) будет использоваться база данных *database_name*.
Если на момент вызова данной функции подключений к базе данных нет, то косвенно вызывается функция **mysql_connect()** с параметрами по умолчанию.

---

**mysql_create_db**
Создание базы данных MySQL.

**Синтаксис :**
int mysql_create_db(string dbname [, int link_identifier])

Эта функция создает новую базу данных MySQL с именем *dbname*, используя подключение *link_identifier*.

```
$db=mysql_connect("localhost", "name", "pass");
if(mysql_create_db("my_db_name")) {
  echo "БД my_db_name создана");
} else {
  echo "Ошибка создания БД: %s\n".mysql_error());
}
```

---

**mysql_drop_db**
Удаление базы данных MySQL.

**Синтаксис :**
int mysql_drop_db(string database_name [, int link_identifier])

Функция **mysql_drop_db()** удаляет базу данных *database_name*, доступную в подключении *link_identifier*.
В случае успешного удаления возвращает true, при ошибке - false.

---

**mysql_list_tables**
Возвращает список таблиц в БД.

**Синтаксис :**
int mysql_list_tables(string database [,int link_identifier])

Функция возвращает идентификатор результата (одна колонка), в котором содержаться имена всех таблиц, присутствующих в базе данных. Для извлечения этих имен можно использовать функцию **mysql_result()** с номером колонки, равным 0, иди функцию **mysql_tablename()**.

Следующий пример выведет все имена баз данных и таблиц, кororые в них содержаться:

```
$db=mysql_connect("localhost", "user_name", "");
$db_list=mysql_list_dbs($db);
while($r_db=mysql_fetch_object($db_list)) {
  echo $r_db->Database."\n";
    // распечатать список таблиц
  $t_list=mysql_list_tables($r_db->Database);
  for($i=0;$i<mysql_num_rows($t_list);$i++) {
    echo " - ".mysql_tablename($t_list,$i)."\n";
  }
}
```

---

**mysql_tablename**
Возвращает имя таблицы в БД.

**Синтаксис :**
int mysql_tablename(int result, int i)

Функция возвращает имя таблицы с номером *i* из набора записей, полученных при помощи функции **mysql_list_tables()**.

```
$db=mysql_connect("localhost", "user_name", "");
$result=mysql_list_tables("db_name");
$i=0;
while($i<mysql_num_rows($result)) {
  $t_name[$i]=mysql_tablename($result, $i);
  echo $t_name[$i]."<BR>";
  $i++;
}
```

---

**mysql_query**
Посылает запрос базе данных MySQL.

**Синтаксис :**
int mysql_query(string query [,int link_identifier])

Эта функция посылает запрос *query* базе данных, связанной с идентификатором *link_identifier* Если идентификатор не указан, то принимается во внимание последнее открытое соединение. Если до этого соединение небыло установлено, то выполняется операция **mysql_connect()** с параметрами по умолчанию.
SQL-выражение, указанное в параметре *query*, не должно оканчиваться ";".
Если выражение содержит ошибки, или его выполнение приводит к ошибкам, то функция **mysql_query()** возвращает false.
В результате успешно выполненного запроса возвращается набор записей, который можно обработать следующими функциями:

- mysql_result() - получить элемент набора записей
- mysql_fetch_array() - внести запись в массив
- mysql_fetch_row() - занести запись в нумерованный массив
- mysql_fetch_assoc() - занести запись в ассоциативный массив
- mysql_fetch_object() - занести запись в объект

Чтобы узнать, сколько записей было найдено командой SELECT, воспользуйтесь функцией **mysql_num_rows()**.
Для того, чтобы узнать, сколько записей было изменено в результате выполнения запросов DELETE, INSERT, REPLACE или UPDATE, воспользуйтесь функцией **mysql_affected_rows()**.

После обработки результатов запроса он может быть удален функцией

**mysql_free_result()**. Но в этом нет необходимости, т.к. результаты сами уничтожаются после завершения работы сценария.

---

**mysql_db_query**
Посылает запрос к указанной базе данных MySQL.

**Синтаксис :**
int mysql_db_query(string database, string query [,int link_identifier])

Эта функция эквивалентна следующей последовательности функций:

mysql_select_db(string database [, int link_identifier]);
mysql_query(string query [, int link_identifier]);

---

**mysql_num_rows**
Возвращает количество строк в результате запроса.

**Синтаксис :**
int mysql_num_rows(int result)

Эта функция возвращает число записей, найденных в результате выполнения SQL-команды SELECT (поиск по базе данных).

```
<?
$link = mysql_connect("localhost", "username", "password");
mysql_select_db("database", $link);

$result = mysql_query("SELECT * FROM table1", $link);
$num_rows = mysql_num_rows($result);

echo "Получено строк: $num_rows\n";
?>
```

---

**mysql_affected_rows**
Возвращает количество измененных записей в БД MySQL.

**Синтаксис :**
int mysql_affected_rows([int link_identifier]);

Функция **mysql_affected_rows()** возвращает количество записей, которые были изменены в базе данных в результате выполнения запросов DELETE, INSERT, REPLACE или UPDATE.
Если последним запросом была команда DELETE без ограничения WHERE (т.е. из таблицы были удалены все записи), то наша функция возвратит 0.

---

**mysql_insert_id**
Получает вставленный идентификатор.

**Синтаксис :**
int mysql_insert_id([int $link_identifier])

Функция возвращает непосредственно перед ее вызовом сгенерированный идентификатор записи для автоинкрементного поля после выполнения команды insert. Вызывать ее разумно только сразу после выполнения инструкции insert, например, в таком контексте:

mysql_query("insert into Таблица(поле 1, поле 2) values("aa","bb")");
$id=mysql_insert_id();

Теперь к только что вставленной записи можно обратиться, используя идентификатор $id:

```
$r=mysql_query("select * from Таблица where id=$id");
$Row=mysql_fetch_array($r);
```

---

**mysql_data_seek**
Устанавливает указатель текущей строки.

**Синтаксис :**
int mysql_data_seek(int result, int row_number)

Эта функция устанавливает указатель текущей строки в результате *result* в позицию *row_number*, так что следующий вызов **mysql_fetch_row()** и **mysql_fetch_array()** вернет значения полей именно этой строки.
Нумерация записей ведется с 0.
Возвращает false в случае ошибки или если строки кончились.

---

**mysql_free_result**
Уничтожает набор записей.

**Синтаксис :**
int mysql_free_result(int result)

Данная функция освобождает память, занимаемую набором записей *result*, возвращенным запросом.
Эта функция необходима, когда нужно экономить память, т.к. PHP автоматически освобождает память при завершении сценария.

---

## Базы данных MySQL : Обработка результатов запроса

**mysql_result**
Получение определенного поля результата.

**Синтаксис :**
int mysql_result(int result, int row [, mixed field])

Функция возвращает значение поля *field* в строке результата с номером *row*.Параметр *field* может задавать не только имя поля, но и его номер - позицию, на которой столбец "стоял" при создании таблицы, а также полное имя поля вида: "имя_таблицы.имя_поля". Тем не менее, рекомендуется везде, где это только возможно,использовать именно имена полей.

Функция универсальна: с ее помощью можно "обойти" весь результат по одной ячейке. И хотя это не возбраняется, но делать, однако, не рекомендуется, т.к. **mysql_result()** работает довольно медленно.

---

**mysql_fetch_array**
Извлекает из результата очередную запись и помещает ее в ассоциативный массив.

**Синтаксис :**
array mysql_fetch_array(int result [, int result_type])

Функция **mysql_fetch_array()** возвращает очередную строку результата в виде

ассоциативного массива, где каждому полю сопоставлен элемент с ключом, совпадающим с именем поля. Дополнительно в массив записываются элементы с числовыми ключами и значениями, соответствующими величинам полей с этими индексами. В возвращаемом массиве они размещаются сразу за элементами с "обычными" ключами.

Параметр *result_type* задает вид возвращаемого массива и может принимать одно из следующих значений: MYSQL_NUM, MYSQL_ASSOC, MYSQL_BOTH (по умолчанию).
Может возникнуть вопрос: зачем вообще нужны числовые индексы. Ответ прост: дело в том, что в результате выборки в действительности могут присутствовать поля (фактически, колонки) с одинаковыми именами, но, соответственно, с различными индексами. Это происходит тогда, когда выборка в SELECT производится одновременно из нескольких таблиц.

```
mysql_connect($host, $user, $pass);
$result=mysql_db_query("database", "select id, name from tabl");
while($row=mysql_fetch_array($result)) {
  echo "id: ".$row["id"]."<BR>";
  echo "id: ".$row[0]."<BR>";
  echo "name: ".$row["name"]."<BR>";
  echo "name: ".$row[1]."<BR>";
};
mysql_free_result($result);
```

---

**mysql_fetch_row**
Записывает запись в нумерованный массив.

**Синтаксис :**
array mysql_fetch_row(int result)

Функция возвращает массив-список со значениями полей очередной строки результата *result*. Если указатель текущей позиции результата был установлен за последней записью (то есть строки кончились), возвращается false. Текущая позиция сдвигается к следующей записи, так что очередной вызов **mysql_fetch_row()** вернет следующую строку результата.
Каждое поле записи сохраняется в нумерованном элементе массива. Нумерация начинается с 0.

```
$r=mysql_query("select * from OutTable where age<30");
while($Row=mysql_fetch_row($r)) {
  // обрабатываем строку $Row
}
```

Как видим, цикл оборвется, как только строки закончатся, т.е. когда **mysql_fetch_row()** вернет false.

---

**mysql_fetch_object**
Получение записи в свойствах объекта.

**Синтаксис :**
object mysql_fetch_object(int result)

Функция возвращает объект, в свойствах которого находятся поля текущей записи. В случае, если записи кончились, возвращает false.

```
mysql_connect($host, $user, $pass);
$result=mysql_db_query("database", "select * from table");
while($rows=mysql_fetch_object($result)) {
  echo $rows->id;
  echo $rows->name;
};
```

---

**mysql_fetch_lengths**
Возвращает длину элемента записи.

**Синтаксис :**
array mysql_fetch_lengths(int result)

Функция **mysql_fetch_lengths()** возвращает длину значения, полученного при помощи функций **mysql_fetch_row()**, **mysql_fetch_array()** или **mysql_fetch_object()**.

Например, в следующем примере:
```
$arr=mysql_fetch_row($result);
$len=mysql_fetch_lengths($result);
```
массив $len будет содержать длину соответствующих элементов массива $arr, т.е. $len[0]=strlen(arr[0]) и т.д.

---

**mysql_fetch_field**
Возвращает информацию о свойствах объекта и о поле записи.

**Синтаксис :**
object mysql_fetch_field(int result [, int field_offset])

В необязательном параметре *field_offset* задается номер поля, свойства которого мы хотим получить. Если этот параметр не указан, при каждом вызвове функции **mysql_fetch_field()** возвращаются свойства следующего поля из набора записей *result*.

Возвращаемый объект имеет следующие свойства:
- name - имя поля
- table - имя таблицы, которой принадлежит поле
- max_length - максимальная длина поля
- not_null - 1, если полю разрешено пустое значение
- primary_key - 1, если поле является ключевым
- unique_key - 1, если в поле допускаются только уникальные значения
- multiple_key - 1, если в поле допустимо иметь повторяющиеся значения
- numeric - 1, если поле числовое
- blob - 1, если поле имеет тип BLOB
- type - тип поля
- unsigned - 1, если поле числовое беззнаковое
- zerofill - 1, есле поле заполняется нулями

```
mysql_connect($host,$user,$pass);
$result=mysql_db_query("database", select * from table");
for($i=0;$i<mysql_num_fields($result);$i++) {
  echo "Свойства поля $i:<BR>";
  $param=mysql_fetch_field($result);
  if(!$param) echo "Нет информации о свойствах!";
  echo "<PRE>
name:          $param->name
table:         $param->table
max_length:    $param->max_length
not_null:      $param->not_null
primary_key:   $param->primary_key
unique_key:    $param->unique_key
multiple_key:  $param->multiple_key
numeric:       $param->numeric
blob:          $param->blob
type:          $param->type
unsigned:      $param->unsigned
zerofill:      $param->zerofill
```

```
</PRE>";
}
```

---

**mysql_field_seek**
Производит перемещение курсора к указанному полю.

**Синтаксис :**
int mysql_field_seek(int result, int field_offset)

Данная функция является излишней. Следующие фрагменты будут эквивалентны:

```
$param=mysql_fetch_field($result, field_offset);
и
mysql_field_seek($result, field_offset);
$param=mysql_fetch_field($result);
```

---

**mysql_field_name**
Возвращает имя поля.

**Синтаксис :**
string mysql_field_name(int result, int filed_index)

Функция **mysql_field_name()** возвращает имя поля, которое расположено в результате *result* с индексом *filed_index* (нумерация начинается с 0).

```
$result=mysql_query("SELECT id, name from table");
echo mysql_field_name($result,1); // Выведет: name
```

---

**mysql_field_table**
Возвращает имя таблицы, из которой было извлечено поле.

**Синтаксис :**
string mysql_field_table(int result, int field_offset)

Возвращает имя таблицы, из которой было извлечено поле со смещением *field_offset* в результате *result*.

---

**mysql_field_len**
Возвращает длину поля.

**Синтаксис :**
int mysql_field_len(int result, int filed_offset)

Функция возвращает длину поля в результате *result*. Поле, как обычно, задается указанием его смещения. Под длиной здесь подразумевается не размер данных поля в байтах, а тот размер, который был указан при его создании. Например, если поле имеет тип *varchar* и было создано (вместе с таблицей) с типом *varchar(100)*, то для него будет возвращено 100.

---

**mysql_field_type**
Возвращает тип набора записей в результате.

**Синтаксис :**
string mysql_field_type(int result, int filed_offset)

Эта функция похожа на **mysql_filed_name()**, только возвращает не имя , а тип соответствующей колонки в результате. Им может быть, например, int, double, real и т.д.

```
mysql_connect($host,$user,$pass);
mysql_select_db("mydb");
```

```
$result=mysql_query("SELECT * FROM tabl");
$fields=mysql_num_fields($result);
$rows=mysql_num_rows($result);
$i=0;
$table=mysql_field_table($result,$i);
echo "Таблица "$table" имеет $fields полей и $rows записей<BR>";
echo "Структура таблицы:<BR>";
while($i<$fields) {
  $type=mysql_field_type($result,$i);
  $name=mysql_field_name($result,$i);
  $len=mysql_field_len($result,$i);
  $flags=mysql_field_flags($result,$i);
  echo $type." ".$name." ".$len." ".$flags."<BR>";
  $i++;
}
```

---

**mysql_field_flags**
Эта функция возвращает флаги, которые были использованы при создании указанного поля в таблице.

**Синтаксис :**
string mysql_field_flags(int result, int field_offset)

Возвращаемая строка представляет собой набор слов, разделенных пробелами, так что вы можете преобразовать ее в массив при помощи функции **explode()**:

```
$Flags=explode(" ",mysql_field_flags($r,$field_offset));
```

Поля записей в MySQL могут иметь следующие свойства-флаги: "not_nul", "primary_key", "unique_key", "multiple_key", "blob", "unsigned", "zerofill", "binary", "enum", "auto_increment", "timestamp".

---

**mysql_list_fields**
Возвращает список полей таблицы.

**Синтаксис :**
int mysql_list_fields(string dbname, string tblname [,int link_identifier])

Функция **mysql_list_fields()** возвращает информацию об указанной таблице *tblname* в базе данных *bdname*, используя идентификатор соединения *link_identifier*, если он задан (в противном случае - последнее открытое соединение). Возвращаемое значение - идентификатор результата, который может быть проанализирован обычными средствами. В случае ошибки возвращается -1, текс сообщения ошибки может быть получен обычным способом.

```
$link=mysql_connect($host,$user,$pass);
$fields=mysql_list_fields("db1", "table", $link);
$colums=mysql_num_fields($fields); // число полей в таблице
// Далее распечатаем имена всех полей таблицы
for($i=0;$i<$colums;$i++) {
  echo mysql_field_name($fields,$i)."<BR>";
}
```

---

**mysql_num_fields**
Эта функция возвращает число полей в одной строке результата, т.е. число колонок в результате.

**Синтаксис :**
int mysql_num_fields(int result)

В силу сказанного, функция позволяет определить горизонтальную размерность "двумерного массива результата".

---

**mysql_errno**
Возвращает номер последней ошибки.

**Синтаксис :**
int mysql_errno ([int link_identifier])

Данная функция возвращает номер последней зарегистрированной ошибки или 0, если ошибок нет.
Идентификатор *link_identifier* можно не указывать, если за время работы сценария было установлено только одно соединение.

```
mysql_connect("dbname");
echo mysql_errno().": ".mysql_error()."<BR>";
```

---

**mysql_error**
Возвращает сообщение об ошибке.

**Синтаксис :**
string mysql_error ([int link_identifier])

Эта функция возвращает строку, содержащую текст сообщения об ошибке или пустую строку, если ошибок небыло.

```
mysql_connect("dbname");
echo mysql_errno().": ".mysql_error()."<BR>";
```

---

## Работа с изображениями и библиотека GD : Параметры изображения

### Применение и установка

Здесь мы рассмотрим идею создания рисунков сценарием "на лету". Это очень может пригодиться при создании сценариев-счетчиков, графиков, картинок-заголовков, да и многого другого.

Для деятельности такого рода существует специальная библиотека под названием GD. Она содержит в себе множество функций (такие как рисование линий, растяжение/сжатие изображения, заливка до границы, вывод текста и т.д.), которые могут использовать программы, поддерживающие работу с данной библиотекой.

Поддержка GD включается при компиляции и установке PHP. Возможно, некоторые хостинг-провайдеры ее не имеют.

Для подключения модуля на вашем локальном диске нужно открыть в Блокноте файл *php.ini* из каталога с файлами Windows (обычно C:\Windows)
Затем:
1. Настройте следующий параметр:
*extension_dir=C:\Program Files\PHP4\extensions*
  Здесь мы уведомляем PHP, что модули он должен искать
  в каталоге C:\Program Files\PHP4\extensions.
2. Найдите закомментированную строку *;extension=php_gd.dll*
   и разкомментируйте ее, т.е. уберите ; в начале.
3. Сохраните изменения в файле *php.ini*

---

**imageTypes**
Определение графических форматов, поддерживаемых PHP.

**Синтаксис :**
int imageTypes(void)

Функция возвращает битовую маску графических форматов, которые поддерживает данная версия библиотеки GD: IMG_GIF | IMG_JPG | IMG_PNG | IMG_WBMP

```
<?php
if(imageTypes() && IMG_PNG) echo "формат PNG поддерживается";
?>
```

---

**GetImageSize**
Определение размеров рисунка.

**Синтаксис :**
array GetImageSize(string filename [,array imageinfo])

Эта функция предназначена для быстрого определения в сценарии размеров (в пикселях) и формата рисунка, имя файла которого передано ей в первом параметре. Она возвращает список из четырех элементов. Первый элемент (с ключем 0) хранит ширину картинки в пикселях, второй (с ключем 1) - его высоту. Ячейка массива с ключем 2 определяется форматом изображения: 1, если это GIF, 2 в случае JPG, 3 для PNG и 4 - SWF. Следующий элемент, имеющий ключ 3, будет содержать после вызова функции строку примерно следующего вида: height=sx width=sy, где sx и sy - соответственно ширина и высота изображения. Это применение задумывалось для того, чтобы облегчить вставку данных о размере изображения в тег <*img*>, который может быть сгенерирован сценарием:

```
<?php
$size_img=GetImageSize("img/image.jpg");
echo "<IMG src='img/image.jpg' $size_img>";
?>
```

Если при обращении к функции был указан второй необязательный массив imageinfo, в него можно записать дополнительную информацию о файле. Это могут быть, например, различные маркеры JPG APP (внедренная информация). Функция iptcparse() позволяет конвертировать эти данные в читаемый вид:

```
<?php
$size_img=GetImageSize("img/image.jpg", &$info_arr);
if(isset($info_Arr["APP13"])) {
  $iptc = iptcparse($info_arr["APP13"]);
  var_dump($iptc);
};
?>
```

Данная функция не требует наличия библиотеки GD.

---

**imageSX**
Определение ширины рисунка.

**Синтаксис :**
int imageSX(int im)

Функция возвращает горизонтальный размер изображения, заданного своим идентификатором *im*, в пикселях.

**imageSY**
Определение высоты рисунка.

**Синтаксис :**
int imageSY(int im)

Функция возвращает вертикальный размер изображения, заданного своим идентификатором *im*, в пикселях.

---

**read_exif_data**
Чтение заголовков EXIF из файла JPEG.

**Синтаксис :**
array read_exif_data(string filename)

Параметр *filename* не может быть URL.

Функция возвращает ассоциативный массив, в котором индексами являются имена заголовков EXIF.
Заголовки EXIF обычно хранят информацию цифровых камер (в различном виде).

```
<?php
$exif = read_exif_data("img/file.jpg");
print_r($exif);
?>
```

Данный пример выведет что-то наподобие:

```
Array
(
    [FileName] => file.jpg
    [FileDateTime] => 1064566998
    [FileSize] => 31646
    [CameraMake] => Eastman Kodak Company
    [CameraModel] => KODAK DC265 ZOOM DIGITAL CAMERA (V01.00)
    [DateTime] => 2002:08:31 02:12:45
    [Height] => 454
    [Width] => 620
    [IsColor] => 1
    [FlashUsed] => 0
    [FocalLength] => 8.0mm
    [RawFocalLength] => 8
    [ExposureTime] => 0.004 s (1/250)
    [RawExposureTime] => 0.0040000001899898
    [ApertureFNumber] => f/ 9.5
    [RawApertureFNumber] => 9.5100002288818
    [FocusDistance] => 16.66m
    [RawFocusDistance] => 16.659999847412
    [Orientation] => 1
    [ExifVersion] => 0200
)
```

Данная функция доступна, если подключена библиотека EXIF.
Для этого необходимо либо снять комментарий со строки ;extension=php_exif.dll в файле windows\php.ini (должно быть extension=php_exif.dll), либо откомпилировать PHP с параметром --enable-exif.

Для работы этой функции библиотека GD не требуется.

---

**imageInterlace**
Установка черезстрочности.

**Синтаксис :**
int imageInterlace(int im [, int interlace])

Если в функции задан второй необязательный параметр *interlace*, и он равен 1, то рисунок *im* отображается черезстрочно, если равен 0, то последовательно.

Функция возвращает текущую установку черезстрочности.

---

**gd_info**
Возвращает информацию о библиотеке GD.

**Синтаксис :**
array gd_info(void)

Функция возвращает массив, содержащий версию и параметры установленной библиотеки GD.

```
<?php
$gd=gd_info();
echo "<pre>";
print_r($gd);
echo "</pre>";
?>
```

Приведенный пример выведет примерно следующее:

```
Array
(
    [GD Version] => bundled (2.0.22 compatible)
    [FreeType Support] => 1
    [FreeType Linkage] => with freetype
    [T1Lib Support] =>
    [GIF Read Support] => 1
    [GIF Create Support] =>
    [JPG Support] => 1
    [PNG Support] => 1
    [WBMP Support] => 1
    [XBM Support] => 1
    [JIS-mapped Japanese Font Support] =>
)
```

---

**image_type_to_mime_type**
Возвращает Mime-Type типа изображения.

**Синтаксис :**
string image_type_to_mime_type( int imagetype)

Функция возвращает MIME-тип картинки, заданный константой в параметре *imagetype*.

```
<?php
header("Content-type: " . image_type_to_mime_type(IMAGETYPE_PNG));
?>
```

Список констант и возвращаемых значений функции **image_type_to_mime_type()**:

- **IMAGETYPE_GIF** - image/gif

- **IMAGETYPE_JPEG** - image/jpeg
- **IMAGETYPE_PNG** - image/png
- **IMAGETYPE_SWF** - application/x-shockwave-flash
- **IMAGETYPE_PSD** - image/psd
- **IMAGETYPE_BMP** - image/bmp
- **IMAGETYPE_TIFF_II** - image/tiff
- **IMAGETYPE_TIFF_MM** - image/tiff
- **IMAGETYPE_JPC** - application/octet-stream
- **IMAGETYPE_JP2** - image/jp2
- **IMAGETYPE_JPX** - application/octet-stream
- **IMAGETYPE_JB2** - application/octet-stream
- **IMAGETYPE_SWC** - application/x-shockwave-flash
- **IMAGETYPE_IFF** - image/iff
- **IMAGETYPE_WBMP** - image/vnd.wap.wbmp
- **IMAGETYPE_XBM** - image/xbm Данная функция не требует наличия библиотеки GD.

---

## Работа с изображениями и библиотека GD : Манипулирование изображениями

**imageCreate**
Создание пустой картинки.

**Синтаксис :**
int imageCreate(int x, int y)

Создает пустую картинку размером *x* на *y* точек и возвращает ее идентификатор. После того, как картинка создана, вся работа с ней осуществляется именно через этот идентификатор, по аналогии с тем, как мы работаем с файлом через его дескриптор.

**Пример:**
Создание новой картинки при помощи GD и вывод ее в экран браузера:

```
<?php
header ("Content-type: image/png");
$im = @imagecreate (50, 100)
    or die ("Не удается открыть новую картинку!");
$background_color = imagecolorallocate ($im, 255, 255, 255);
$text_color = imagecolorallocate ($im, 233, 14, 91);
imagestring ($im, 1, 5, 5,  "A Simple Text String", $text_color);
imagepng ($im);
?>
```

---

**imageCreateFromPng**
Создание рисунка из файла PNG.

**Синтаксис :**
int imageCreateFromPng(string filename)
Эта функция загружают изображения из файла PNG в память и возвращает его идентификатор. Как и после вызова **imageCreate()**, дальнейшая работа с картинкой возможна только через этот идентификатор. При загрузке с диска изображение распаковывается и храниться в память уже в неупакованном формате, для того чтобы можно было максимально быстро производить с ним различные операции, такие как масштабирование, рисование линий и т.д.

**Пример:**
Пример нахождения ошибки при открытии графического файла.

```
function LoadPNG ($imgname) {
  $im = @imagecreatefrompng ($imgname); /* Attempt to open */
  if (!$im) { /* See if it failed */
    $im  = imagecreate (150, 30); /* Create a blank image */
    $bgc = imagecolorallocate ($im, 255, 255, 255);
    $tc  = imagecolorallocate ($im, 0, 0, 0);
    imagefilledrectangle ($im, 0, 0, 150, 30, $bgc);
    /* Output an errmsg */
    imagestring ($im, 1, 5, 5, "Error loading $imgname", $tc);
  }
  return $im;
}
```

---

**imageCreateFromJpeg**
Создание рисунка из файла JPEG.

**Синтаксис :**
int imageCreateFromJpeg(string filename)
Эта функция загружают изображения из файла в память и возвращают его идентификатор. Как и после вызова **imageCreate()**, дальнейшая работа с картинкой возможна только через этот идентификатор. При загрузке с диска изображение распаковывается и храниться в память уже в неупакованном формате, для того чтобы можно было максимально быстро производить с ним различные операции, такие как масштабирование, рисование линий и т.д.

---

**imageCreateFromGif**
Создание рисунка из файла GIF.

**Синтаксис :**
int imageCreateFromGif(string filename)
Эта функция загружают изображения из файла в память и возвращают его идентификатор. Как и после вызова **imageCreate()**, дальнейшая работа с картинкой возможна только через этот идентификатор. При загрузке с диска изображение распаковывается и храниться в память уже в неупакованном формате, для того чтобы можно было максимально быстро производить с ним различные операции, такие как масштабирование, рисование линий и т.д.

Стоит упомянуть, что GD начиная с версии 1.6 не поддерживает формат GIF. В связи с этим данная функция практически не используется.

---

**imagePng**
Функция выводит изображение в PNG-формате в любой браузер или в файл.

**Синтаксис:**
int imagePng(int im [, string filename])

Эта функция сохраняет изображение, заданное своим идентификатором и находящееся в памяти, на диск, или же выводят его в браузер.

Разумеется, вначале изображение должно быть загружено или создано при помощи функции **imageCreate()**, т.е. мы должны знать его идентификатор *im*.

Если аргумент *filename* опущен, то сжатые данные в соответствующем формате выводятся прямо в стандартный выходной поток, т.е. в браузер. Нужный заголовок *Content-type* при этом не выводится, ввиду чего нужно выводить его вручную при помощи **Header()**.
Фактически, вы должны вызвать одну из трех команд, в зависимости от типа изображения:

Header("Content-type: image/png") для PNG.

**Пример:**
Пример использования функции imagepng():

```
<?php
$im = imagecreatefrompng ("test.png");
Header("Content-type: image/png")
imagepng ($im);
?>
```

---

**imageJpeg**
Отсылка рисунка JPEG браузеру или сохранение его в файле.

**Синтаксис:**
int imageJPEG(int im [, string filename [, int quality]])

Эта функция сохраняет изображение, заданное своим идентификатором и находящееся в памяти, на диск, или же выводит его в браузер.

Разумеется, вначале изображение должно быть загружено или создано при помощи функции **imageCreate()**, т.е. мы должны знать его идентификатор *im*.

Если аргумент *filename* опущен, то сжатые данные в соответствующем формате выводятся прямо в стандартный выходной поток, т.е. в браузер. Нужный заголовок *Content-type* при этом не выводится, ввиду чего нужно выводить его вручную при помощи **Header()**.
Фактически, вы должны вызвать одну из трех команд, в зависимости от типа изображения:

Header("Content-type: image/jpeg") для Jpeg

Третий необязательный параметр *quality* задает качество изображения (от 0 до 100).

```
<?php
$im=imageCreateFromJPEG("img/file.jpg");
Header("Content-type: image/jpeg");
imageJPEG($im,"",30);
?>
```

---

**image2WBMP**
Вывод изображения в браузер или файл.

**Синтаксис :**
int image2WBMP( resource image [, string filename [, int threshold]])

Функция выводит изображение, заданное дескриптором *image*, в браузер, либо в файл, имя которого задано необязательным параметром *filename*.

Если изображение выводится в браузер, необходимо задать его тип WBMP как *image/vnd.wap.wbmp* функцией Header():

```
<?php
$file = "php.png";
$image = imagecreatefrompng($file);

header("Content-type: " . image_type_to_mime_type(IMAGETYPE_WBMP));
image2wbmp($image); // Вывод wbmp-картинки в браузер
?>
```

Функция **image2WBMP()** доступна PHP только если версия библиотеки GD 1.8 или ниже.

---

**imageGif**
Отсылка рисунка GIF браузеру или сохранение его в файле.

**Синтаксис:**
int imageGIF(int im [, string filename])

Функция сохраняют изображение, заданное своим идентификатором и находящееся в памяти, на диск, или же выводят его в браузер.

Разумеется, вначале изображение должно быть загружено или создано при помощи функции **imageCreate()**, т.е. мы должны знать его идентификатор *im*.

Если аргумент *filename* опущен, то сжатые данные в соответствующем формате выводятся прямо в стандартный выходной поток, т.е. в браузер. Нужный заголовок *Content-type* при этом не выводится, ввиду чего нужно выводить его вручную при помощи **Header()**.
Фактически, вы должны вызвать одну из трех команд, в зависимости от типа изображения:

Header("Content-type: image/gif").

Т.к. библиотека GD, начиная с версии 1.6, не поддерживает формат GIF, данная функция используется редко.

---

**imageCopy**
Копирование части рисунка.

**Синтаксис :**
int imageCopy(int dst_im, int src_im, int dst_x, int dst_y, int src_x, int src_y, int src_w, int src_h)

Функция копирует прямоугольную область начиная с позиции (*src_x, src_y*) шириной *src_w* и высотой *src_h* из рисунка *src_im* в рисунок *dst_im*, придав копируемой области смещение (*dst_x, dst_y*).

В следующем примере будет картинка *file1.png* целиком скопирована в *file2.png*

```
<?php
// Создаем первую картинку на основе готового изображения
$im1=imageCreateFromPNG("img/file1.png");
// Определяем ее размеры
$size_x=imageSX($im1);
$size_y=imageSY($im1);
```

```
// Создаем вторую пустую картинку
$im2=imageCreate($size_x,$size_y);
// Копируем рисунок целиком из первого изображения во второе
imageCopy($im2,$im1,0,0,0,0,$size_x,$size_y);
// Сохранение скопированной картинки в файле
imagePNG($im2, "img/file2.png");
?>
```

**imageCopyResized**
Копирование части рисунка с масштабированием.

**Синтаксис :**
int imageCopyResized(int dst_im, int src_im, int dstX, int dstY, int srcX, int srcY, int dstW, int dstH, int srcW, int srcH)

Эта функция - одна из самых мощных и универсальных. С помощью нее можно копировать изображения (или их участки), перемещать или масштабировать их.

*dst_im* задает идентификатор изображения, в который будет помещен результат работы функции. Это изображение должно уже быть создано или загружено и иметь надлежащие размеры.

*src_im* - идентификатор изображения, над которым проводится работа. Впрочем, *src_im* и *dst_im*могут и совпадать.

Параметры *srcX, srcY, srcW, srcH* задают область внутри исходного изображения, над которой будет осуществлена операция - соответственно, координаты ее верхнего левого угла, ширину и высоту.

Наконец, четверка *dstX, dstY, dstW, dstH* задает то место на изображении *dst_im*, в которое будет "втиснут" указанный в предыдущей четверке прямоугольник. Заметьте, что, если ширина или высота двух прямоугольников не совпадают, то картинка автоматически будет нужным образом растянута или сжата.

В следующем примере файл *file1.jpg* уменьшается вдвое и записывается в файл *file2.jpg*:

```
<?php
$old = imageCreateFromJpeg("img/file1.jpg");
$w = imageSX($old);
$h = imageSY($old);
$w_new=rounr($w/2);
$h_new=round($h/2);
$new = imageCreate($w_new, $h_new);
imageCopyResized($new, $old, 0, 0, 0, 0, $w_new, $h_new, $w, $h);
imageJpeg($new, "img/file2.jpg");
imageDestroy($old);
imageDestroy($new);
?>
```

**imageDestroy**
Уничтожение рисунка.

**Синтаксис :**
int imageDestroy(int im)

Функция уничтожает дескриптор *im* ранее созданного рисунка (наподобие закрытия файла fclose() после открытия fopen()).

# Работа с изображениями и библиотека GD : Работа с цветом в формате RGB

**imageColorAllocate**
Создание нового цвета и занесение его в палитру рисунка.

**Синтаксис :**
int imageColorAllocate(int im, int red, int green, int blue)

Функция возвращает идентификатор цвета, связанного с соответствующей тройкой RGB. Первым параметром функция требует идентификатор изображения, загруженного в память или созданного до этого.
Параметры *red*, *green* и *blue* задают красный, зеленый и синий компоненты цвета соответственно. Значения этих параметров должны лежать в пределах от 0 до 255, или от 0x00 до 0xFF.
Практически каждый цвет, который планируется в дальнейшем использовать, должен быть получен (определен) при помощи вызова этой функции.

**Пример:**
Пример использования функции imageColorAllocate():

```
<?php
. . .
// белый
$white = imagecolorallocate($im, 255, 255, 255);
$white = imagecolorallocate($im, 0xFF, 0xFF, 0xFF);
// черный
$black = imagecolorallocate($im, 0, 0, 0);
$black = imagecolorallocate($im, 0x00, 0x00, 0x00);
. . .
?>
```

---

**imageColorDeAllocate**
Исключение цвета из палитры рисунка.

**Синтаксис :**
int imageColorDeAllocate(int im, int color)

Эта функция удаляет из палитры рисунка *im* цвет *color*, который был предварительно занесен в рисунок функцией **imageColorAllocate()**.

**Пример:**
Пример использования функции imageColorDeAllocate():

```
<?php
. . .
$white = imageColorAllocate ($im, 255, 255, 255);
imageColorDeAllocate ($im, $white);
. . .
?>
```

---

**imageColorSet**
Замена цвета определенного элемента палитры.

**Синтаксис :**
bool imageColorSet(int im, int index, int red, int green, int blue)

Данная функция устанавливает для элемента палитры *index* рисунка *im* значения компонент цвета: *red* (красный), *green* (зеленый), *blue* (синий). При этом все части рисунка, закрашенные данным цветом, также изменят свой оттенок.

---

**imageColorClosest**
Получение цвета палитры, наиболее близкого к указанному.

**Синтаксис :**
int imageColorClosest(int im, int red, int green, int blue)

Вместо того, чтобы пытаться выискать свободное место в палитре цветов, эта функция просто возвращает идентификатор цвета, уже существующего в рисунке и находящегося ближе всего к затребованному. Таким образом, нового цвета в палитру не добавляется. Если палитра не велика, то функция может вернуть не совсем тот цвет, который вы ожидаете. Например, в палитре из трех цветов "красный-зеленый-синий" на запрос желтого цвета будет, скорее всего, возвращен идентификатор зеленого - он ближе всего с точки зрения GD соответствует понятию "зеленый".

---

**imageColorTransparent**
Определение цвета прозрачности.

**Синтаксис :**
int imageColorTransparent(int im [,int color])

Эта функция указывает GD, что соответствующий цвет *color* (заданный своим идентификатором при помощи функции imageColorAllocate()) в изображении *im* (*im* - идентификатор изображения, заданный функцией imageCreate()) должен обозначиться как прозрачный. Возвращает идентификатор установленного до этого прозрачного цвета, либо false, если таковой не был определен ранее. Надо отметить, что не все форматы поддерживают задание прозрачного цвета - например, JPEG не может его содержать.

---

**imageColorsForIndex**
Получение RGB-составляющих элемента палитры.

**Синтаксис :**
array imageColorsForIndex(int im, int index)

Функция возвращает ассоциативный массив с ключами *red, green, blue* (именно в таком порядке), которым соответствуют значения, равные величинам компонент RGB в идентификаторе цвета *index*. Но мы можем и не обращать особого внимания на ключи и преобразовать возвращенное значение как список:

```
<?php
. . .
$color=imageColorAt($im,0,0);
list($r,$g,$b)=array_values(imageColorsForIndex($im,$color));
echo "R=$r, g=$g, b=$b";
. . .
```

```
?>
```

---

**imageColorAt**
Возвращает индекс цвета точки.

**Синтаксис :**
int imageColorAt(int im, int x, int y)

Эта функция возвращает цвет точки, расположенной на координатах (*x, y*).
Если PHP скомпилирован с GD library 2.0 или выше, а картинка - truecolor, то эта функция возвратит идентификатор цвета, а не его RGB-представление.

```
<?php
$im = imageCreateFromPng("file.png");
$rgb = ImageColorAt($im, 100, 100);
$r = ($rgb >> 16) & 0xFF;
$g = ($rgb >> 8) & 0xFF;
$b = $rgb & 0xFF;
?>
```

---

**imageColorsTotal**
Получение количества цветов в палитре.

**Синтаксис :**
int imageColorsTotal(int im)

Функция возвращает число цветов в палитре указанного изображения.

---

**imageColorExact**
Получение индекса цвета палитры.

**Синтаксис :**
int imageColorExact(int im, int red, int green, int blue)

Функция возвращает индекс указанного цвета (*red, green, blue*) в палитре изображения *im*.
Функция возвратит -1, если указанного цвета нет в палитре изображения.

---

**imageColorResolve**
Нахождение или создание указанного цвета.

**Синтаксис :**
int imageColorResolve(int im, int red, int green, int blue)

Функция возвращает индекс указанного цвета (*red, green, blue*) в палитре изображения *im*.
В случае, если такой цвет в палитре отсутствует, то он создается.

---

**imageGammaCorrect**
Применение гамма-коррекции рисунка.

**Синтаксис :**
int imageGammaCorrect(int im, double inputgamma, double outputgamma)

Эта функция производит исправления гаммы к изображению, заданному дескриптором *im*.
Параметр *inputgamma* задает входную гамму, а *outputgamma* - гамму вывода.

---

## Работа с изображениями и библиотека GD : Графические примитивы

**imageSetPixel**
Рисует пиксель.

**Синтаксис :**
int imageSetPixel(int im, int x, int y, int color)

Выводит один пиксель цвета *color* в изображении *im*, расположенный в точке (*x, y*).

---

**imageLine**
Рисует сплошную тонкую линию.

**Синтаксис :**
int imageLine(int im, int x1, int y1, int x2, int y2, int color)

Эта функция рисует сплошную тонкую линию в изображении *im*, проходящую через точки (*x1, y1*) и (*x2, y2*), цветом *color*. Линия получается слабо связанной.

```
<?php

function imagelinethick($image, $x1, $y1, $x2, $y2, $color, $thick = 1)
{
    /* this way it works well only for orthogonal lines
    imagesetthickness($image, $thick);
    return imageline($image, $x1, $y1, $x2, $y2, $color);
    */
    if ($thick == 1) {
        return imageline($image, $x1, $y1, $x2, $y2, $color);
    }
    $t = $thick / 2 - 0.5;
    if ($x1 == $x2 || $y1 == $y2) {
        return imagefilledrectangle($image,
         round(min($x1, $x2) - $t),
         round(min($y1, $y2) - $t),
         round(max($x1, $x2) + $t),
         round(max($y1, $y2) + $t), $color);
    }
    $k = ($y2 - $y1) / ($x2 - $x1); //y = kx + q
    $a = $t / sqrt(1 + pow($k, 2));
    $points = array(
        round($x1 - (1+$k)*$a), round($y1 + (1-$k)*$a),
        round($x1 - (1-$k)*$a), round($y1 - (1+$k)*$a),
        round($x2 + (1+$k)*$a), round($y2 - (1-$k)*$a),
        round($x2 + (1-$k)*$a), round($y2 + (1+$k)*$a),
    );
    imagefilledpolygon($image, $points, 4, $color);
    return imagepolygon($image, $points, 4, $color);
};
?>
```

**imageDashedLine**
Рисует пунктирную линию.

**Синтаксис :**
int imageDashedLine(int im, int x1, int y1, int x2, int y2, int color)

Эта функция работает почти так же, как и **imageLine()**, только рисует не сплошную, а пунктирную линию. К сожалению, ни размер, ни шаг штрихов задавать нельзя, так что, если вам обязательна нужна пунктирная линия произвольной фактуры, придется заняться математическими расчетами и использовать **imageLine()**.

**imageRectangle**
Рисует прямоугольник.

**Синтаксис :**
int imageRectangle(int im, int x1, int y1, int x2, int y2, int color)

Эта функция рисует в изображении *im* прямоугольник с границей толщиной 1 пиксель цветом *color*.
Левый верхний угол задается (*x1, y1*), а правый нижний - (*x2, y2*).

**imageFilledRectangle**
Зарисовка прямоугольной области.

**Синтаксис :**
int imageFilledRectangle(int im, int x1, int y1, int x2, int y2, int color)

Эта функция рисует закрашенный прямоугольник в изображении, заданном идентификатором *im*, цветом *color* (полученным, например, при помощи функции **imageColorAllocate()**). Координаты (*x1, y1*) и (*x2, y2*) задают координаты верхнего левого и правого нижнего углов, соответственно (отсчет, как обычно, начинается с верхнего угла и идет слева направо и сверху вниз).

Эта функция часто применяется для того, чтобы целиком закрасить только что созданный рисунок, например, прозрачным цветом:

```
<?php
$im=imageCreate(100,100);
$color=imageColorAllocate($i,0,0,0);
imageColorTransparent($im,$color);
imageFilledRectangle($im,0,0,imageSX($im)-1,imageSY($im)-1,$color);
// дальше работаем с изначально прозрачным фоном
?>
```

**imageArc**
Рисование части эллипса.

**Синтаксис :**
int imageArc(int im, int cx, int cy, int w, int h, int s, int e, int color)

Эта функция рисует в изображении *im* дугу сектора эллипса от угла *s* до *e* (углы указываются в градусах против часовой стрелки, отсчитываемых от горизонтали).

Эллипс рисуется такого размера, чтобы вписываться в прямоугольник (*w, h*), где *w* и *h* задают его ширину и высоту. *cx* и *cy* - координаты центра эллипса. Сама фигура не закрашивается, обводится только ее контур, для чего используется цвет *color*.

```
<?php
// создаем изображение размером 200x200
$img = imagecreate(200, 200);
// задаем цвет окружности
$white = imagecolorallocate($img, 255, 255, 255);
// рисуем окружность
imagearc($img, 100, 100, 150, 150, 0, 360, $white);
// вывод рисунка в браузер
header("Content-type: image/png");
imagepng($img);
// закрытие рисунка
imagedestroy($img);
?>
```

---

**imageFill**
Заливка цветом ограниченной области.

**Синтаксис :**
int imageFill(int im, int x, int y, int color)

Эта функция выполняет сплошную заливку одноцветной области, содержащей точку с координатами (*x, y*) цветом *color*. Нужно замерить, что современные алгоритмы заполнения работают довольно эффективно, так что не стоит особо заботиться о скорости ее работы. Будут закрашены только те точки, к которым можно проложить "одноцветный сильно связанный путь" из точки *x, y*.

Две точки называются связанными сильно, если у них совпадает, по крайней мере, одна координата, а по другой координате они отличаются не более, чем на 1 в любую сторону.

---

**imageFillToBorder**
Заливка области, ограниченной бордюром.

**Синтаксис :**
int imageFillToBorder(int im, int x, int y, int border, int color)

Эта функция очень похожа на **imageFill()**, только она выполняет закраску цветом *color* не одноцветных точек, а любых, но до тех пор, пока не будет достигнута граница цвета *border*.

---

**imagePolygon**
Рисует многоугольник с заданными вершинами.

**Синтаксис :**
int imagePolygon(int im, array points, int num_points, int color)

Эта функция рисует в изображении *im* многоугольник, заданный своими вершинами. Координаты углов передаются в массиве *points*, причем *$points[0]=x0, $points[1]=y0, $points[2]=x1, $points[3]=y1* и т.д.
Параметр *num_points* указывает общее число вершин - на тот случай, если в

массиве их больше, чем нужно нарисовать. Многоугольник не закрашивается - только рисуется его граница цветом *color*.

```
<?php
// создаем изображение 400x300
$image = imagecreate(400, 300);

// устанавливаем цвет бордюра многоугольника
$col_poly = imagecolorallocate($image, 255, 255, 255);

// рисуем многоугольник
imagepolygon($image,
             array (
                    0, 0,
                    100, 200,
                    300, 200
             ),
             3, $col_poly);

// вывод картинки в браузер
header("Content-type: image/png");
imagepng($image);
?>
```

---

**imageFilledPolygon**
Рисует закрашенный многоугольник с заданными вершинами.

**Синтаксис :**
int imageFilledPolygon(int im, array points, int num_points, int color)

Эта функция делает практически то же самое, что и **imagePolygon()**, за исключением одного очень важного свойства: полученный многоугольник целиком заливается цветом *color*.
При этом правильно обрабатываются вогнутые части фигуры, если она не выпукла.

```
<?php
// задаем массив с координатами углов
$values = array(
  0  => 40,    // x1
  1  => 50,    // y1
  2  => 20,    // x2
  3  => 240,   // y2
  4  => 60,    // x3
  5  => 60,    // y3
  6  => 240,   // x4
  7  => 20,    // y4
  8  => 50,    // x5
  9  => 40,    // y5
  10 => 10,    // x6
  11 => 10,    // y6
);

// создаем картинку 250x250
$im = imagecreate(250, 250);

// задаем цвет заполнения многоугольника
$blue = imagecolorallocate($im, 0, 0, 255);

// рисуем многоугольник
imagefilledpolygon($im, $values, 6, $blue );
```

```
// вывод каринки в браузер и ее закрытие
header('Content-type: image/png');
imagepng($im);
imagedestroy($im);
?>
```

---

## Работа с изображениями и библиотека GD : Работа с фиксированными шрифтами

Библиотека GD имеет некоторые возможности по работе с текстом и шрифтами. Шрифты представляют собой специальные ресурсы, имеющие собственный идентификатор, и чаще всего загружаемые из файла или встроенные в GD. Каждый символ шрифта может быть отображен лишь в моноцветном режиме, т.е. "рисованные" символы не поддерживаются. Встроенных шрифтов всего 5 (идентификаторы от 1 до 5), чаще всего в них входят моноширные символы разных размеров. Остальные шрифты должны быть предварительно загружены.

---

**imageLoadFont**
Загрузка шрифта.

**Синтаксис :**
int imageLoadFont(string file)

Функция загружает файл шрифтов *file* и возвращает идентификатор шрифта - это будет цифра, большая 5, потому что пять первых номеров зарезервировано как встроенные. Формат файла - бинарный, а потому зависит от архитектуры машины. Это значит, что файл со шрифтами должен быть сгенерирован по крайней мере на машине с процессором такой же архитектуры, как и у той, на котором вы собираетесь использовать PHP.

**Формат файла со шрифтом**

| Смещение | Тип | Описание |
|---|---|---|
| Byte 0-3 | long | Число символов в шрифте (nchars) |
| byte 4-7 | long | Индекс первого символа шрифта (обычно 32 - пробел) |
| byte 8-11 | long | Ширина (в пикселях) каждого знака (width) |
| byte 12-15 | long | Высота (в пикселях) каждого знака (height) |
| byte 16-... | array | Массив с информацией о начертании каждого символа, по одному байту на пиксел. На один символ, таким образом, приходится width*height*nchars байтов. 0 означает отсутствие точки в данной позиции, все остальное - ее присутствие. |

Левая колонка задает смещение начала данных внутри файла, а группами цифр, записанных через дефис, определяется, до какого адреса продолжаются данные.

---

**imageFontHeight**
Установка высоты шрифта.

**Синтаксис :**
int imageFontHeight(int font)

Функция возвращает высоту в пикселях символов в заданном шрифте.

---

**imageFontWidth**
Установка ширины шрифта.

**Синтаксис :**
int imageFontWidth(int font)

Функция возвращает ширину в пикселях символов в заданном шрифте.

---

**imageString**
Выводит строку в горизонтальном направлении.

**Синтаксис :**
int imageString(int im, int font, int x, int y, string s, int color)

Функция выводит строку *s* в изображение *im*, используя шрифт *font* и цвет *color*. Координаты (*x, y*) будут координатами верхнего левого угла прямоугольника, в который вписана строка.

Если параметр *font* задан как 1, 2, 3, 4 или 5, то выводится шрифт соответствующего размера.

```
<?php
  // создаем изображение 100x30
  $im = imagecreate(100, 30);

  // задаем цвет текста
  $textcolor = imagecolorallocate($im, 0, 0, 255);

  // выводим надпись в верхнем левом углу
  imagestring($im, 5, 0, 0, "Hello world!", $textcolor);

  // выводим изображение в браузер
  header("Content-type: image/jpg");
  imagejpeg($im);
?>
```

---

**imageStringUp**
Выводит строку в вертикальном направлении.

**Синтаксис :**
int imageStringUp(int im, int font, int x, int y, string s, int color)

Эта функция также выводит строку текста, но не в горизонтальном, а в вертикальном направлении.
Верхний левый угол задается координатами (*x, y*).

Если параметр *font* задан как 1, 2, 3, 4 или 5, то выводится шрифт соответствующего размера.

---

**imageChar**
Вывод символа горизонтально.

**Синтаксис :**
int imageChar(int im, int font, int x, int y, string c, int color)

Функция выводит символ *c* в горизонтальном положении в месте на рисунке, заданном координатами (*x, y*). Шрифт символа задается параметром *font*. Если этот параметр принимает значение от 1 до 5, то используются встроенные шрифты. Цвет символа задается параметром *color*.

```
<?php
// создаем картинку размером 100x100
$im = imagecreate(100, 100);

$string = "PHP";

// задаем цвет символа
$black = imagecolorallocate($im, 0, 0, 0);

// Выведем символ "P" в верхнем левом углу
imagechar($im, 1, 0, 0, $string, $black);

// выведем картинку в браузер
header("Content-type: image/png");
imagepng($im);
?>
```

---

**imageCharUp**
Вывод символа вертикально.

**Синтаксис :**
int imageCharUp(int im, int font, int x, int y, string c, int color)

Функция выводит символ *c* в вертикальном положении в месте на рисунке, заданном координатами (*x, y*). Шрифт символа задается параметром *font*. Если этот параметр принимает значение от 1 до 5, то используются встроенные шрифты. Цвет символа задается параметром *color*.

---

## Работа с изображениями и библиотека GD : Работа со шрифтами TrueType и PostScript Type 1

Библиотека GD поддерживает также работу со шрифтами PostScript и TrueType. Для того чтобы заработали приведенные ниже функции, PHP должен быть откомпилирован и установлен вместе с библиотекой FreeType, доступной по адресу http://www.freetype.org. В Windows-версии PHP она установлена по умолчанию.

---

**imageTTFText**
Рисование текста шрифтом TrueType.

**Синтаксис :**
array imageTTFText(int im, int size, int angle, int x, int y, int color, string fontfile, string text)

Эта функция помещает строку *text* в изображение *im* цветом *color*. Как обычно, *color* должен представлять собой допустимый идентификатор цвета. Параметр *angle* задает угол наклона в градусах выводимой строки, отсчитываемой от горизонтали против часовой стрелки. Координаты (*x, y*) указывают положение так называемой *базовой точки строки* (обычно это ее левый нижний угол).

Параметр *size* задает размер шрифта, который будет использоваться при выводе строки. *fontfile* должен содержать имя TTF-файла, в котором и храниться шрифт.

Функция возвращает список из 8 элементов. Первая их пара задает координаты (x,y) верхнего левого угла прямоугольника, описанного вокруг строки текста в изображении, вторая пара - координаты верхнего правого угла, и т.д. Так как в общем случае строка может иметь любой наклон *angle*, здесь требуются 4 пары координат.

Строка текста *text* может содержать символьные последовательности UTF-8 (в виде {) для вывода символов с кодами, большими 255.

При использовании отрицательного значения индекса цвета *color* отключается сглаживание шрифта (antialiasing).

Данная функция требует библиотеке GD и FreeType.

```
<?php
  header("Content-type: image/jpeg");
  $im = imagecreate(400, 30);
  $white = imagecolorallocate($im, 255, 255, 255);
  $black = imagecolorallocate($im, 0, 0, 0);

  // Replace path by your own font path
  imagettftext($im, 20, 0, 10, 20, $black, "/path/arial.ttf",
  "Testing... Omega: &#937;");
  imagejpeg($im);
  imagedestroy($im);
?>
```

Следующий пример выводит строку по центру рисунка

```
<?php
$gi = imageCreate(200,100);
$bg = imageColorAllocate($gi,0,220,0);
$tx = imageColorAllocate($gi,25,2,228);
$w = imageSX($gi); // ширина рисунка
$h = imageSY($gi); // высота рисунка
imageFilledRectangle($gi,0,0,$w,$h,$bg);

$szf = 20; // размер шрифта
$ang = 240; // угол поворота строки
$str = "Heyou"; // текст строки
$font = "symbol.ttf" // файл шрифта
$sz = imageTTFBBox($szf,$ang,$font,$str);
$sdx = $sz[4]/2;
$sdy = ($sz[7]+$sz[3])/2;
imageTTFText($gi,$szf,$ang,$w/2-$sdx,$h/2-$sdy,$tx,$font,$str);
Header("Content-Type: image/png");
imagePng($gi,"file.png");
?>
```

---

**imageTTFBBox**
Расчет площади, занимаемой строкой шрифта TrueType.

**Синтаксис :**
attay imageTTFBBox(int size, int angle, string fontfile, string text)

Эта функция ничего не выводит в изображение, а просто определяет, какой размер и положение заняла бы строка текста *text* размера *size*, выведенная под

углом *angle* в какой-нибудь рисунок. Параметр *fontfile* задает абсолютный путь к файлу шрифта, который будет использован при выводе.

Возвращаемый список содержит всю информацию о размерах строки в формате, похожем на тот, что выдает функция **imageTTFText()**. Однако порядок точек в нем отличается.

**Содержимое массива, возвращаемого функцией imageTTFBBox():**

0 и 1 - (x,y) левого нижнего угла
2 и 3 - (x,y) правого нижнего угла
4 и 5 - (x,y) правого верхнего угла
6 и 7 - (x,y) левого верхнего угла

Координаты могут иметь отрицательные значения.
Функция требует библиотеки GD и FreeType.

---

**imagePSLoadFont**
Загрузка из файла шрифта PostScript Type 1.

**Синтаксис :**
int imagePSLoadFont(string filename)

Возвращает дескриптор загруженного шрифта или FALSE при ошибке (также выводится предупреждение).

```
<?php
header("Content-type: image/jpeg");
$im = imagecreate(350, 45);
$black = imagecolorallocate($im, 0, 0, 0);
$white = imagecolorallocate($im, 255, 255, 255);
$font = imagepsloadfont("bchbi.pfb"); // or locate your .pfb files on your
machine
imagepstext($im, "Testing... It worked!", $font, 32, $white, $black, 32,
32);
imagepsfreefont($font);
imagejpeg($im, "", 100); //for best quality...your mileage may vary
imagedestroy($im);
?>
```

Эта функция доступна только в том случае, если PHP был скомпилирован с опцией --enable-t1lib.

---

**imagePSFreeFont**
Выгрузка шрифта PostScript Type 1.

**Синтаксис :**
void imagePSFreeFont(int fontindex)

Данная функция освобождает память от шрифта, заданного параметром *fontindex*.
Эта функция доступна только в том случае, если PHP был скомпилирован с опцией --enable-t1lib.

---

**imagePSEncodeFont**
Установка схемы перекодировки текста.

**Синтаксис :**
int imagePSEncodeFont(int font_ndex, string encodingfile)

Загружает файл перекодировки *encodingfile* для шрифта *font_index*. Поскольку шрифты PostScript по умолчанию не используют символы с кодами, большими 127, перекодировка требуется при необходимости использования не английского языка. Формат файла описан в документации Tllibs, также с библиотекой поставляются 2 готовых файла: IsoLatinl.enc и IsoL.atin2.enc.

Если перекодировка используется постоянно, установите параметр ps.default_encoding в файле конфигурации со значением имени файла перекодировки, который будет загружаться автоматически.

Эта функция доступна только в том случае, если PHP был скомпилирован с опцией --enable-t1lib.

---

**imagePsExtendFont**
Масштабирование шрифта.

**Синтаксис :**
bool imagePsExtendFont (int font_index, float extend)

Функция производит растяжение или сжатие шрифта, заданного параметром *font_index* до размера, заданного параметром *extend*.
Если значение параметра *extend* меньше 1, то шрифт будет уменьшаться.

Эта функция доступна только в том случае, если PHP был скомпилирован с опцией --enable-t1lib.

---

**imagePsSlantFont**
Установка наклона шрифта.

**Синтаксис :**
bool imagePsSlantFont(int font_index, double slant)

Фукнция устанавливает наклон шрифта *font_index* в значение, заданное параметром *slant*.

Эта функция доступна только в том случае, если PHP был скомпилирован с опцией --enable-t1lib.

---

**imagePSBBox**
Расчет площади, занимаемой строкой шрифта PostScript Type 1.

**Синтаксис :**
array imagePSBBox( string text, int font, int size [, int space [, int tightness [, float angle]]])

Расчеты производятся на основании аргументов:
*size* - размер шрифта в пикселах;
*space* — изменение размера пробелов по отношению к нормальному (может быть отрицательным);

*tightness* — промежутки между символами по отношении) к нормальному (может быть отрицательным);
*angle* — угол наклона строки в градусах.

Значения *space* и *tightness* измеряются в долях пробела (1/1000).
Аргументы *space*, *tightness*, *angle* не обязательны.

Результаты расчета недостаточно точны. Функция возвращает массив:

- 0 - нижний левый угол, X-координата;
- 1 - нижний левый угол, Y-координата;
- 2 - верхний правый угол, X-координата;
- 3 - верхний правый угол, Y- координата.

Эта функция доступна только в том случае, если PHP был скомпилирован с опцией --enable-t1lib.

---

**imagePSText**
Вывод текста поверх рисунка шрифтом PostScript Type 1.

**Синтаксис :**
array imagePSText ( resource image, string text, int font, int size, int foreground, int background, int x, int y [, int space [, int tightness [, float angle [, int antialias_steps]]]])

Параметр *size* задает размер шрифта.
Координаты *x*, *y* указывают левый нижний угол первого символа.
Аргументами *foreground* и *background* задаются цвета текста и фона (фон необходим только для сглаживания шрифта).
Аргумент *antialias_steps* позволяет указать число цветов, используемых при сглаживании текста (допустимые значения 4 и 16). Для шрифтов размером меньше 20 используйте большее значение, так как это улучшает читабельность; для больших шрифтов используйте меньшее значение, так как это увеличивает быстродействие.
Параметр *angle* задает наклон текста в градусах.
Функция возвращает массив, подобно **imagepsbbox()**.

Эта функция доступна только в том случае, если PHP был скомпилирован с опцией --enable-t1lib.

---

## PDF-документы : Введение

PDF-функции позволяют PHP создавать PDF-файлы с помощью библиотеки PDF, созданной Томасом Мерзем (http://www.pdflib.com/pdflib/index.html); также могут потребоваться библиотеки JPEG (ftp://ftp.uu.net/graphics/jpeg/) и TIFF (http://www.libtiff.org/).

С pdflib поставляется хорошая документация, описывающая возможности библиотеки. Имена функций и аргументы идентичны в библиотеке и PHP. Размеры и координаты измеряются в единицах Postscript (72 на дюйм), но это зависит от выбранного разрешения.

Аналогом библиотеки является ClibPDF.

Версии ниже 3.0 pdflib не поддерживается в PHP 4.

```
<?php
$fp = fopen("test.pdf". "w");
$pdf = pdf_open($fp);
pdf_set_info($pdf, "Author", "Uwe Streinmann");
pdf_set_info($pdf, "Title", "Test for PHP PDFlib");
pdf_set_info($pdf, "Creator", "See Author");
pdf_set_info($pdf, "Subject", "Testing");
pdf_begin_page($pdf, 595, 842);
pdf_add_outline($pdf, "Page 1");
pdf_set_font("$pdf, "Times-Roman", 30, "host");
pdf_set_value($pdf, "textrendering", 1);
pdf_show_xy($pdf, "Times Roman outlined", 50, 750);
pdf_moveto($pdf, 50, 740);
pdf_lineto($pdf, 330, 740);
pdf_stroke($pdf);
pdf_end_page($pdf);
pdf_close($pdf);
fclose($fp);
echo "<A href=getpdf.php>finished</A>";
?>
<?php
// Сценарий getpdf.php просто возвращает документ pdf
$fp = fopen("test.pdf", "r");
header("Content-type: application/pdf");
fpassthru($fp);
fclose($fp);
?>
```

---

## PDF-документы : Открытие документа

**pdf_set_info**
Заполнение поля информации документа.

**Синтаксис :**
void pdf_set_info(int pdf_document, string fieldname, string value)

Возможные поля *fieldname*:
- Subject
- Title
- Creator
- Author
- Keywords
- Одно, определяемое пользователем.

Функция должна вызываться до создания страниц.

```
<?php
$fd = fopen("test.pdf", "w");
$pdfdoc = pdf_open($fd);
pdf_set_info($pdfdoc, "Author", "Имя автора");
pdf_set_info($pdfdoc, "Creator", "Название создателя");
pdf_set_info($pdfdoc, "Title", "Заголовок");
pdf_set_info($pdfdoc, "Subject", "Тема");
pdf_set_info($pdfdoc, "Kewwords", "Ключевые, слова");
pdf_set_info($pdfdoc, "CustomField", "Чтото еще");
pdf_begin_page($pdfdoc, 595, 842);
pdf_end_page($pdfdoc);
```

```
pdf_close($pdfdoc);
?>
```
Эта функция заменяет собой pdf_set_info_keyword(), pdf_set_info_title(), pdf_set_info_subject(), pdf_set_info_creator().

---

**pdf_open**
Открытие нового документа pdf.

**Синтаксис :**
int pdf_open(int file)

Функция делает файл, открытый функцией fopen(), документом pdf. Если не указывать дескриптор файла, он создается в памяти и затем может выводиться на стандартный поток вывода или отсылаться броузеру. Функция возвращает дескриптор документа, который следует указывать в последующих pdf-функциях.

---

**pdf_close**
Закрытие документа pdf.

**Синтаксис :**
void pdf_close(int pdf_document)

---

**pdf_begin_page**
Начало новой страницы.

**Синтаксис :**
void pdf_begin_page(int pdf_document, double width, double height)

Аргументы *height* и *width* задают высоту и ширину страницы. После внесения на страницу информации ее следует закрыть функцией pdf_end_page().

---

**pdf_end_page**
Завершение страницы.

**Синтаксис :**
void pdf_end_page(int pdf_document)

После этой функции модификация этой страницы невозможна.

---

## PDF-документы : Работа с текстом

**pdf_show**
Вывод текста в текущую позицию.

**Синтаксис :**
void pdf_show(int pdf_document, string text)

Для вывода используются текущая позиция и текущий шрифт.

---

**pdf_show_boxed**
Вывод текста в прямоугольную область.

**Синтаксис :**
void pdf_show_boxed(int pdf_document, string text, double x, double y, double width, double height, string mode [, string feature])

Левый нижний угол области вывода задается (*x*:*y*); высота и ширина - *height*,*width*. Аргумент *mode* определяет выравнивание текста: если высота и ширина равны нулю, то возможны значения:
- left
- right
- center,

если они не равны нулю, то
- justify
- fulljustify

Если аргумент *feature* содержит значение "blind", текст не отображается.

Функция возвращает число символов, которые не поместились в указанный прямоугольник.

---

**pdf_show_xy**
Вывод текста в указанную позицию.

**Синтаксис :**
void pdf_show_xy(int pdf_document, string text, double x, double y)

---

**pdf_set_font**
Выбор шрифта, его размера и кодировки.

**Синтаксис :**
void pdf_set_font(int pdf_document, string font_name, double size, string encoding [, int embed])

Аргумент вида кодировки *encoding* может принимать значения:
- winansi (по умолчанию)
- builtin
- host
- macroman и т.д.

Если для последнего аргумента задано значение 1, шрифт будет внедрен в документ pdf (иначе нет). Если шрифт распространен, внедрять его не следует из-за увеличения размера документа.

Функция должна вызываться после pdf_begin_page().

---

**pdf_set_leading**
Установка промежутка между строками текста.

**Синтаксис :**
void pdf_set_leading(int pdf_document, double distance)

Используется при выводе текста функцией pdf_continue_text().

---

**pdf_set_parameter**
Установка строкового значения параметра pdflib.

**Синтаксис :**
void pdf_set_parameter(int pdf_document, string name, string value)

---

**pdf_get_parameter**
Получение строкового значения параметра pdflib.

**Синтаксис :**
void pdf_get_parameter(int pdf_document, string name [, double modifier])

Аргумент *modifier* используется при необходимости.

---

**pdf_set_value**
Установка численного значения параметра pdflib.

**Синтаксис :**
void pdf_set_value(int pdf_document, string name, double value)

---

**pdf_get_value**
Получение численного значения параметра pdflib.

**Синтаксис :**
void pdf_get_value(int pdf_document, string name [, double modifier])

Аргумент *modifier* используется при необходимости.

---

**pdf_set_text_rendering**
Установка метода вывода текста.

**Синтаксис :**
void pdf_set_text_rendering(int pdf_document, string mode)

Устарела, используйте pdf_set_value().

---

**pdf_set_horiz_scaling**
Установка масштабирования текста по горизонтали.

**Синтаксис :**
void pdf_set_horiz_scaling(int pdf_document, double scale)

---

**pdf_set_text_rise**
Установка подъема текста.

**Синтаксис :**
void pdf_set_text_rise(int pdf_document, double rise)

---

**pdf_set_text_matrix**
Установка матрицы преобразований шрифта.

**Синтаксис :**
void pdf_set_text_matrix(int pdf_document, array matrix)

Начиная с версии pdflib 2.3 эта функция недоступна.

---

**pdf_set_text_pos**
Установка позиции шрифта.

**Синтаксис :**
void pdf_set_text_pos(int pdf_document, double x-coor, double y-coor)

Устанавливает позицию вывода текста последующим вызовом pdf_show().

---

**pdf_set_char_spacing**
Установка интервала между символами.

**Синтаксис :**
void pdf_set_char_spacing(int pdf_document, double space)

Устарела, используйте pdf_set_value().

---

**pdf_set_word_spacing**
Установка интервала между символами.

**Синтаксис :**
void pdf_set_word_spacing(int pdf_document, double space)

Устарела, используйте pdf_set_value().

---

**pdf_skew**
Поворот системы координат.

**Синтаксис :**
void pdf_skew(int pdf_document, double alpha, double beta)

Угол поворота в градусах указывается относительно осей alpha (x) и beta (y). Углы не могут принимать значения 90 или 270 градусов.

---

**pdf_continue_text**
Вывод текста со следующей строки.

**Синтаксис :**
void pdf_continue_text(int pdf_document, string text)

Расстояние между строками может быть установлено функцией pdf_set_leading().

---

**pdf_stringwidth**
Вычисление ширины текста.

**Синтаксис :**
void pdf_stringwidth(int pdf_document, string text)

При вычислении длины строки используется текущий шрифт. Предварительно шрифт должен быть установлен с помощью pdf_set_font().

---

**pdf_save**
Сохранение текущих установок.

**Синтаксис :**
void pdf_save(int pdf_document)

Действует подобно команде postscript gsave. Полезна при необходимости масштабировать или развернуть объект, не воздействуя на другие объекты. pdf_save() требует, чтобы затем была вызвана функция pdf_restore().

---

**pdf_restore**
Восстановление ранее сохраненных установок.

**Синтаксис :**
void pdf_restore(int pdf_document)

Восстанавливает установки, сохраненные pdf_save(). Действует подобно команде postscript grestore.

```
<?php
pdf_save($pdf);
// всякие вращения и трансформации ...
pdf_restore($pdf);
?>
```

---

## PDF-документы : Установка масштаба и системы координат

**pdf_translate**
Установка начала системы координат.

**Синтаксис :**
void pdf_translate(int pdf_document, double x, double y)

Координаты указываются относительно текущей точки отсчета. Затем, до начала рисования объектов, требуется установить текущую точку.

```
<?php
pdf_moveto($pdf, 0, 0);
pdf_lineto($pdf, 100, 100);
pdf_stroke($pdf);
psd_translate($pdf, 100, 100);
pdf_moveto($pdf, 0, 0);
pdf_lineto($pdf, 100, 100);
pdf_stroke($pdf);
?>
```

---

**pdf_scale**
Установка масштабирования.

**Синтаксис :**
void pdf_scale(int pdf_document, double x_scale, double y_scale)

```
<?php
pdf_scale($pdf, 72.0, 72.0);
pdf_lineto($pdf, 1, 1); // на дюйм
pdf_stroke($pdf);
?>
```

---

**pdf_rotate**
Установка угла вращения в градусах.

**Синтаксис :**
void pdf_rotate(int pdf_document, double angle)

---

**pdf_setflat**
Установка равномерности.

**Синтаксис :**
void pdf_setflat(int pdf_document, double value)

Возможные значения параметра - от 0 до 100.

---

**pdf_setlinejoin**
Установка параметра linejoin.

**Синтаксис :**
void pdf_setlinejoin(int pdf_document, double value)

Возможные значения параметра - от 0 до 2.

---

**pdf_setlinecap**
Установка параметра linecap.

**Синтаксис :**
void pdf_setlinecap(int pdf_document, double value)

Возможные значения параметра - от 0 до 2.

---

**pdf_setmiterlimit**
Установка параметра miter limit.

**Синтаксис :**
void pdf_miterlimit(int pdf_document, double value)

Возможные значения параметра - 1 и более.

---

**pdf_setlinewidth**
Установка ширины строк.

**Синтаксис :**
void pdf_setlinewidth(int pdf_document, double width)

---

**pdf_setdash**
Установка текущей точки.

**Синтаксис :**
void pdf_setdash(int pdf_document, double white, double black)

---

**pdf_moveto**
Установка текущей точки.

**Синтаксис :**
void pdf_moveto(int pdf_document, double x, double y)

---

## PDF-документы : Черчение и заполнение фигур

**pdf_curveto**
Черчение кривой.

**Синтаксис :**
void pdf_curveto(int pdf_document, double x1, double y1, double x2, double y2, double x3, double y3)

Чертит кривую Безье от текущей точки до (x3,y3), использую точки (x1,y1) и (x2,y2) как ориентирующие.

---

**pdf_lineto**
Черчение отрезка.

**Синтаксис :**
void pdf_lineto(int pdf_document, double x, double y)

Чертит линию от текущей точки до указанной (x,y).

---

**pdf_circle**
Черчение окружности.

**Синтаксис :**
void pdf_circle(int pdf_document, double x, double y, double radius)

---

**pdf_arc**
Черчение дуги.

**Синтаксис :**
void pdf_arc(int pdf_document, double x, double y, double radius, double start, double end)

Начальный и конечный угол задаются в *start* и *end*.

---

**pdf_rect**
Черчение прямоугольника.

**Синтаксис :**
void pdf_rect(int pdf_document, double x, double y, double width, double height)

Левый нижний угол задается (*x*,*y*); высота и ширина - *height* и *width*.

---

**pdf_closepath**
Завершение текущего пути.

**Синтаксис :**
void pdf_closepath(int pdf_document)

Чертит линию от текущей точки до точки, где начиналась первая линия. Многие функции, например pdf_moveto(), pdf_circle(), pdf_rect() начинают новый путь.

---

**pdf_stroke**
Заштриховка пути.

**Синтаксис :**
void pdf_stroke(int pdf_document)

Текущий путь - это совокупность всех линий. Без этой функции линии начерчены не будут.

---

**pdf_closepath_stroke**
Черчение и закрытие пути.

**Синтаксис :**
void pdf_closepath_stroke(int pdf_document)

Это комбинация pdf_closepath() и pdf_stroke().

---

**pdf_fill**
Заполнение пути цветом.

**Синтаксис :**
void pdf_fill(int pdf_document)

---

**pdf_fill_stroke**
Заполнение пути цветом и закрытие его.

**Синтаксис :**
void pdf_fill_stroke(int pdf_document)

---

**pdf_closepath_fill_stroke**
Черчение, закрашивание и закрытие пути.

**Синтаксис :**
void pdf_closepath_fill_stroke(int pdf_document)

---

**pdf_endpath**
Завершение пути без его закрытия.

**Синтаксис :**
void pdf_endpath(int pdf_document)

---

**pdf_clip**
Прикрепление всех линий к текущему пути.

**Синтаксис :**
void pdf_clip(int pdf_document)

---

**pdf_setgray_fill**
Установка заполнения серым цветом.

**Синтаксис :**
void pdf_setgray_fill(int pdf_document, double gray_value)

---

**pdf_setgray_stroke**
Установка штриховки серым цветом.

**Синтаксис :**
void pdf_setgray_stroke(int pdf_document, double gray_value)

---

**pdf_setgray**
Установка заполнения и штриховки серым цветом.

**Синтаксис :**
void pdf_setgray(int pdf_document, double gray_value)

---

**pdf_setrgbcolor_fill**
Установка заполнения цветом RGB.

**Синтаксис :**
void pdf_setrgbcolor_fill(int pdf_document, double red_value, double green_value, double blue_value)

---

**pdf_setrgbcolor_stroke**
Установка штриховки цветом RGB.

**Синтаксис :**
void pdf_setrgbcolor_stroke(int pdf_document, double red_value, double green_value, double blue_value)

---

**pdf_setrgbcolor**
Установка заполнения и штриховки цветом RGB.

**Синтаксис :**
void pdf_setrgbcolor(int pdf_document, double red_value, double green_value, double blue_value)

---

**pdf_add_outline**
Добавление закладки для текущей страницы.

**Синтаксис :**
void pdf_add_outline(int pdf_document, string text [, int parent [, int open]])

Название закладки определяется аргументом *text*. Она становится дочерним объектом объекта *parent* и по умолчанию открыта (если аргумент *open* не равен 0). Возвращается идентификатор закладки, который может использоваться как родительский для других закладок.

---

**pdf_set_transition**
Установка режима перехода между страницами.

**Синтаксис :**
void pdf_set_transition(int pdf_document, int transition)

Используйте функцию pdf_set_parameter() с параметром "transition".

---

**pdf_set_duration**
Установка интервала между страницами.

**Синтаксис :**
void pdf_set_duration(int pdf_document, double duration)

---

## PDF-документы : Размещение рисунков

**pdf_open_gif**
Открытие рисунка GIF.

**Синтаксис :**
void pdf_open_gif(int pdf_document, string filename)

Используйте функцию pdf_open_image_file().

```
<?php
$im = pdf_open_gif($pdf, "test.gif");
pdf_place_image($pdf, $im, 100, 100, 1);
pdf_close_image($pdf, $im);
?>
```

---

**pdf_open_png**
Открытие рисунка PNG.

**Синтаксис :**

void pdf_open_png(int pdf_document, string filename)

Используйте функцию pdf_open_image_file().

---

**pdf_open_jpeg**
Открытие рисунка JPEG.

**Синтаксис :**
void pdf_open_jpeg(int pdf_document, string filename)

Используйте функцию pdf_open_image_file().

---

**pdf_open_tiff**
Открытие рисунка TIFF.

**Синтаксис :**
void pdf_open_tiff(int pdf_document, string filename)

Используйте функцию pdf_open_image_file().

---

**pdf_open_image_file**
Чтение рисунка из файла.

**Синтаксис :**
void pdf_open_tiff(int pdf_document, string format, string filename)

Эта функция загружает рисунок формата *format* из файла *filename* и возвращает его идентификатор.
Возможные форматы:

- PNG
- TIFF
- JPEG
- GIF

```
<?php
$pim = pdf_open_image_file($pdf, "png", "pic.png");
pdf_place_image($pdf, $pim, 100, 100, 1);
pdf_close_image($pdf, $pim);
?>
```

Эта функция заменяет pdf_open_image(), pdf_open_gif(), pdf_open_tiff(), pdf_open_png().

---

**pdf_open_memory_image**
Открытие рисунка, созданного графическими функциями PHP.

**Синтаксис :**
void pdf_open_memory_image(int pdf_document, int image)

Функция принимает дескриптор рисунка, созданного PHP, и делает его доступным для документа pdf. Функция возвращает идентификатор рисунка pdf.

```
<?php
$im = ImageCreate(100, 100);
$col = ImageColorAllocate($im, 80, 45, 190);
ImageFill($im, 10, 10, $col);
$pim = pdf_open_memory_image($pdf, $im);
ImageDestroy($im);
pdf_place_image($pdf, $pim, 100, 100, 1);
pdf_close_image($pdf, $pim);
?>
```

**pdf_close_image**
Закрытие рисунка.

**Синтаксис :**
void pdf_close_image(int pdf_document, int image)

Закрывает рисунок, открытый функциями pdf_open_().

**pdf_get_image_height**
Установка высоты рисунка в пикселах.

**Синтаксис :**
void pdf_get_image_height(int pdf_document, int image)

**pdf_get_image_width**
Установка ширины рисунка в пикселах.

**Синтаксис :**
void pdf_get_image_width(int pdf_document, int image)

**pdf_place_image**
Размещение рисунка на странице.

**Синтаксис :**
void pdf_place_image(int pdf_document, int image, double x, double y, souble scale)

Позиция размещения задается (*x*,*y*); масштаб - *scale*.

**pdf_put_image**
Сохранение рисунка в pdf для дальнейшего использования.

**Синтаксис :**
void pdf_put_image(int pdf_document, int image)

Функция внедряет рисунок в документ без его отображения. Затем рисунок может быть размещен на странице функцией pdf_execute_image() необходимое число раз. Полезно при многократной вставке рисунка (уменьшает размер файла).

Начиная с версии 2.01 pdflib функция бесполезна и выводит только предупреждение.

**pdf_execute_image**
Размещение сохраненного рисунка на странице.

**Синтаксис :**
void pdf_execute_image(int pdf_document, int image, double x, double y, double scale)

Отображает рисунок, внедренный функцией pdf_put_image().
Начиная с версии 2.01 pdflib функция бесполезна и выводит только предупреждение.

```
<?php
$im = ImageCreate(100, 100);
$col1 = ImageColorAllocate($im, 80, 45, 190);
ImageFill($im, 10, 10, $col1);
$pim = pdf_open_memory_image($pdf, $im);
```

```
pdf_put_image($pdf, $pim);
pdf_execute_image($pdf, $pim, 100, 100, 1);
// 200%
pdf_execute_image($pdf, $pim, 200, 200, 2);
pdf_close_image($pdf, $pim);
?>
```

---

## PDF-документы : Стиль документа

**pdf_set_border_style**
Установка стиля обрамления примечаний и гиперссылок.

**Синтаксис :**
void pdf_set_border_style(int pdf_document, string style, double width)

Аргумент *style* может принимать значения "solid" или "dashed". Ширина задается аргументом *width*.

---

**pdf_set_border_color**
Установка цвета обрамления примечаний и гиперссылок.

**Синтаксис :**
void pdf_set_border_color(int pdf_document, double red, double green, double blue)

Три компонента цвета могут принимать значения из диапазона от 0.0 до 1.0

---

**pdf_set_border_dash**
Установка стиля окантовки ссылок и примечаний.

**Синтаксис :**
void pdf_set_border_dash(int pdf_document, double black, double white)

Устанавливает длину черных и белых полос прерывистых линий.

---

**pdf_add_annotation**
Добавление примечания.

**Синтаксис :**
void pdf_add_annotation(int pdf_document, double llx, double lly, double urx, double ury, string title, string content)

Примечание предполагается в нижнем левом углу (*llx*, *lly*), верхний правый угол (*urx*, *ury*).

---