

# Веб-разработка с использованием PHP

---

Рабочая тетрадь

# Содержание

|  |            |
|--|------------|
| <b>ПРЕДИСЛОВИЕ .....</b>   | <b>4</b>   |
| <b>ТЕМА 1. ОСНОВЫ ВЕБ-ТЕХНОЛОГИЙ.....</b>  | <b>5</b>   |
| <b>1.1. ОБОБЩЁННАЯ СХЕМА ВЕБ-ТЕХНОЛОГИЙ .....</b>  | <b>5</b>   |
| <b>1.2. МНОГОУРОВНЕВАЯ АРХИТЕКТУРА ПРИЛОЖЕНИЙ .....</b>                                  | <b>15</b>  |
| <b>1.3. ПРИНЦИПЫ РАБОТЫ ВЕБ-ПРИЛОЖЕНИЙ.....</b>  | <b>17</b>  |
| <b>1.4. АРХИТЕКТУРА ВЕБ-ПРИЛОЖЕНИЙ.....</b>  | <b>20</b>  |
| 1.4.1. ВЕБ-ПРИЛОЖЕНИЯ В JAVA.....  | 20         |
| 1.4.2. ВЕБ-ПРИЛОЖЕНИЯ В ASP.....   | 20         |
| 1.4.3. ВЕБ-ПРИЛОЖЕНИЯ В PHP, PERL, PYTHON, RUBY.....                                     | 21         |
| 1.4.4. СРАВНЕНИЕ АРХИТЕКТУР .....  | 21         |
| <b>1.5. ПРОЦЕСС РАЗРАБОТКИ ВЕБ-ПРИЛОЖЕНИЙ .....</b>                                      | <b>22</b>  |
| <b>1.6. РЕКОМЕНДУЕМЫЕ ИСТОЧНИКИ ИНФОРМАЦИИ .....</b>                                     | <b>24</b>  |
| <b>ТЕМА 2. ИНФРАСТРУКТУРА В РАЗРАБОТКЕ НА PHP .....</b>                                  | <b>25</b>  |
| <b>2.1. ИСПОЛЬЗОВАНИЕ LINUX В РАЗРАБОТКЕ НА PHP .....</b>                                | <b>25</b>  |
| 2.1.1. Подготовка виртуальной машины.....  | 25         |
| 2.1.2. Установка UBUNTU SERVER.....  | 29         |
| 2.1.3. Настройка UBUNTU SERVER .....   | 38         |
| <b>2.2. УСТАНОВКА, НАСТРОЙКА И ИСПОЛЬЗОВАНИЕ LAMP И LEMP.....</b>                        | <b>43</b>  |
| 2.2.1. Упрощённый способ установки.....  | 43         |
| 2.2.2. Более сложный путь установки .....  | 48         |
| 2.2.3. Установка и настройка NGINX .....   | 50         |
| 2.2.4. Как теперь всем этим пользоваться .....   | 51         |
| <b>2.3. ИСПОЛЬЗОВАНИЕ WINDOWS В РАЗРАБОТКЕ НА PHP .....</b>                              | <b>52</b>  |
| <b>2.4. ОТЛАДКА И УСТРАНЕНИЕ ПРОБЛЕМ В PHP-ПРИЛОЖЕНИЯХ И КОНФИГУРАЦИИ ОКРУЖЕНИЯ.....</b> | <b>56</b>  |
| <b>2.5. ПРОСТЕЙШИЕ СПОСОБЫ РАЗВЁРТЫВАНИЯ PHP-ПРИЛОЖЕНИЙ.....</b>                         | <b>61</b>  |
| 2.5.1. РАЗВЁРТЫВАНИЕ ВЕБ-ПРИЛОЖЕНИЙ.....   | 61         |
| 2.5.2. Типичные вопросы и ответы по настройке .....                                      | 69         |
| <b>ТЕМА 3. ОСНОВЫ PHP .....</b>  | <b>75</b>  |
| <b>3.1. ОБЩИЕ СВЕДЕНИЯ О PHP .....</b>   | <b>75</b>  |
| <b>3.2. ЧТО НУЖНО ДЛЯ РАБОТЫ PHP .....</b>   | <b>76</b>  |
| <b>3.3. ОБЩИЙ СИНТАКСИС PHP .....</b>  | <b>82</b>  |
| <b>3.4. ПЕРЕМЕННЫЕ И ТИПЫ ДАННЫХ PHP.....</b>  | <b>83</b>  |
| <b>3.5. ОПРЕДЕЛЕНИЕ И ПРЕОБРАЗОВАНИЕ ТИПОВ ДАННЫХ .....</b>                              | <b>89</b>  |
| <b>3.6. ОСНОВНЫЕ ФУНКЦИИ PHP, С КОТОРЫХ НАДО НАЧАТЬ .....</b>                            | <b>95</b>  |
| <b>3.7. ОПЕРАТОРЫ И УПРАВЛЯЮЩИЕ КОНСТРУКЦИИ PHP .....</b>                                | <b>97</b>  |
| 3.7.1. ОПЕРАТОРЫ PHP .....   | 97         |
| 3.7.2. УПРАВЛЯЮЩИЕ КОНСТРУКЦИИ PHP .....   | 106        |
| <b>3.8. МАТЕМАТИЧЕСКИЕ ФУНКЦИИ PHP .....</b>   | <b>109</b> |
| <b>3.9. ФУНКЦИИ PHP , ОПРЕДЕЛЯЕМЫЕ ПОЛЬЗОВАТЕЛЕМ .....</b>                               | <b>112</b> |
| <b>3.10. РАБОТА С МАССИВАМИ В PHP .....</b>  | <b>119</b> |

|       |  |     |
|-------|--|-----|
| 3.11. | РАБОТА СО СТРОКАМИ В PHP .....                           | 133 |
| 3.12. | ФУНКЦИИ PHP ПО РАБОТЕ С ДАТОЙ И ВРЕМЕНЕМ .....           | 151 |
| 3.13. | ФУНКЦИИ PHP ПО РАБОТЕ С ФАЙЛОВОЙ СИСТЕМОЙ.....           | 153 |
| 3.14. | ЭЛЕМЕНТАРНЫЕ ФУНКЦИИ PHP ПО РАБОТЕ С БАЗАМИ ДАННЫХ ..... | 158 |
| 3.15. | РАБОТА С XML НА PHP .....                                | 161 |
| 3.16. | ОБРАБОТКА ОШИБОЧНЫХ СИТУАЦИЙ И ИСКЛЮЧЕНИЙ В PHP .....    | 171 |
| 3.17. | ООП В PHP .....  | 179 |
| 3.18. | ПАТТЕРНЫ ПРОЕКТИРОВАНИЯ, СХЕМА MVC .....                 | 188 |
| 3.19. | РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ В PHP.....                          | 202 |
| 3.20. | ОБРАБОТКА ФОРМ НА PHP .....                              | 212 |
| 3.21. | СТАНДАРТЫ PSR .....                                      | 215 |
| 3.22. | БИБЛИОТЕКА PEAR, ФОРМАТ PHAR .....                       | 216 |
| 3.23. | PHPDOC .....   | 221 |

## Предисловие

Что такое «рабочая тетрадь»? В отличие от книги в рабочей тетради можно и **НУЖНО** делать пометки и записи. Это ВАШ материал, в котором отражено ВСЁ, что будет показано на презентациях.

Здесь есть специальные места для записей (разлинованные участки), но вы также можете что-то обводить, подчёркивать и т.д.

Этот материал можно будет использовать на всех практических занятиях, а также для повторения изученного и просто как своего рода справочник или «то, что можно просто почитать».

Наш тренинг охватывает материал, для полноценного усвоения которого вам рекомендуется приложить максимум усилий в выполнении практических заданий, использовать все свои знания из области ИТ и активно пользоваться технической документацией (часть которой придётся искать самостоятельно, т.к. невозможно предсказать заранее, какие затруднения могут возникнуть).

Автор рабочей тетради неставил перед собой задачу скопировать официальную документацию, а потому каждая рассмотренная тема НЕ является «исчерпывающим руководством», после изучения которого вы сразу станете гуру. НО! Вы всегда можете обратиться к официальной документации от разработчиков. Не удивляйтесь, что некоторой информации не будет и там: разработчики считают многие вещи «самоочевидными». Что делать? Спрашивать тренера и пользоваться поиском в Интернет.

Помните: задача тренинга и данной рабочей тетради – обозначить ключевые моменты, отталкиваясь от которых вы сможете нарабатывать свой личный опыт значительно быстрее, чем при «полностью самостоятельном» исследовании рассмотренных тем, но никакая рабочая тетрадь и ни один тренер не заменят ваш личный опыт, все условия для приобретения которого мы создадим.

Итого:

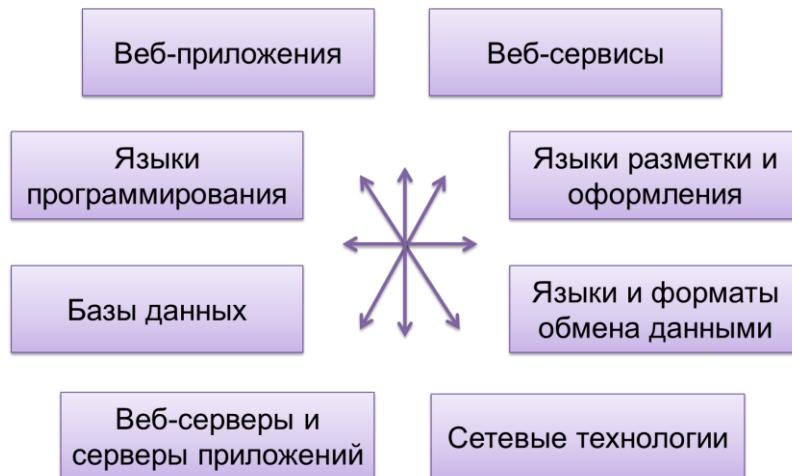
- Больше вопросов – любых! Обязательно ЗАПИСЫВАЙТЕ вопросы, которые возникли у вас дома при повторении материала. Тогда вы вспомните них на занятии и сможете задать.
- Пишите комментарии – везде! Это ВАША тетрадь. Используйте её!
- Обращайтесь к дополнительным источникам в Интернет и в документации, поставляемой с программами, которые мы рассмотрим.
- Используйте любую возможность, чтобы закрепить изученное на практике.

Успехов!

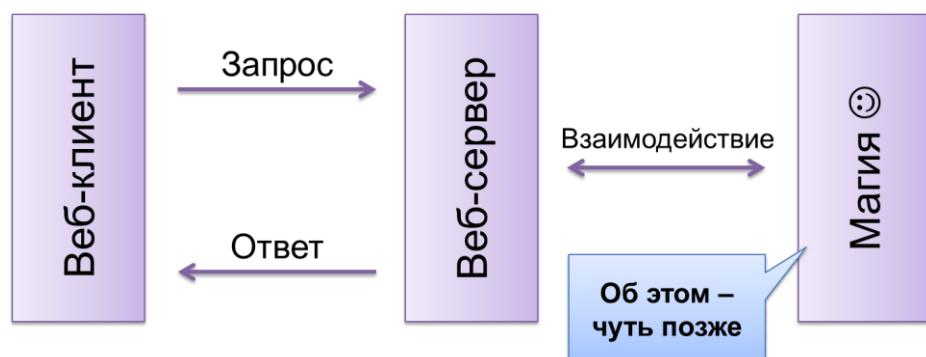
## Тема 1. Основы веб-технологий

### 1.1. Обобщённая схема веб-технологий

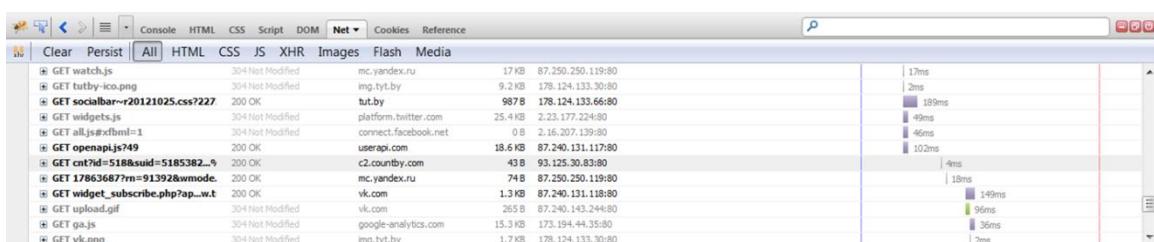
Веб-технологии работают на стыке множества самых разнообразных предметных областей, к которым относятся и целые технологические домены, и отдельные виды программных средств. В общем случае суть работы веб-технологий может быть выражена следующим рисунком:



**Веб-сервер (web server)** – специальное ПО, принимающее запросы (request) (как правило – HTTP) от веб-клиентов (web client), и генерирующее соответствующий ответ (response).



Для отображения одной страницы может выполняться много запросов: браузеру необходимо получить множество различных элементов – css-файлы, js-файлы, картинки и т.д. Увидеть выполняемые запросы очень просто – достаточно открыть панель «Net» в Firebug (бесплатном расширении браузера Firefox для разработчиков):



К основным функциям веб-сервера можно отнести:

| Функция   | Запишите здесь свои комментарии |
|---|---------------------------------|
| Обработка запросов и генерация ответов.         |                                 |
| Обеспечение опосредованного доступа к ФС.       |                                 |
| Обеспечение интерфейса к серверу приложений.    |                                 |
| Контроль прав доступа.                          |                                 |
| Шифрование трафика.                             |                                 |
| Балансировка нагрузки.                          |                                 |
| Маршрутизация запросов (по виртуальным хостам). |                                 |
| Обработка ошибочных ситуаций.                   |                                 |
| Протоколирование.                               |                                 |

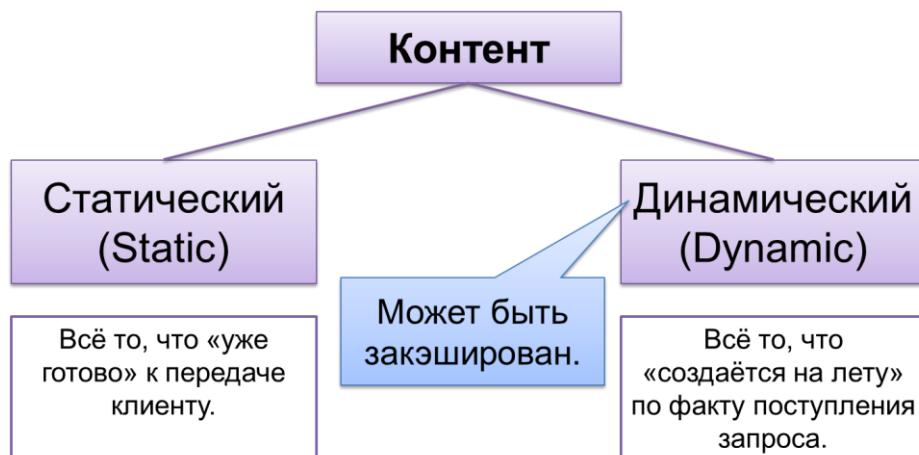
А ещё? Допишите ещё несколько функций со своими пояснениями.



[Почитать подробнее...](#)

[http://en.wikipedia.org/wiki/List\\_of\\_HTTP\\_status\\_codes](http://en.wikipedia.org/wiki/List_of_HTTP_status_codes)

Как уже было сказано ранее, веб-сервер обрабатывает запросы и генерирует ответ. В общем случае веб-сервер в ответ на запрос отдаёт «некий контент»:



Веб-сервер также может выполнять «преобразование URL».

**Преобразование URL (URL rewriting)** – способ представить динамические URL в максимально удобной для восприятия человеком форме.

[www.site.com/catalog/notebooks/hp/new/](http://www.site.com/catalog/notebooks/hp/new/)

VS

[www.site.com/index.php?page=catalog&category=notebooks&vendor=hp&mode=new](http://www.site.com/index.php?page=catalog&category=notebooks&vendor=hp&mode=new)



[Почитать подробнее...](#)

[http://httpd.apache.org/docs/current/mod/mod\\_rewrite.html](http://httpd.apache.org/docs/current/mod/mod_rewrite.html)

Один из способов реализации преобразования URL – использование модуля mod\_rewrite веб-сервера Apache.

Не самый корректный, но самый простой пример. В файле .htaccess прописывается:

```
RewriteEngine On
RewriteBase /
RewriteRule .* index.php?url=$0 [QSA,L]
```

Теперь запрос к любому объекту будет переадресован на index.php.

**Сервер приложений (application server)** – специальное ПО, выполняющее для взаимодействующих с ним приложений ряд специфических функций, таких как управление производительностью, безопасностью, взаимодействие с операционной системой и т.п.



Частным случаем серверов приложений можно считать т.н. «среды исполнения» (.NET Framework, Java Runtime Environment, PHP и т.п.)

К основным функциям веб-сервера можно отнести:

| Функция                                      | Запишите здесь свои комментарии |
|--|---------------------------------|
| Изолирование приложения от ОС и «железа».    |                                 |
| Предоставление API для типичных действий.    |                                 |
| Управление ресурсами.                        |                                 |
| Протоколирование.                            |                                 |
| Оптимизация выполнения операций.             |                                 |
| Балансировка нагрузки (в некоторых случаях). |                                 |
| Обработка ошибочных ситуаций.                |                                 |

А ещё? Допишите ещё несколько функций со своими пояснениями.

---

---

---

---

**Сетевые технологии (network technologies)** – совокупность технологий, обеспечивающих функционирование компьютерных сетей и аппаратных и программных средств, использующих в своей работе компьютерные сети.

Для полноценного понимания сетевых технологий, задействованных в работе веб-приложений, рекомендуется ознакомиться с семиуровневой моделью ISO/OSI (особое внимание стоит уделить уровням 6-7).



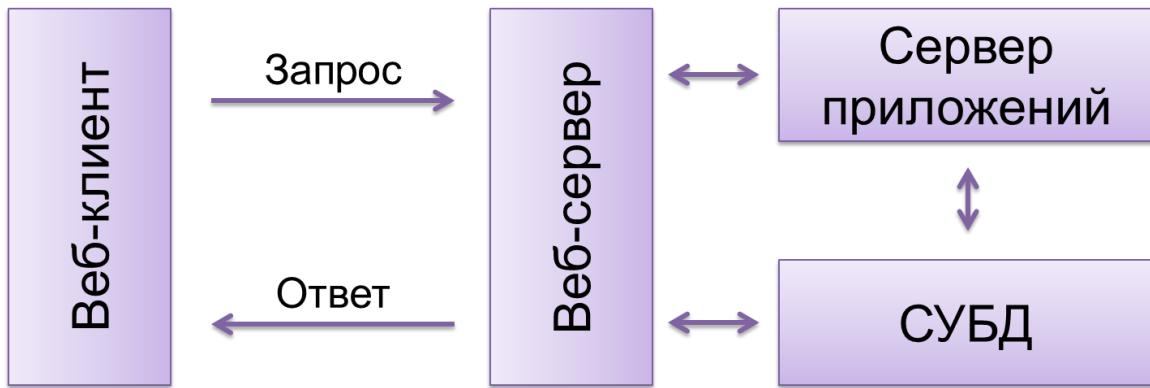
**Почитать подробнее...**

[http://en.wikipedia.org/wiki/OSI\\_model](http://en.wikipedia.org/wiki/OSI_model)

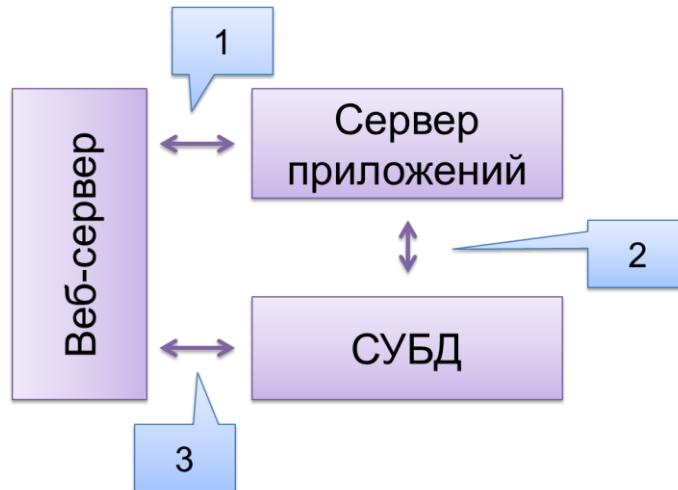
Если вкратце изложить суть модели ISO/OSI, получится:

| Тип данных | Уровень          | Функции  |
|------------|------------------|--|
| Данные     | 7. Прикладной    | Доступ к сетевым службам                                 |
|            | 6. Представления | Представление и кодирование данных                       |
|            | 5. Сеансовый     | Управление сеансом связи                                 |
| Сегменты   | 4. Транспортный  | Прямая связь между конечными пунктами и надежность       |
| Пакеты     | 3. Сетевой       | Определение маршрута и логическая адресация              |
| Кадры      | 2. Канальный     | Физическая адресация                                     |
| Биты       | 1. Физический    | Работа со средой передачи, сигналами и двоичными данными |

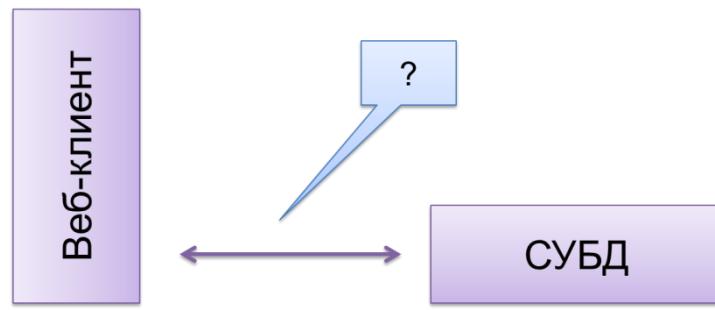
**Базы данных (databases)** под управлением СУБД (DBMS, database management systems, системы управления базами данных) являются универсальным хранилищем информации для веб-приложений.



В каких случаях происходят эти варианты взаимодействий (запишите ответ ниже)?



А бывают ли ситуации, когда с СУБД взаимодействует веб-клиент?



Что хранится и НЕ хранится в БД:



И ещё раз (для лучшего запоминания): «что где хранится» (расширенная версия).

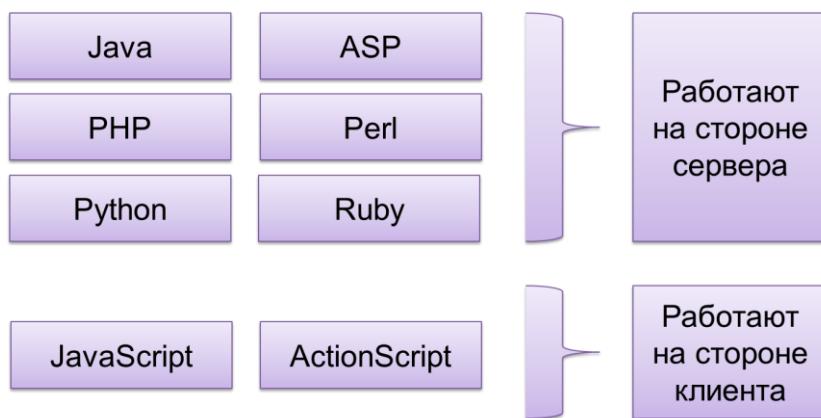
| Хранится в БД                | Хранится в файловой системе | Генерируется динамически  |
|------------------------------|-----------------------------|---------------------------|
| Структура приложения (сайта) | Скрипты                     | Готовые страницы          |
| Текстовое наполнение страниц | Мультимедийная информация   | Текстовая информация      |
| Некоторые настройки          | Некоторые настройки         | Мультимедийная информация |
| Кэш готовых страниц          | Кэш готовых страниц         |                           |

Языки и форматы обмена данными (web-languages and data exchange formats) являются неотъемлемой частью функционирования веб-технологии. К основным языкам и форматам обмена данными можно отнести:

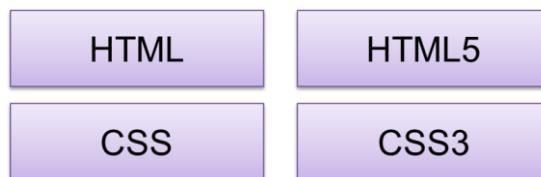


Настоятельно рекомендуется здесь и далее самостоятельно поискать расшифровки этих и подобных аббревиатур и прочитать, «что же это такое».

К основным языкам программирования (programming languages), используемым в контексте веб-технологий, можно отнести:



К основным языкам разметки и оформления (**layout markup and decoration languages**), используемым в контексте веб-технологий, можно отнести:



**HTML** (Hypertext Markup Language, язык гипертекстовой разметки) – специальный язык РАЗМЕТКИ, используемый для описания структуры документа.

**XHTML** (Extensible Hypertext Markup Language, расширяемый язык гипертекстовой разметки) – более строгий «вариант» HTML, базирующийся на спецификациях XML.

**HTML5** – «улучшенная» версия HTML, добавляющая широкий спектр новых возможностей. Полный список нововведений в HTML5 можно увидеть на сайте W3C.



[Почитать подробнее...](#)

<http://www.w3.org/TR/html5-diff/>

**CSS** (Cascading Style Sheets, каскадные таблицы стилей) – язык описания внешнего вида документа (написанного с использованием HTML).

**CSS3 (CSS4)** – новые версии CSS, следующие той же логике развития, что и HTML5 по отношению к «классическому» HTML (XHTML).

**Веб-приложение (web application)** – приложение, использующее для своей работы Интернет или интранет (локальную сеть). Как правило, работает по архитектуре: «веб-клиент + веб-сервер + сервер приложений + СУБД».

Часто веб-приложения строятся так, что для их работы необходим браузер, но это – не обязательное условие.

Запишите несколько примеров веб-приложений разных видов:

-----  
-----  
-----  
-----

**Веб-сервис (web service)** – ПО, предоставляющее возможность обмена данными по сети между приложениями, устройствами и другими веб-сервисами.

Запишите несколько примеров веб-сервисов:

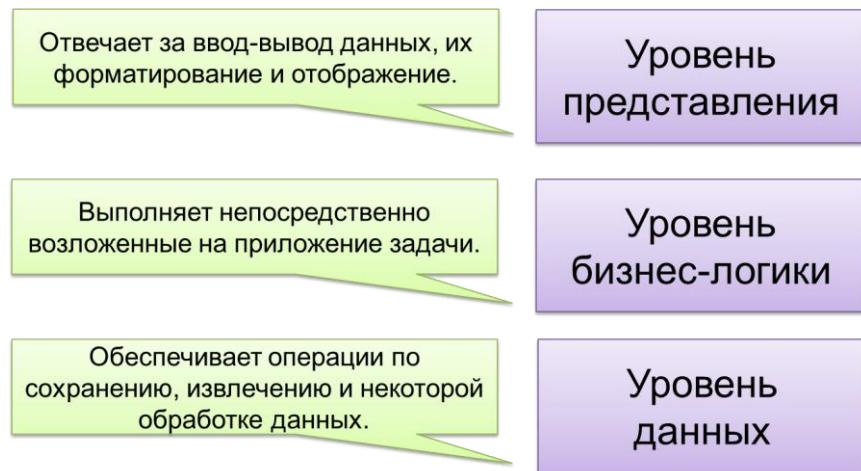
Остались вопросы? Запишите их! И тут же – ответы ☺.

## 1.2. Многоуровневая архитектура приложений

**Многоуровневая архитектура (multi-tier architecture)** – технология построения приложений, при которой для выполнения отдельных классов задач выделяются т.н. уровни (tiers), взаимодействующие друг с другом посредством определённых интерфейсов.



Задачи делятся между уровнями следующим образом:



Реализация уровней выглядит так:



Запишите несколько преимуществ и недостатком многоуровневой архитектуры:

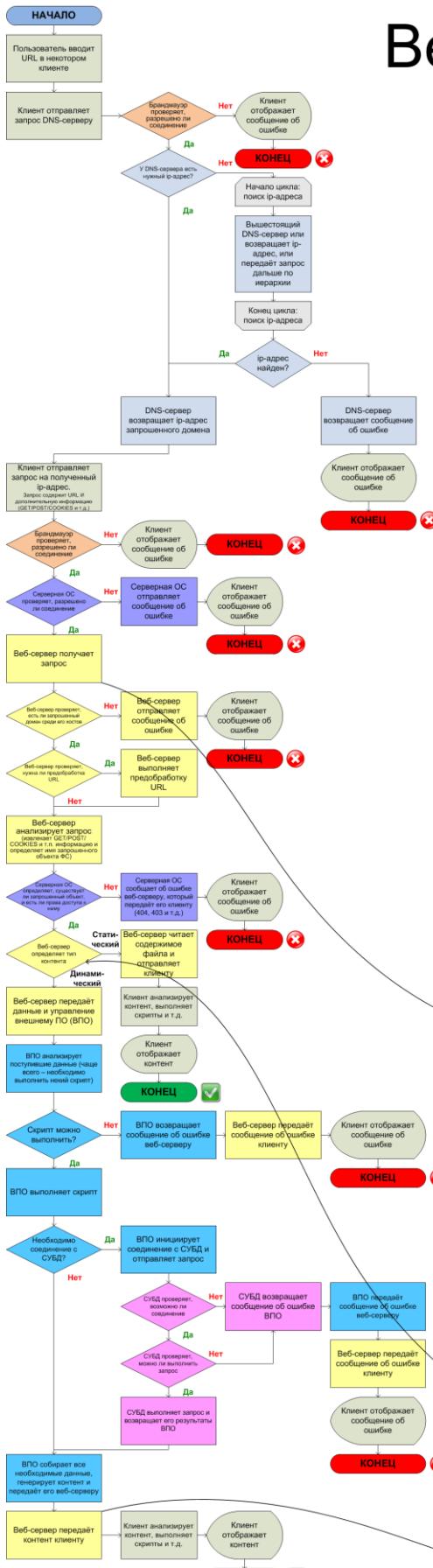
### 1.3. Принципы работы веб-приложений

Сложность понимания принципов работы веб-приложений заключается в том, что в работе задействовано очень много компонентов, которые выполняют очень много действий.

Описать их словами «в совершенно общем виде» едва ли возможно, т.к. простое описание сведётся к многоуровневой архитектуре, а более-менее адекватное будет слишком длинным и запутанным.

Для первичного понимания сути данного вопроса рекомендуется ознакомиться со следующими рисунками (увеличенная версия доступна в раздаточном материале).

В процессе рассмотрения рисунков рекомендуется делать пометки, записывать вопросы и ответы на них.



# Веб-технологии

## Веб-клиенты

**Браузеры**  
Firefox, Chrome, Safari, MSIE, Opera

«Загрузчики» сайтов  
Teleport Pro, Offline Explorer, WebZip  
  


Менеджеры закачек  
FlashGet, GetRight, DownloadMaster

Средства работы с RSS  
Alertbear, NewsGator, RSS bandit  
  


**Брандмауэры**  
Брандмауэр – это специальное программное средство, контролирующее доступ к веб-ресурсам извне. Основная задача – блокировка несанкционированных соединений и передачи несанкционированных данных.

**DNS-серверы**  
DNS-сервер – это специальное ПО, обеспечивающее трансляцию доменных имён в IP-адреса. DNS-серверы обработают исходящую систему, на которую находится хостинг-DNS, определяя полные наименования доменов в соответствии с доменными именами и IP-адресами.

**Серверные ОС**  
Серверные ОС предназначены для выполнения сетевого сервисного ПО. Они удовлетворяют повышенным требованиям к производительности, безопасности и надежности.

**Веб-серверы**  
Web-server is:  
1) A computer program that is accepting HTTP requests from clients and sending them HTTP responses back. These responses are web pages such as HTML documents and linked objects such as images, etc.  
2) A computer (hardware) that runs a computer program as described above.

**Серверы приложений**

Сервер приложений – это:  
1) Начальный сервис, предоставляемый сервером.  
2) Компьютер, на котором выполняется сервис (см. пункт 1).  
3) Программное обеспечение, которое предоставляет услуги (точнее, порт) для пункта 1), например, Java Application Server или Oracle Application Server.

Язык программирования и технологии создания динамического контента – это совокупность языков и технологий, позволяющих обрабатывать поступающий запрос и сгенерировать соответствующий контент для передачи клиенту.

**СУБД**

СУБД (система управления базами данных) – это программа, которая управляет данными. СУБД могут работать с различными видами БД (реляционными, объектными, и т.д.). СУБД позволяют пользователям получать доступ к БД, обрабатывать структурированные данные единодушно.

Сервер баз данных – это:  
1) Системы СУБД.  
2) Компьютер, на котором работает СУБД.

**Браузеры**  
Firefox, Chrome, Safari, MSIE, Opera

«Загрузчики» сайтов  
Teleport Pro, Offline Explorer, WebZip  
  


Менеджеры закачек

FlashGet, GetRight, DownloadMaster

Брандмауэры

MS Firewall, Reasoft, IPSec, Outpost

DNS-серверы

UltraDNS, OpenDNS, BIND

Серверные ОС

FreeBSD, Linux, Sun Solaris, HP UX, MS Windows Server

Веб-серверы

Apache, IIS, Lighttpd, NGINX

Серверы приложений

Zope, WebSphere, Oracle AS, Tomcat

Языки программирования и технологии создания динамического контента

Java, ASP C#, PHP, Python, Perl, Ruby

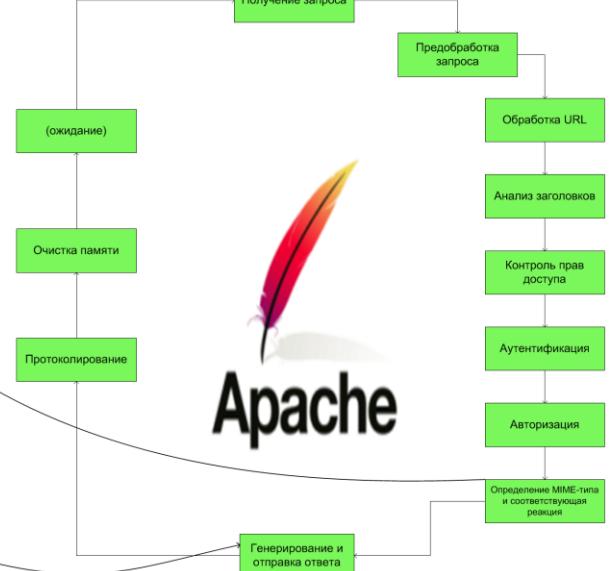
      

СУБД

Oracle, PostgreSQL, MS-SQL, MySQL

## Обработка запроса веб-сервером Apache

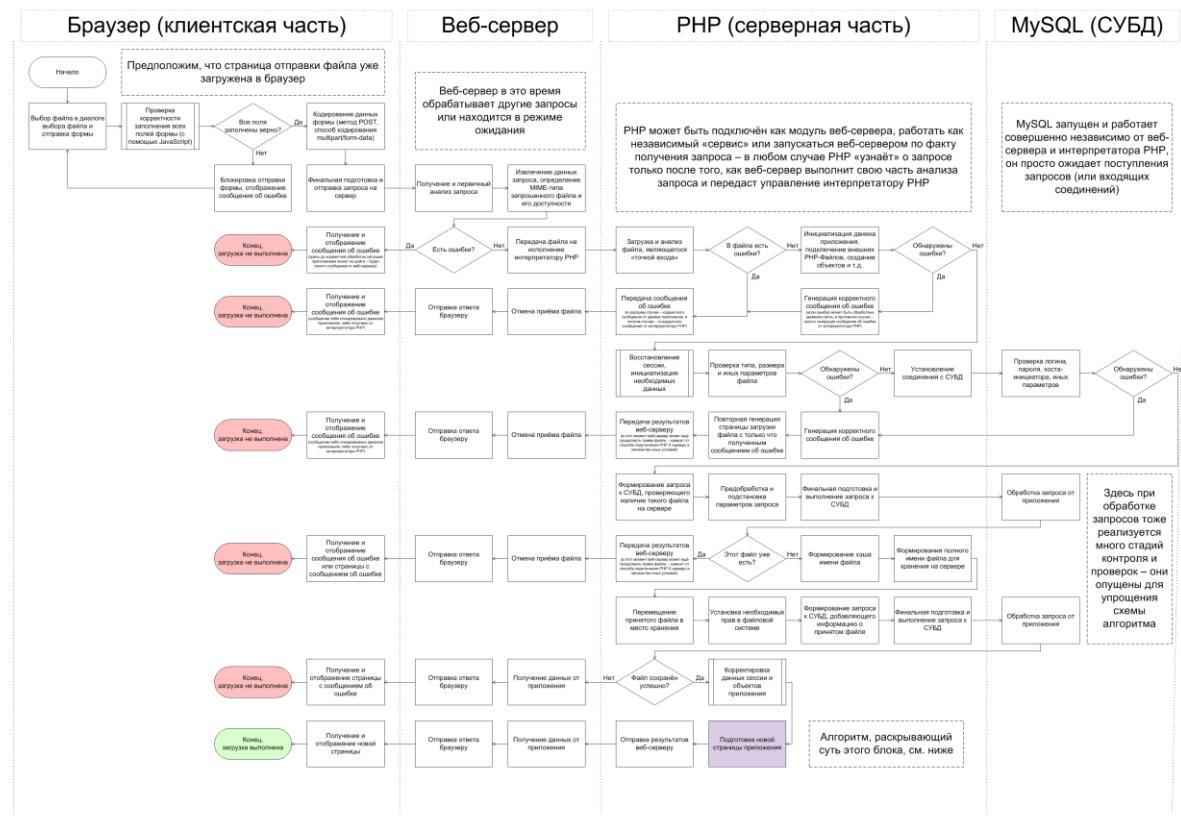


# Apache

## Работа веб-приложения на примере сохранения файла на сервере

Важно! Это – «обобщённый алгоритм», призванный создать понимание сути происходящего. Это НЕ строгая техническая документация!

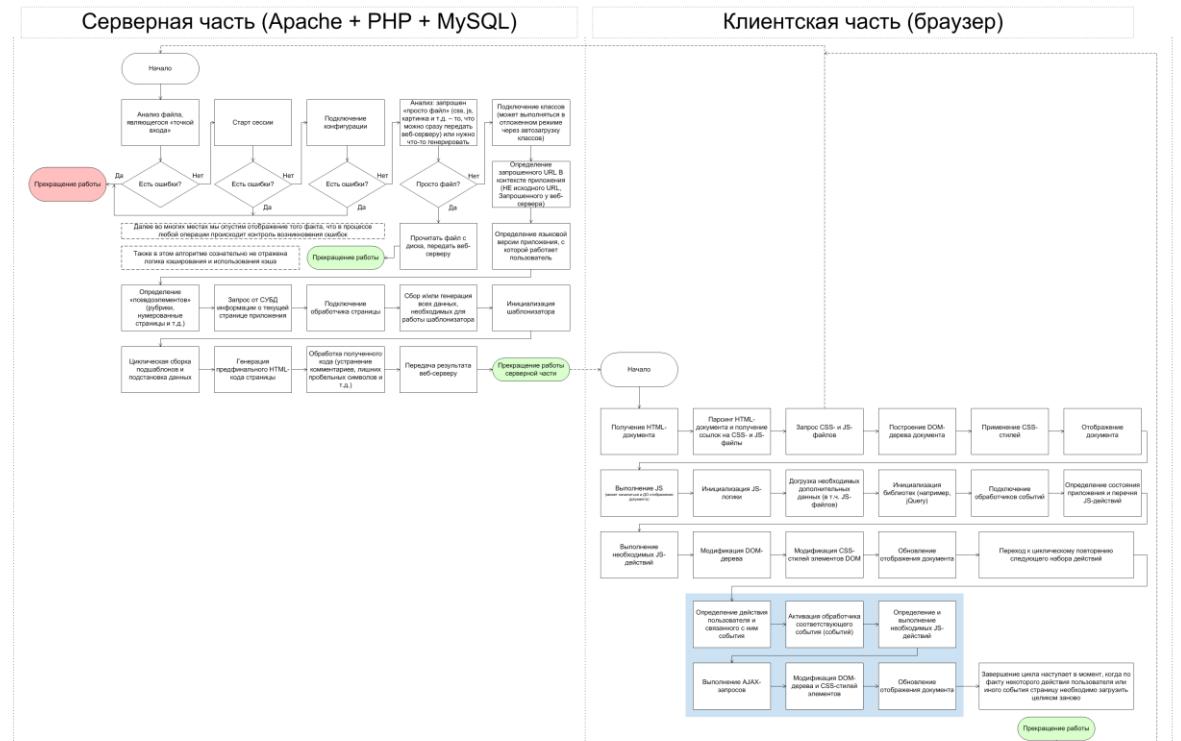
Время (общая последовательность развития событий)



## Генерация страницы приложения (логика сервера и клиента)

Важно! Это – «обобщённый алгоритм», призванный создать понимание сути происходящего. Это НЕ строгая техническая документация!

Время (общая последовательность развития событий)

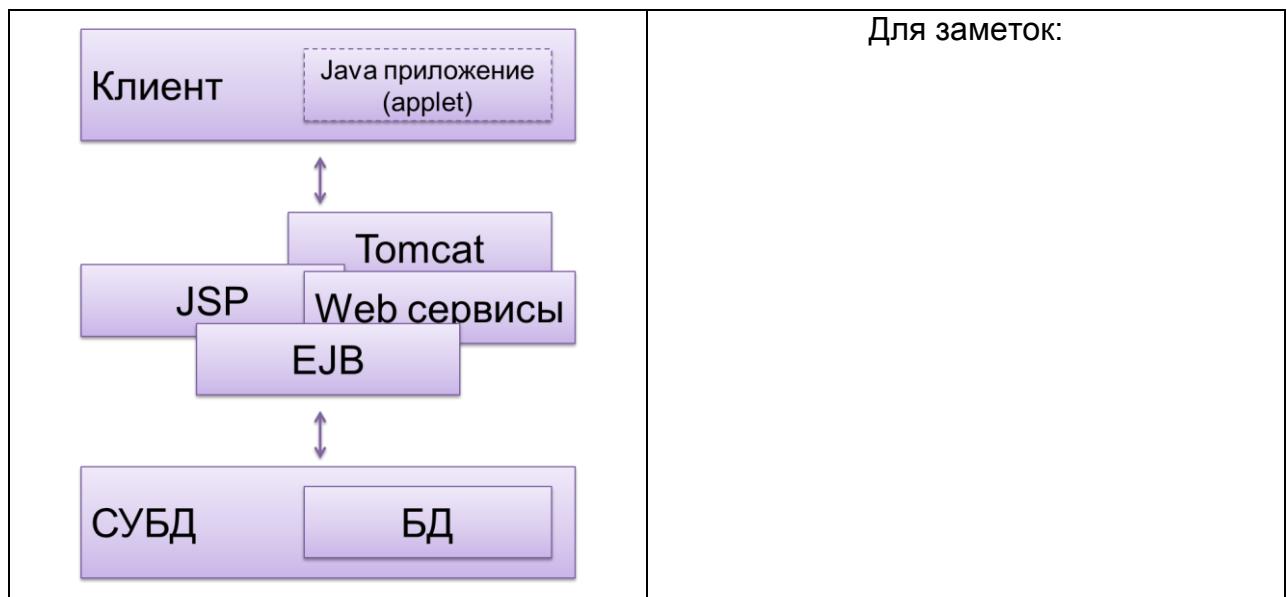


## 1.4. Архитектура веб-приложений

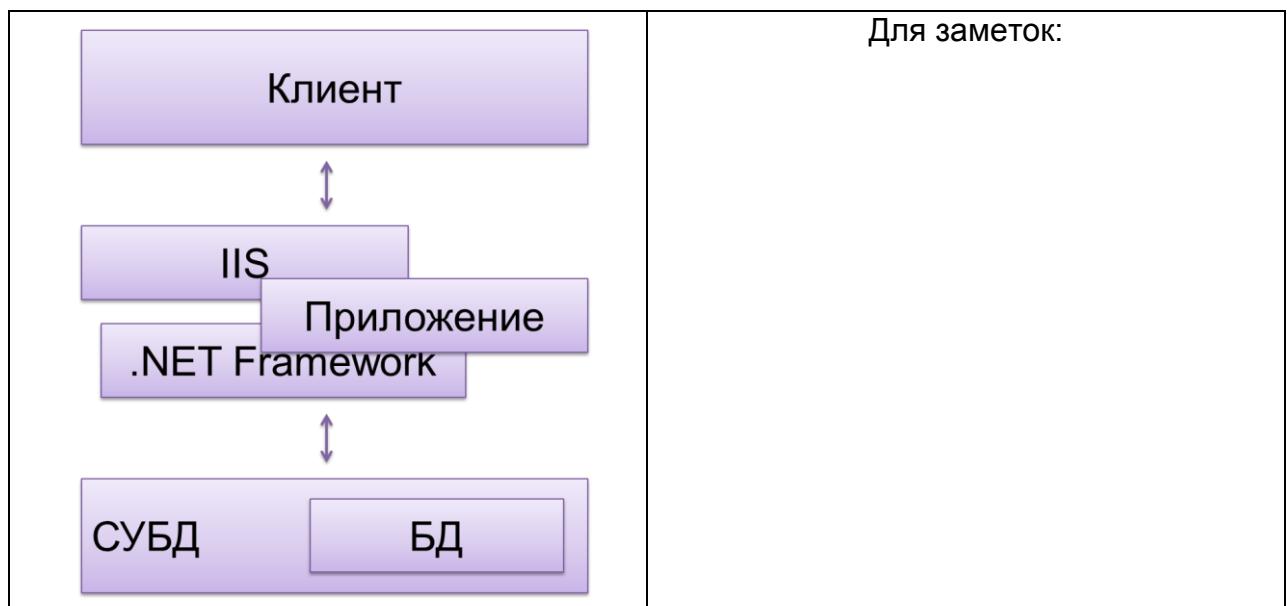
Только что мы вкратце рассмотрели основные понятия многоуровневой архитектуры и основные принципы работы веб-приложений. Данный раздел является небольшим уточнением, позволяющим увидеть реализацию рассмотренного с использованием различных технологий.

Важное примечание: каждый следующий подпункт, фактически, является названием целого курса (или даже нескольких). Мы не можем (и не ставим целью) рассмотреть здесь эти вопросы подробно.

### 1.4.1. Веб-приложения в Java



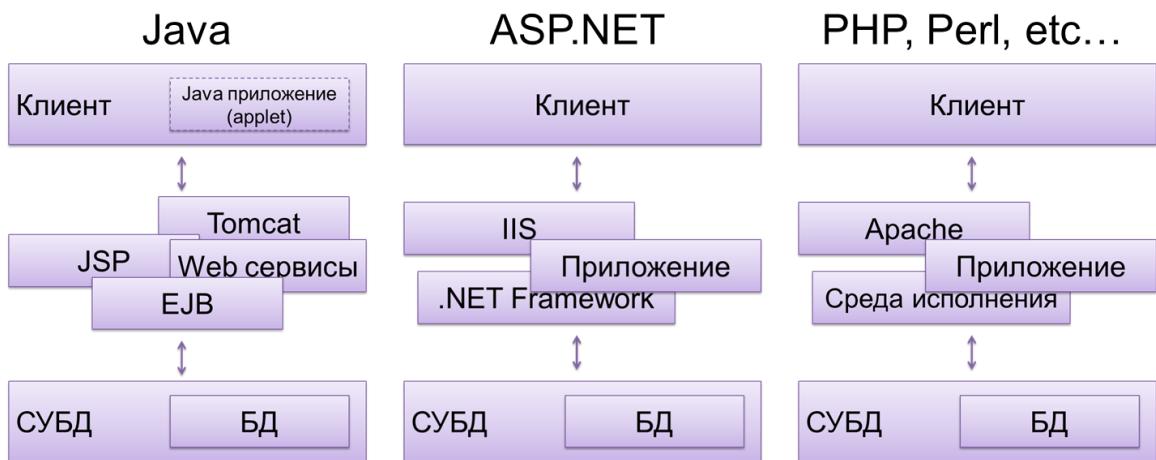
### 1.4.2. Веб-приложения в ASP



### 1.4.3. Веб-приложения в PHP, Perl, Python, Ruby



### 1.4.4. Сравнение архитектур

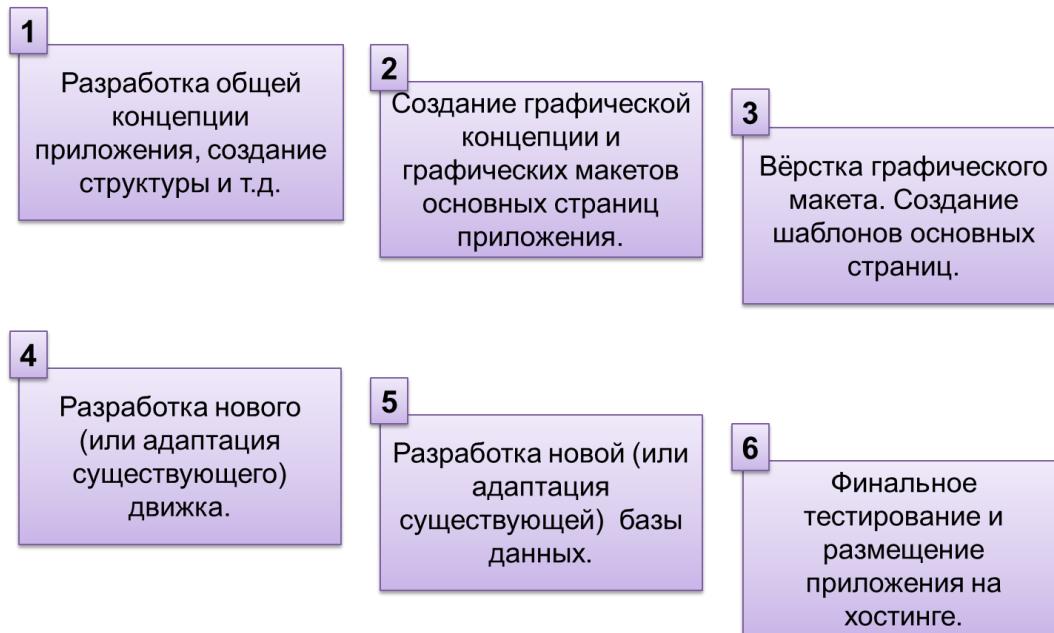


В чём эти архитектуры схожи? В чём их отличия?

| Схожесть | Отличие |
|----------|---------|
|          |         |
|          |         |
|          |         |
|          |         |

## 1.5. Процесс разработки веб-приложений

Если рассматривать процесс разработки веб-приложений линейно, получается следующий алгоритм.



### 1. Разработка общей концепции приложения, его структуры, функциональных особенностей и т.п.

На первой стадии всей проектной команде следует максимально сплочённо поработать над тем, чтобы выработать единую стройную концепцию веб-приложения, чтобы в будущем не пришлось «пришивать верблюду крылья».

Очень хорошо, если в команде есть опытный тестировщик (это, к слову, всегда хорошо), знакомый с дисциплинами «Выработка, анализ и тестирование требований» и «Прототипирование».

На данной стадии очень желательно уделить внимание документации или, хотя бы, составить подробное техническое задание (ТЗ), которое чётко согласовать с заказчиком. Это позволит в будущем избежать ситуации, когда за те же деньги придётся делать в три раза больше работы.

### 2. Создание графического макета основных страниц

В крупных проектах «макетироваться» могут все страницы, причём довольно подробно. В небольших проектах обычно ограничиваются макетами главной страницы, «второстепенной страницы» (единой по внешнему виду для всех подразделов сайта), а также страниц, принципиально отличающихся от главной и «второстепенной» (карты сайта, страницы поиска, страниц каталога, форм регистрации и заказа товара, корзины и т.п.)

На данной стадии работы над проектом основную задачу выполняют художник и дизайнер (если они есть в виде «отдельных людей») или же те, кто выполняет их функции.

### **3. Вёрстка графического макета. Создание т.н. «базовых шаблонов страниц»**

На данной стадии работы над проектом основную задачу выполняет верстальщик. На основе «картинок из фотошопа» (полученных на стадии 2) он создаёт статические шаблоны страниц (перечень страниц – см. в предыдущем пункте).

На выходе его работы получаются страницы сайта в таком виде, в каком они могли бы быть сгенерированы движком при некоторых условиях.

Желательно заполнять такие шаблоны реальной информацией, чтобы видеть, где что-то может «поехать», что-то не отобразиться и т.д.

ОЧЕНЬ ЖЕЛАТЕЛЬНО, чтобы верстальщик (равно как художник и дизайнер) имел представление о том, «что можно запрограммировать, а что – нет», т.к. в противном случае при дальнейшей работе с проектом веб-программистам придется долго ломать голову, как в коде реализовать то, что «на картинке» нарисовалось так просто.

### **4. Разработка нового (или адаптация готового) движка под нужды конкретного проекта**

На данной стадии работы над проектом основную задачу выполняет команда веб-программистов.

Они режут шаблоны, полученные на стадии 3, на «куски» и организуют логику работы движка таким образом, чтобы из этих «кусочных шаблонов» в итоге получился красивый и функциональный сайт.

Подробнее о шаблонизации и разработке движка мы поговорим в соответствующих темах курса.

### **5. Разработка новой (или адаптация существующей) базы данных под нужды конкретного проекта**

На данной стадии работы над проектом основную задачу выполняет специалист по базам данных (если он есть в проектной команде) или же – всё те же веб-программисты.

Несмотря на то, что наш курс не посвящён вопросам разработки баз данных, следует отметить важность этого момента. Хорошая, надёжная, грамотно разработанная база данных – одна из важных составляющих хорошего веб-ориентированного приложения.

### **6. Финальное тестирование и «деплоймент» приложения на хостинговую платформу**

О тестировании следует сказать несколько слов особо. Оно должно быть на КАЖДОЙ стадии работы с проектом. На КАЖДОЙ и без единого исключения. Чем раньше будет обнаружена ошибка, тем быстрее, проще и дешевле будет её исправление.

Однако в конце проекта тестированию следует уделить особое внимание. Следует проверить работу приложения как во всех «стандартных», так и в хотя бы наиболее вероятных «нестандартных» ситуациях.

Если в проектной команде есть тестировщик, он может заняться нагрузочным и стрессовым тестированием, тестированием производительности, безопасности, юзабилити, тестированием базы данных, кросбраузерным и кроссплатформенным тестированием и т.д. и т.п.

Когда всё готово, сайт закачивается на хостинговую платформу, где снова тестируется.

## **1.6. Рекомендуемые источники информации**

К основным источникам информации по нашему курсу стоит отнести:

- <http://google.com> – и это не шутка, т.к. никакое учебное пособие не сможет дать вам ответы на ВСЕ ваши вопросы.
  - <http://w3schools.com> – прекрасный учебный портал, где просто и доступно рассказано о множестве веб-ориентированных тем.
  - <http://habrahabr.ru> – ресурс, который должен читать по утрам любой ИТ-шник ☺.

А где же книги?! Здесь: <http://oz.by>.

## Тема 2. Инфраструктура в разработке на PHP

### 2.1. Использование Linux в разработке на PHP

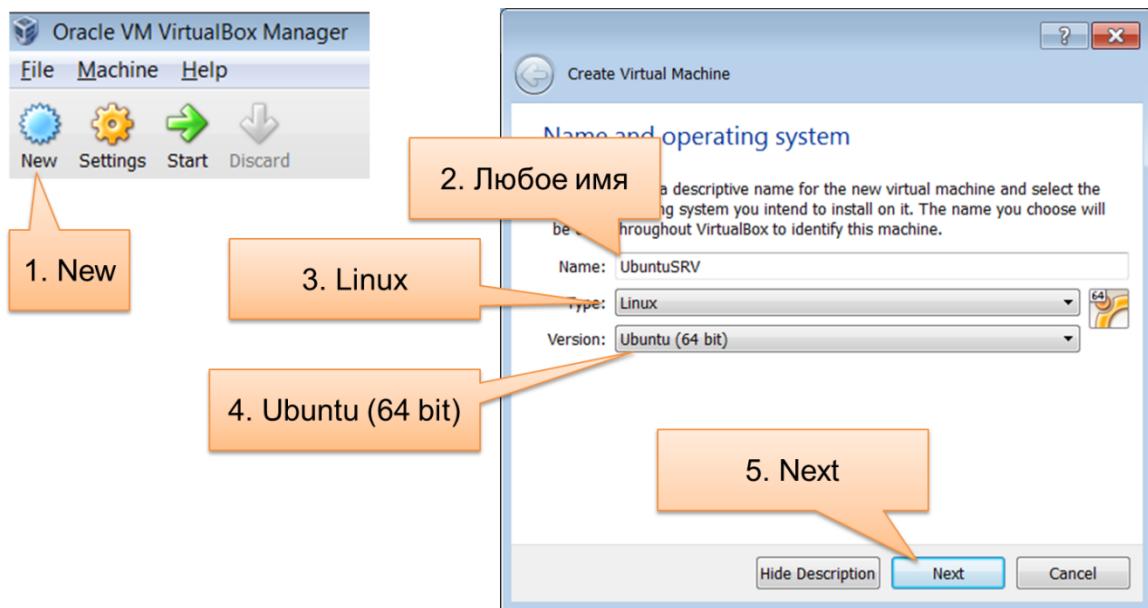
#### 2.1.1. Подготовка виртуальной машины

Мы будем использовать Ubuntu Server x64, которую установим и настроим под управлением Oracle VM VirtualBox.

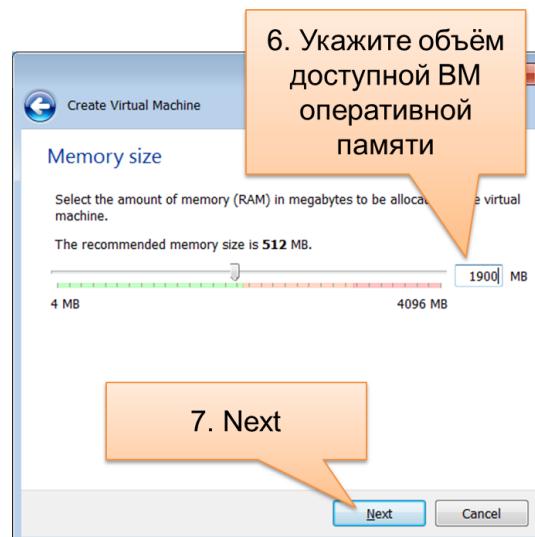
Потому, прежде, чем приступить к работе, скачайте и установите последнюю версию VirtualBox и его расширений: <https://www.virtualbox.org/wiki/Downloads>

Также скачайте последнюю версию Ubuntu Server (мы будем использовать 64-битную версию): <http://www.ubuntu.com/download/server>

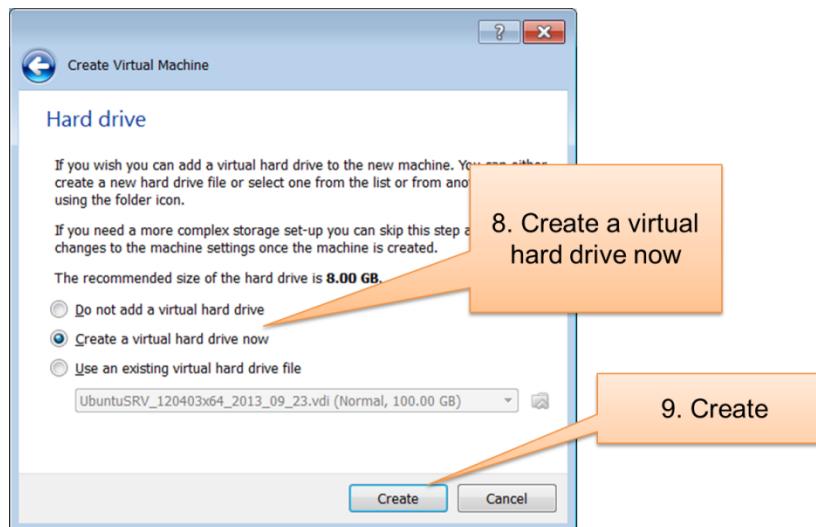
Запустите VirtualBox Manager и начните создание новой виртуальной машины:



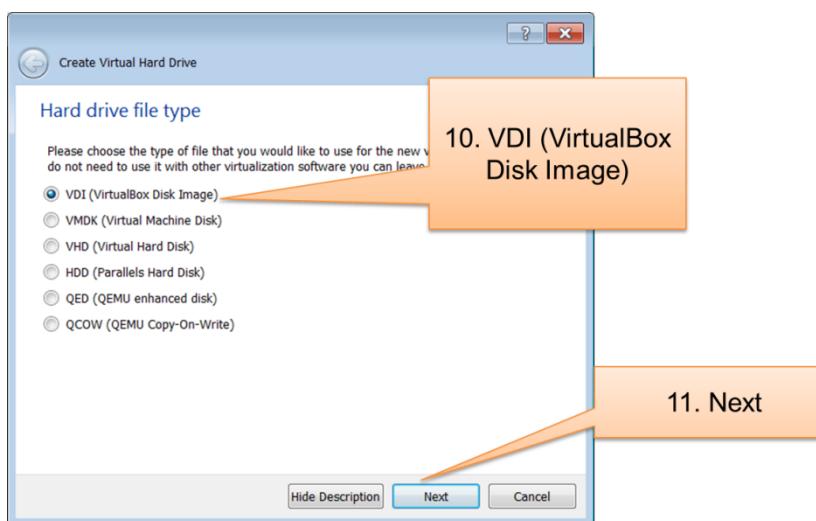
Объём оперативной памяти – от 512 Mb и более. Рекомендуется – примерно 40% от имеющейся физической оперативной памяти:



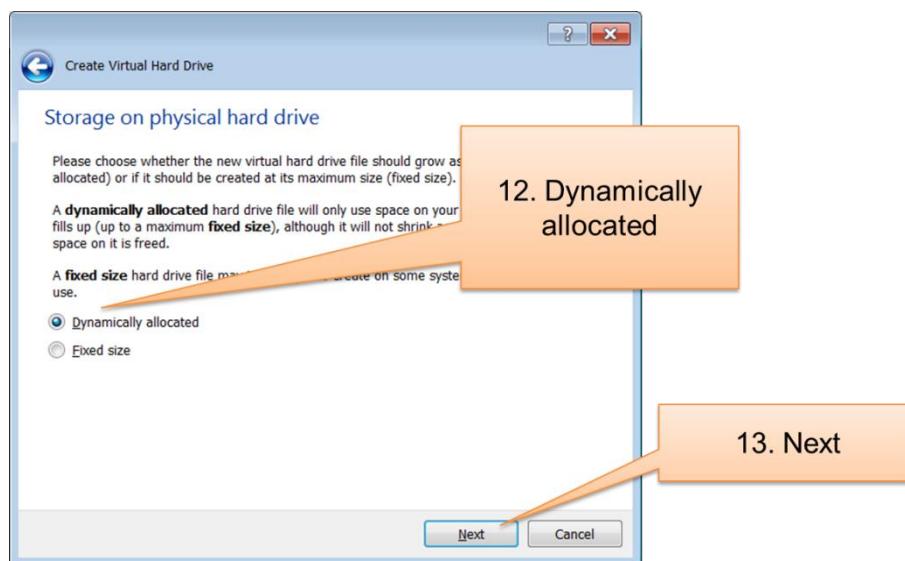
## Создайте новый виртуальный диск:



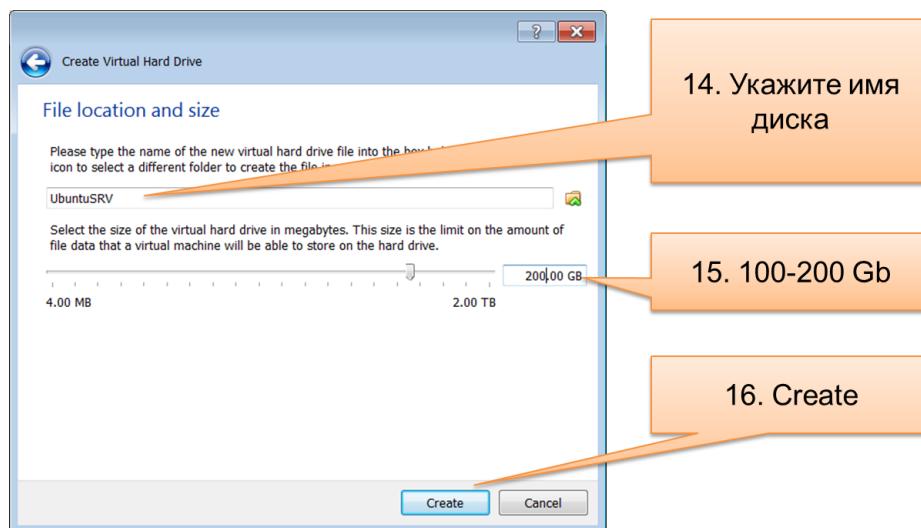
Формат диска можно оставить по умолчанию:



Тип диска – динамически расширяемый:



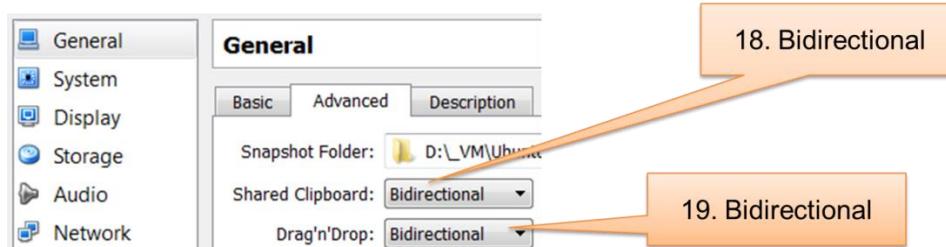
Укажите имя образа и максимальный размер (рекомендуется – 100-200 Gb, чтобы был запас):



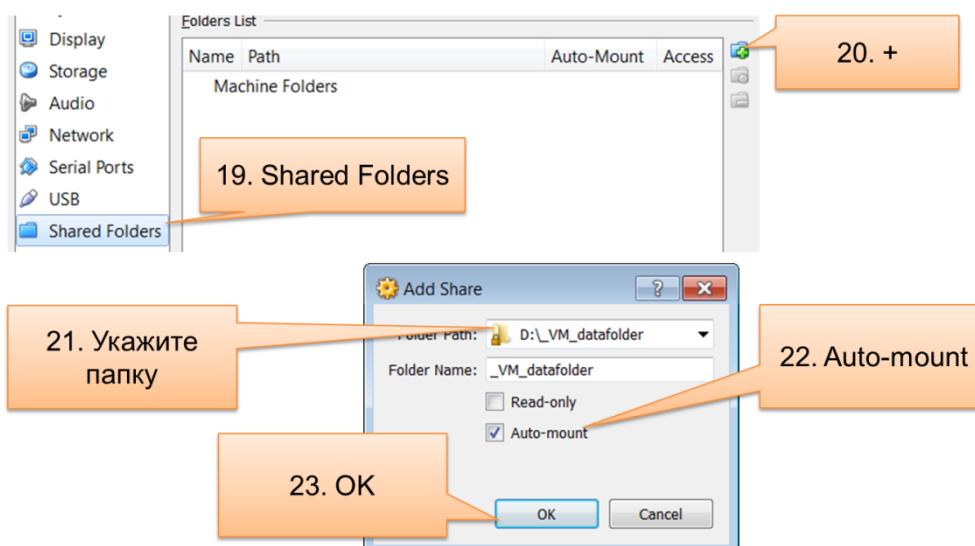
Откройте настройки виртуальной машины:



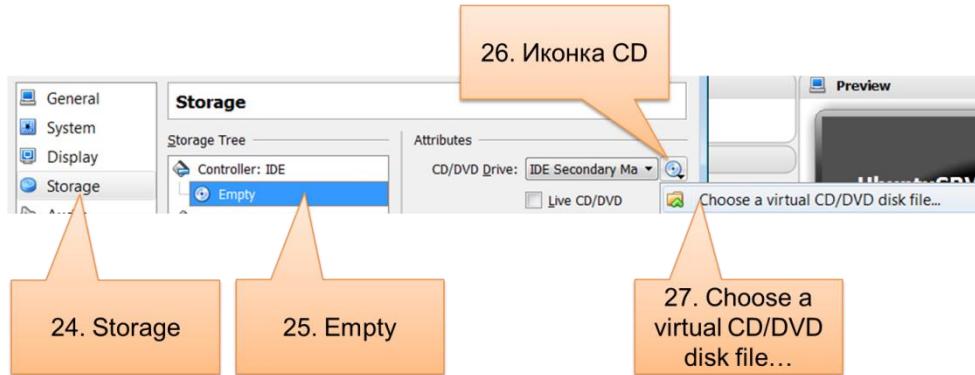
Включите двунаправленный обмен файлами и содержимым буфера:



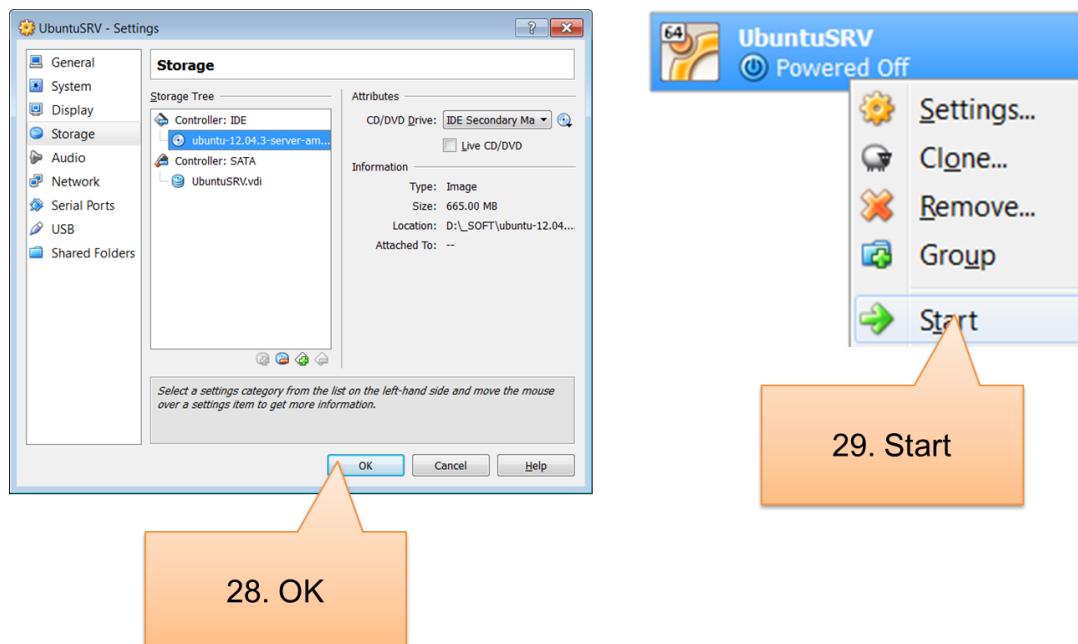
Для упрощения обмена файлами добавьте общую папку (скорее всего, работать не будет 😊, но мы это «поборем» другим способом):



Подключите к вашей ВМ образ диска с дистрибутивом Ubuntu Server:

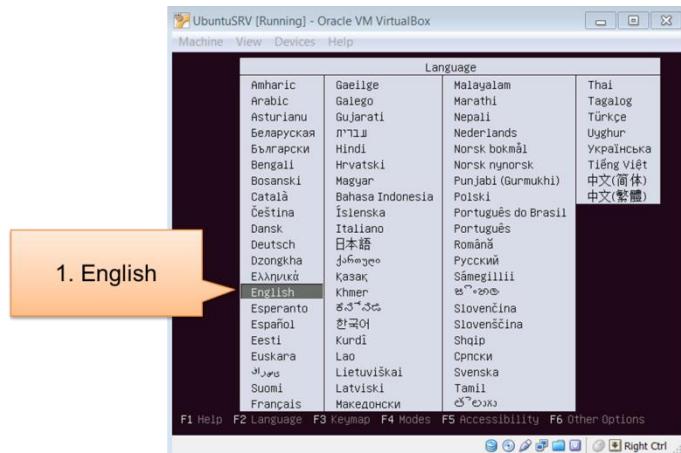


Всё, осталось закрыть окно настроек и запустить ВМ:

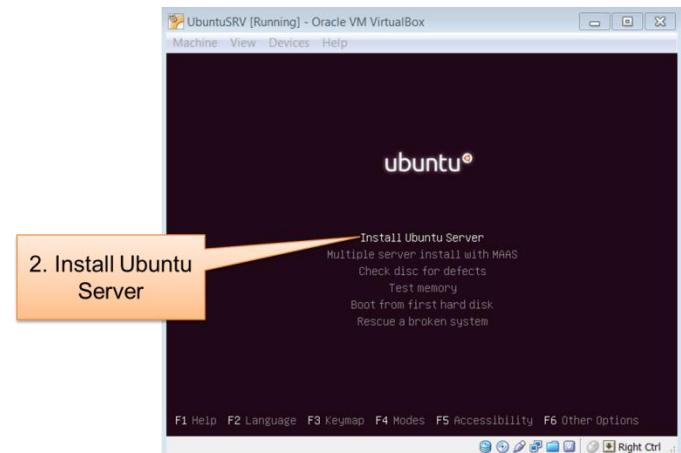


## 2.1.2. Установка Ubuntu Server

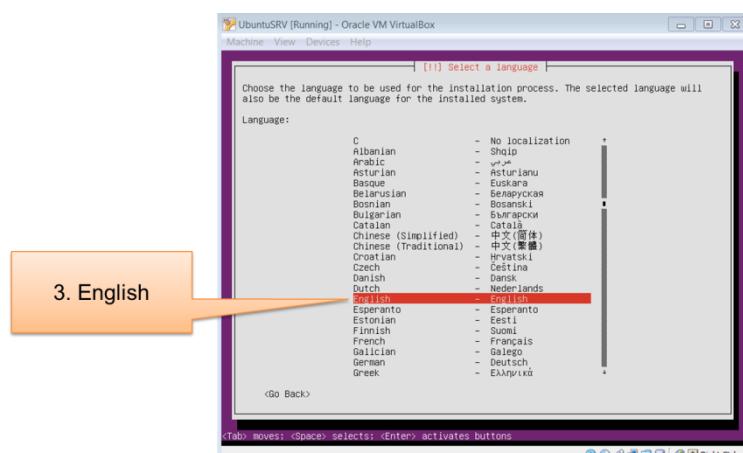
Выберите английский язык:



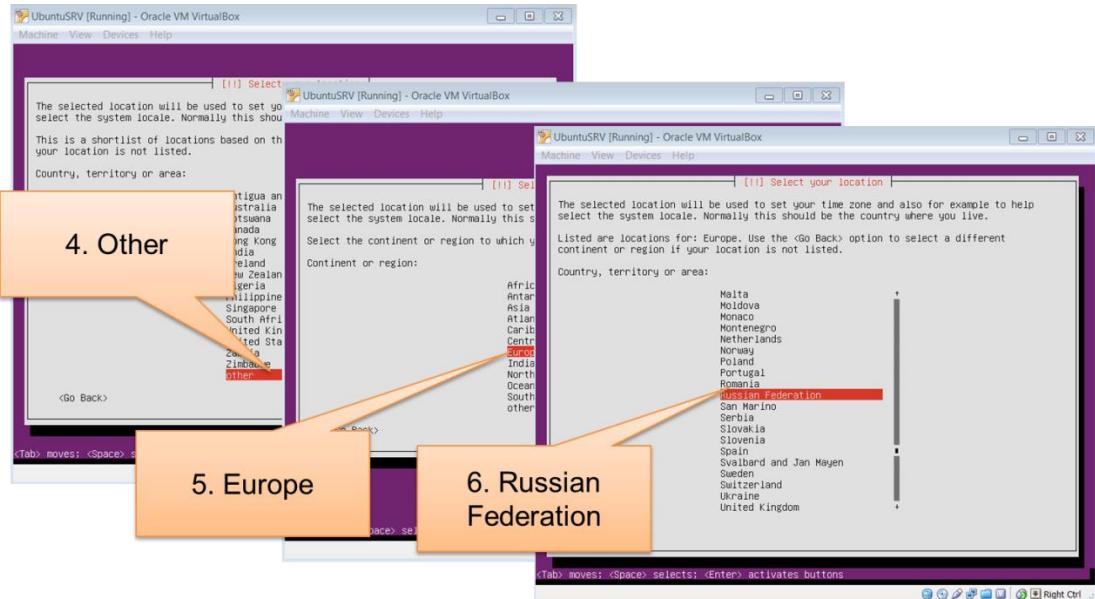
Выберите установку ОС:



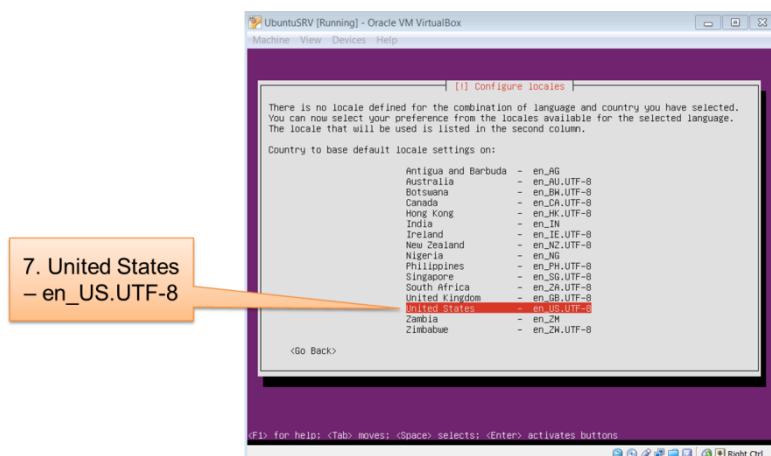
Снова выберите английский язык:



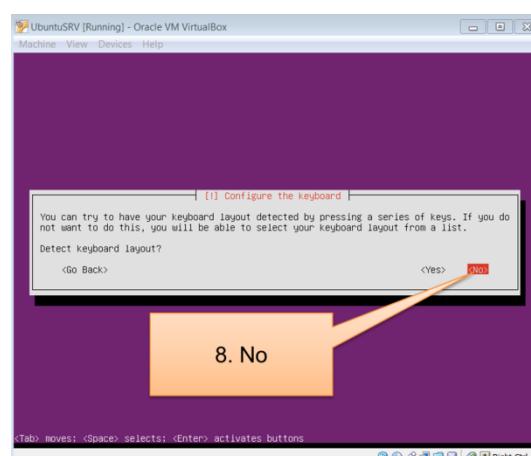
В качестве местонахождения укажите Россию:



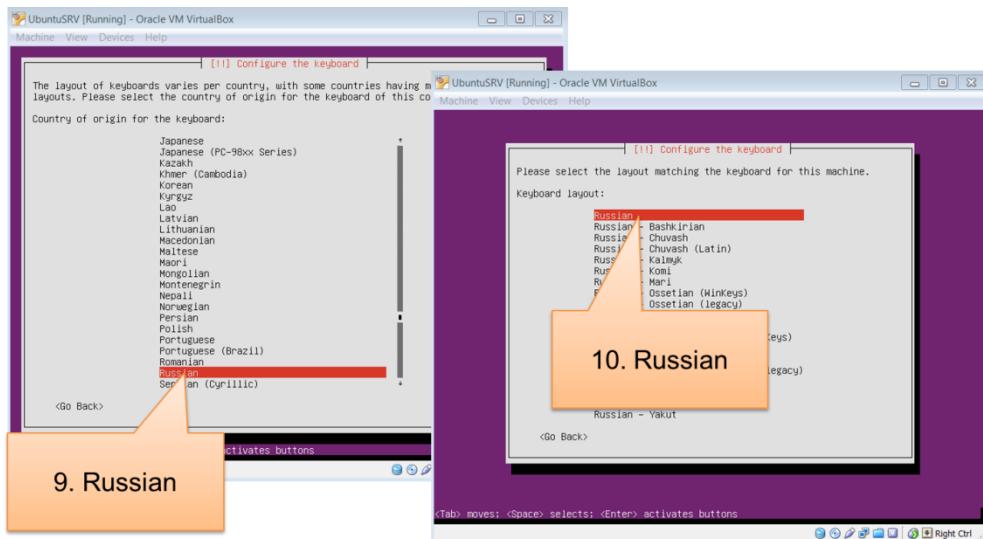
Укажите локаль en\_US.UTF-8:



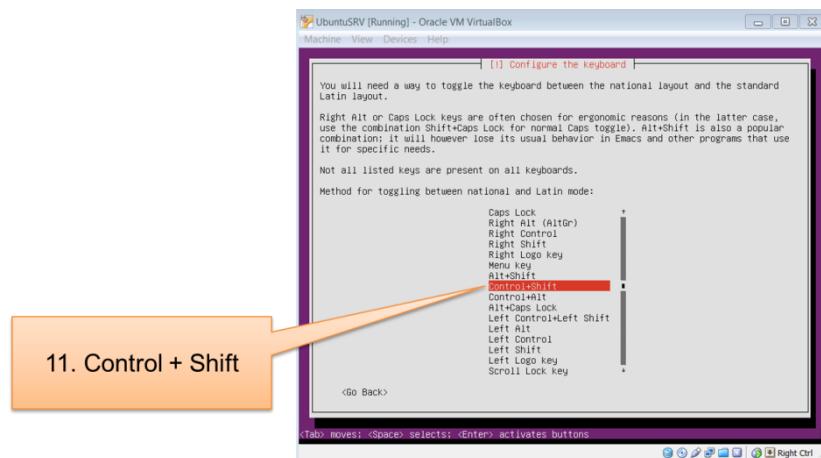
Откажитесь от автоопределения раскладки клавиатуры:



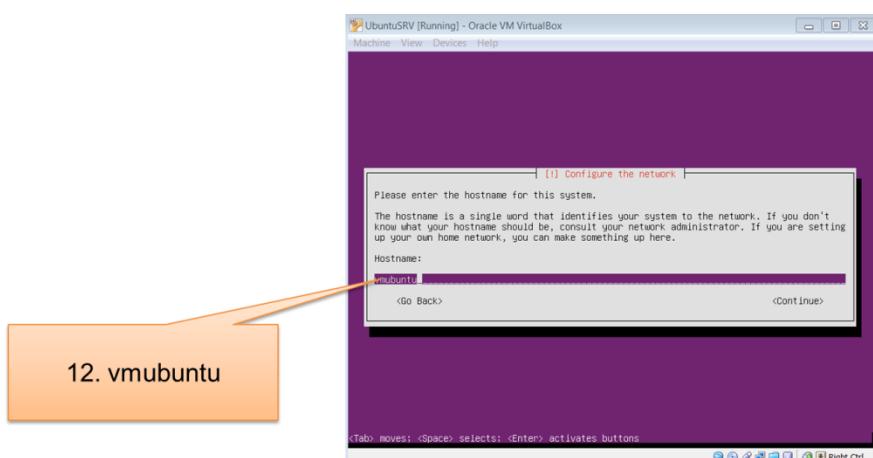
Выберите русскую раскладку:



В качестве комбинации клавиш для переключения языка ввода выберите то, что удобно вам (обычно выбирают Ctrl+Shift):



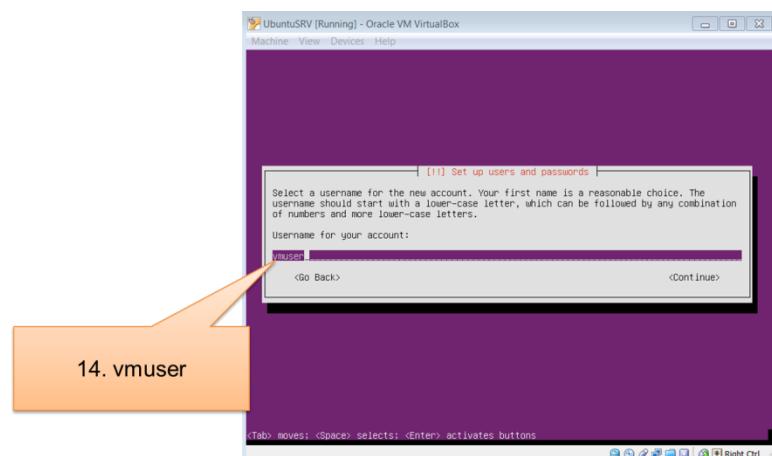
Укажите сетевое имя вашей виртуальной машины (в нашем случае это будет vmubuntu):



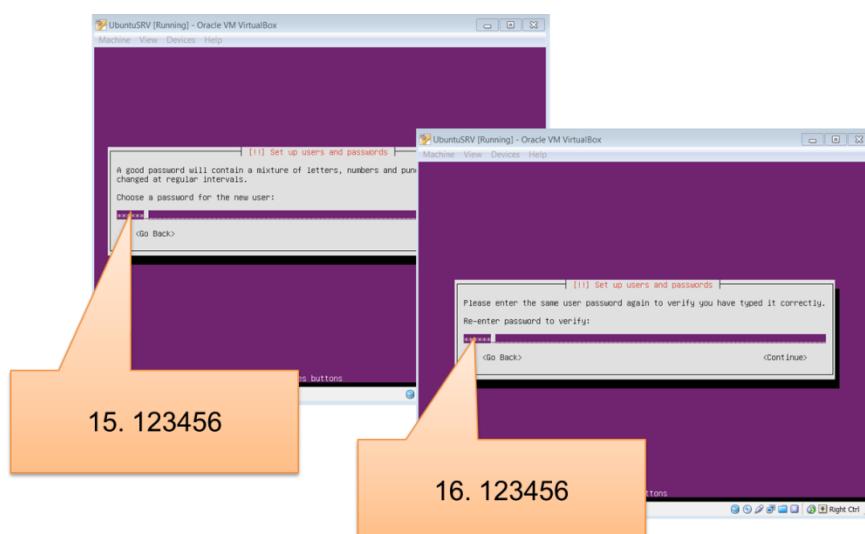
Укажите полное имя вашего пользователя (мы укажем Virtual User):



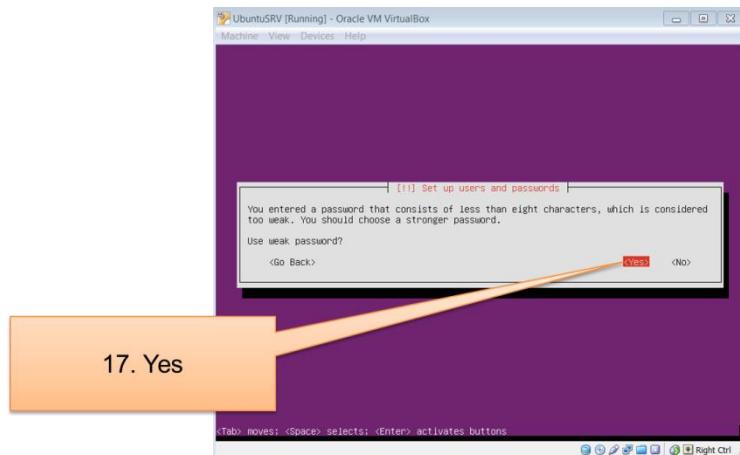
Укажите логин вашего пользователя (мы укажем vmuser):



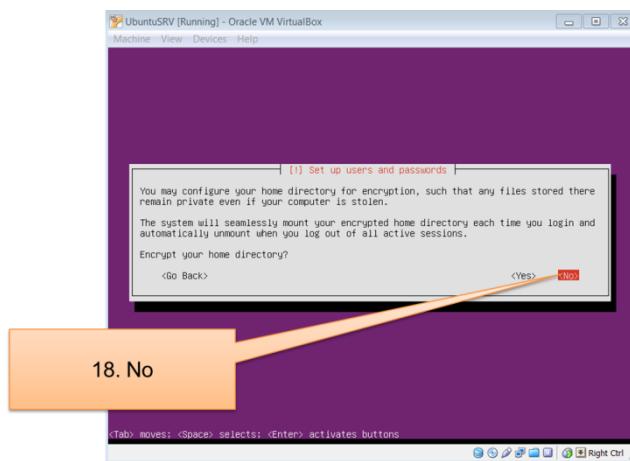
Укажите и подтвердите пароль вашего пользователя (мы укажем 123456):



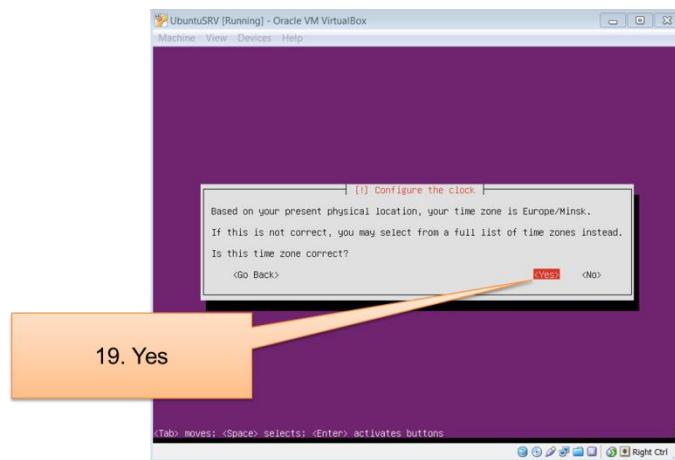
Да, мы знаем, что это слишком простой для подбора пароль. Но это – учебная ВМ, соглашаемся на его использование.



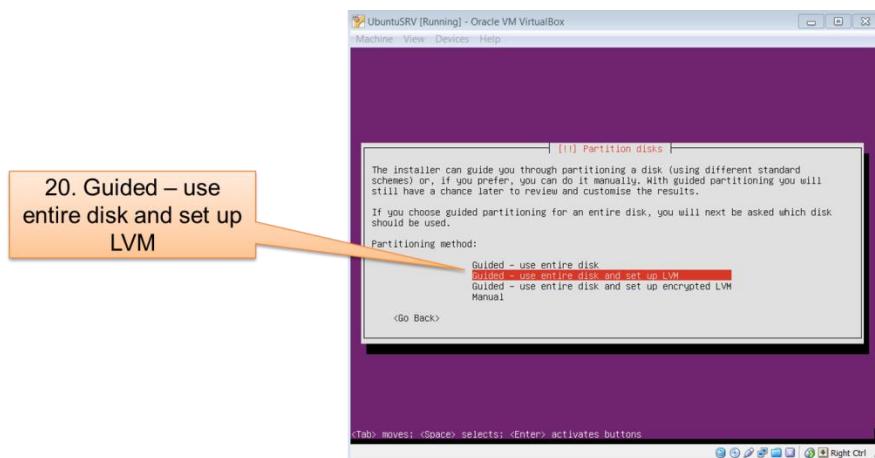
Откажитесь от шифрования домашней папки:



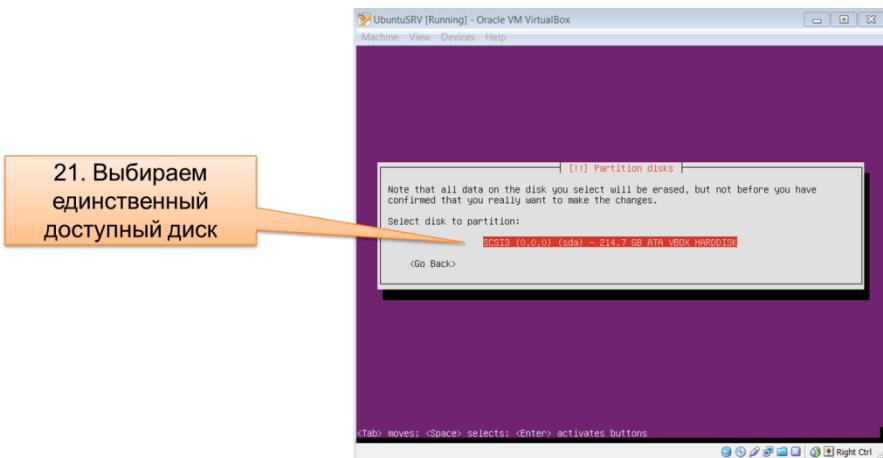
Теперь инсталлятор определит ваше местонахождение и временную зону. Как правило, он всё определяет верно.



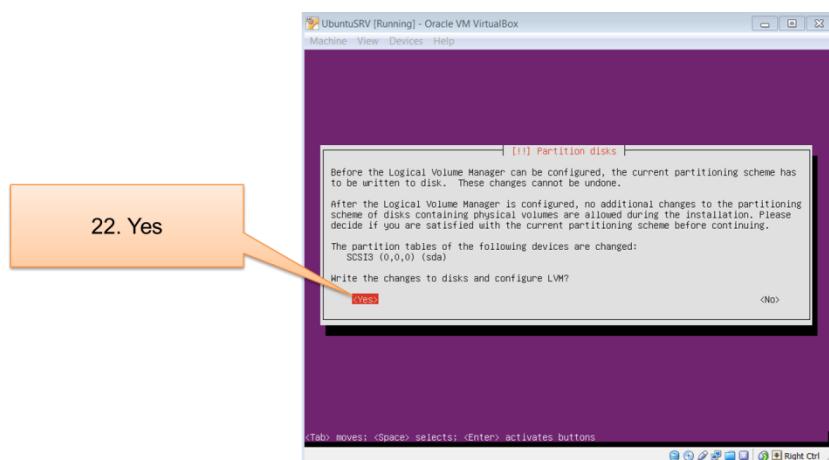
Выберите «использовать весь диск и установить LVM»:



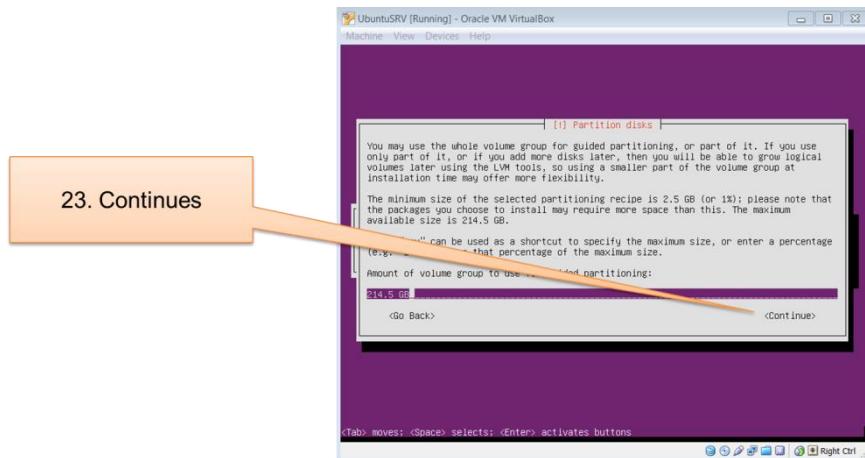
Если вы всё делали по инструкции, то сейчас вам доступен только один диск для установки:



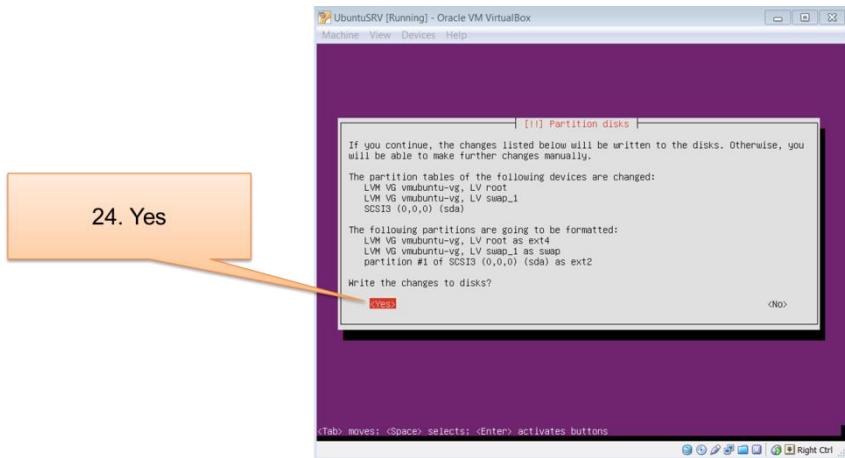
Согласитесь на продолжение установки:



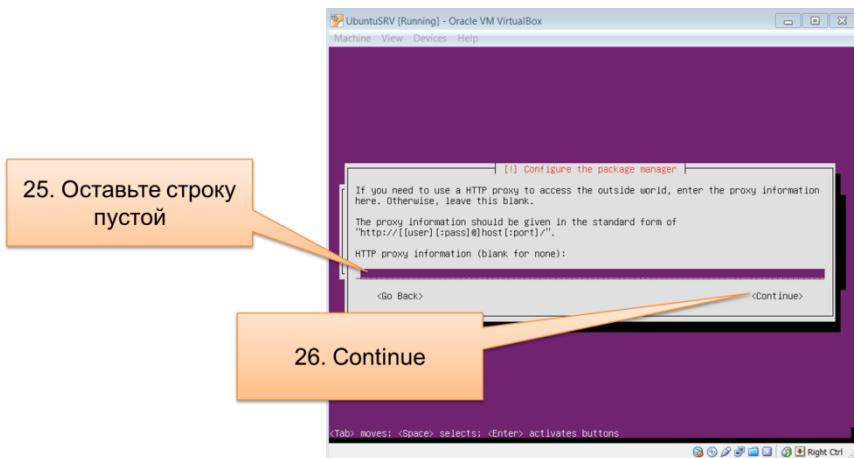
Отведите под устанавливаемую ВМ весь объём виртуального диска (выбрано по умолчанию):



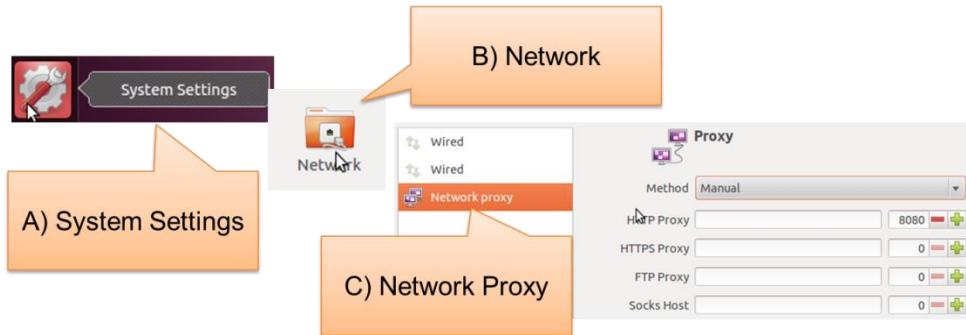
Согласитесь на продолжение установки:



Настройте доступ к прокси (или оставьте строку пустой, если у вас прямой выход в Интернет):



Если выход в интернет у вас всё же осуществляется через Proxy, вы можете настроить его использование потом так:



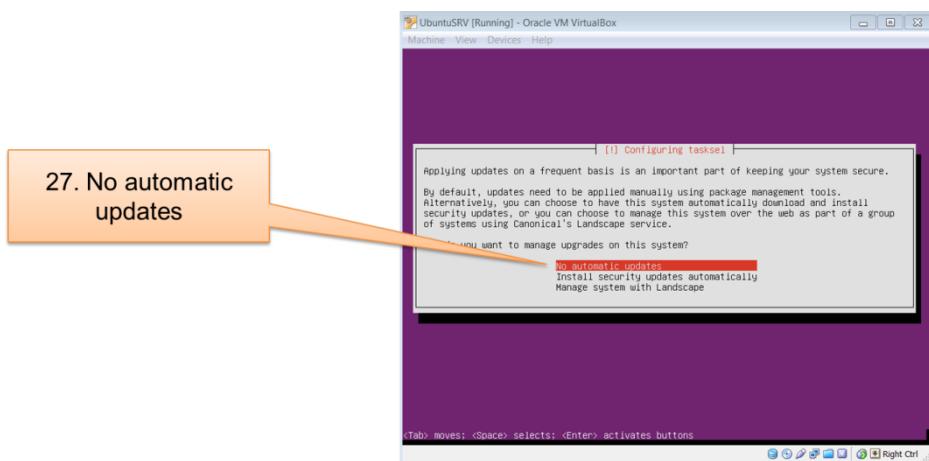
Для настройки Proxy без графического интерфейса добавьте в файл /etc/environment строки:

```
http_proxy="http://user:pass@myproxy.com:port/"
https_proxy="http://user:pass@myproxy.com:port/"
ftp_proxy="http://user:pass@myproxy.com:port/"
no_proxy="localhost,127.0.0.1,localaddress,.localdomain.com"
HTTP_PROXY="http://user:pass@myproxy.com:port/"
HTTPS_PROXY="http://user:pass@myproxy.com:port/"
FTP_PROXY="http://user:pass@myproxy.com:port/"
NO_PROXY="localhost,127.0.0.1,localaddress,.localdomain.com"
```

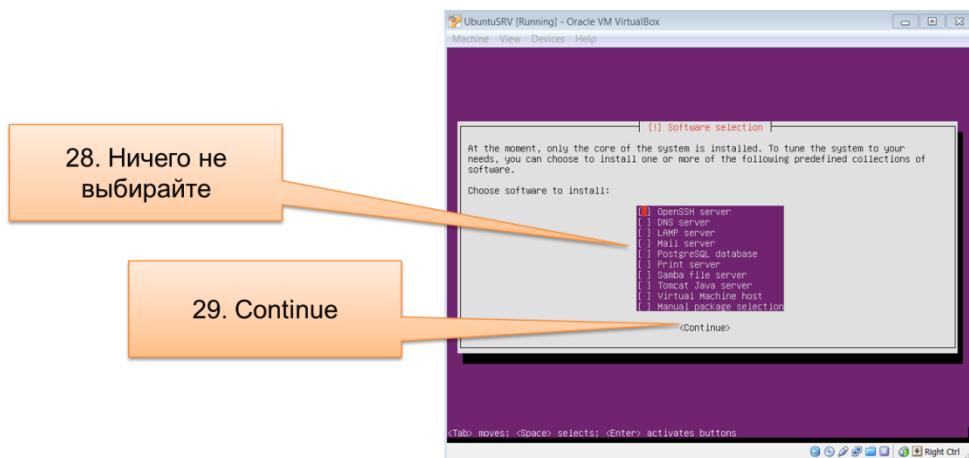
Затем создайте файл /etc/apt/apt.conf.d/95proxies с таким содержимым:

```
Acquire::http::proxy "http://user:pass@myproxy.com:port/";
Acquire::ftp::proxy "ftp://user:pass@myproxy.com:port/";
Acquire::https::proxy "https://user:pass@myproxy.com:port/";
```

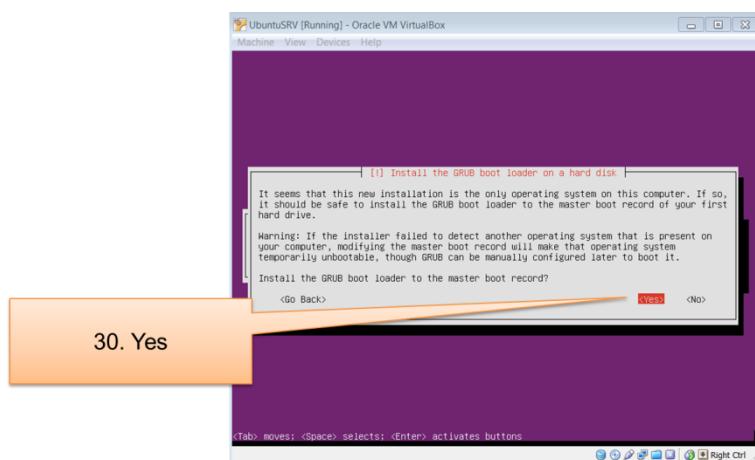
Чтобы минимизировать «скрытую деятельность» ВМ откажитесь от автоматической установки обновлений:



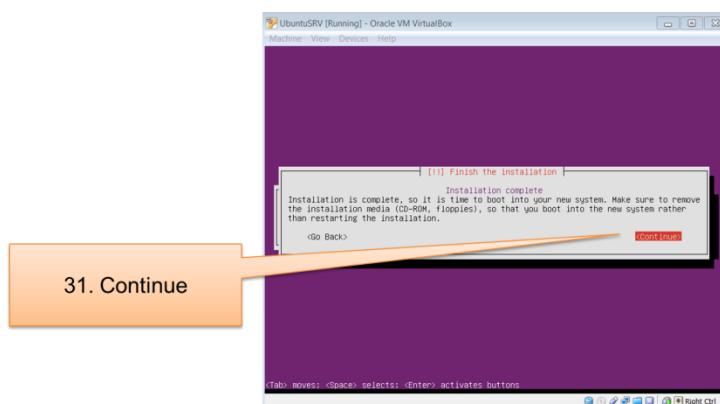
На этом этапе вы можете выбрать готовые наборы установки ПО, но рекомендуется пока отказаться, чтобы потом установить вручную нужные версии:



Согласитесь с предложением записать GRUB в MBR:



Установка завершена. Инсталлятор сам размонтирует образ установочного диска, так что можно просто перезагружать ВМ:



Всё, ОС установлена. Перед дальнейшими действиями рекомендуется экспортировать образ созданной ВМ, чтобы всегда можно было быстро вернуться к «чистой» инсталляции.

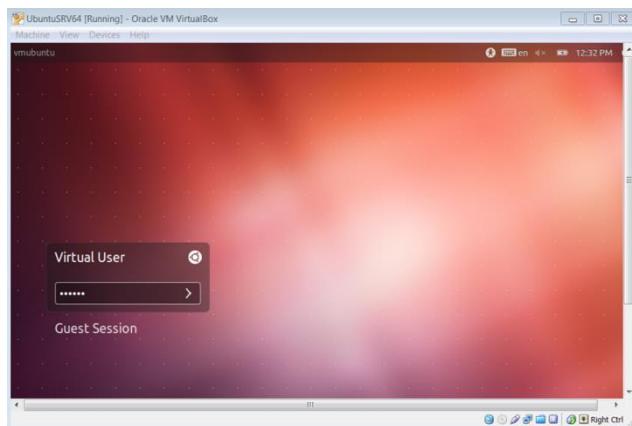
### 2.1.3. Настройка Ubuntu Server

Итак, «чистая» система у нас установлена и «забэкаплена». Сейчас будем ставить софт. И начнём с графического интерфейса (всё же, увы, большинству он понятнее, чем командная строка).

Выполним команду:

```
sudo apt-get install ubuntu-desktop
```

По завершении установки перезагрузим систему. После перезагрузки всё становится красивым 😊.



Доустановим Guest Additions для более удобной работы с ВМ:



Теперь разрешение экрана «подстраивается» само и работают некоторые иные «плюшки».

Прежде, чем продолжить, обновим всю систему. Откроем терминал и выполним команды (чуть быстрее терминал можно открыть клавиатурной комбинацией Ctrl-Alt-T.):



```
sudo apt-get update
sudo apt-get upgrade
```

Для управления «веб-окружением» установим WebMin:  
<http://www.webmin.com/download.html>

В терминале выполним команды:

```
sudo apt-get install perl libnet-ssleay-perl openssl
libauthen-pam-perl libpam-runtime libio-pty-perl
apt-show-versions python

wget http://prdownloads.sourceforge.net/webadmin/webmin_1.700_all.deb

sudo dpkg -i webmin_1.700_all.deb
```

Берите самую свежую версию!

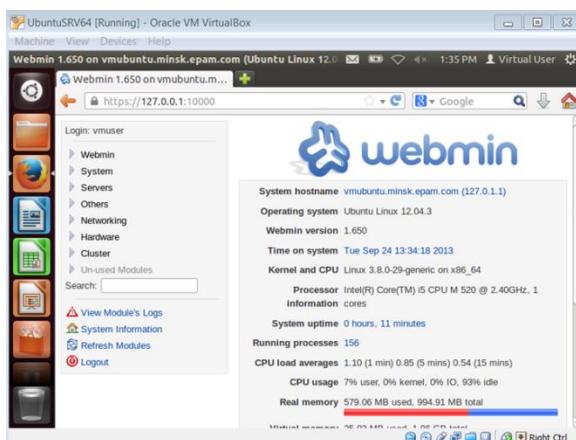
Если сразу не сработает, выполните ещё команду:

```
sudo apt-get install -f
```

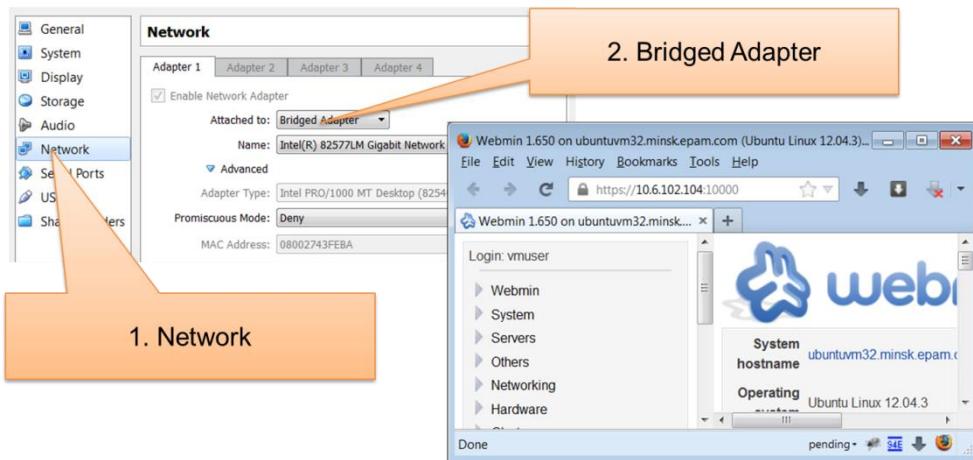
Проверим работоспособность. В браузере наберём

<https://localhost:10000>

и авторизуемся (vmuser:123456)



Чтобы получить доступ к WebMin из хостовой операционной системы, надо переключить сетевой адаптер в режим моста и перезагрузить Ubuntu.



Если у вас не настроен DHCP-сервер, ip-адрес Ubuntu придётся назначать вручную. Для этого откроем файл **/etc/network/interfaces** и внесём нужные правки:

```
## Для автоматической конфигурации
auto eth0
iface eth0 inet dhcp

## Для ручной конфигурации
auto eth0
iface eth0 inet static
address 192.168.1.1
gateway 192.168.1.1
netmask 255.255.255.0
network 192.168.1.0
broadcast 192.168.1.255

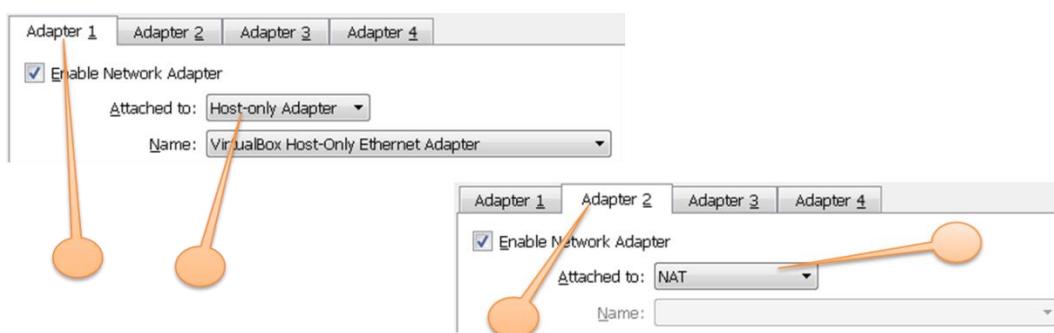
sudo /etc/init.d/networking restart
```

Закомментируйте ненужный вариант

Обновление настроек

Есть и ещё один – самый универсальный способ настройки сети в гостевой ОС:

- В настройках ВМ активировать два сетевых адаптера.
- Первый перевести в режим Host-only Adapter.
- Второй оставить в режиме NAT.



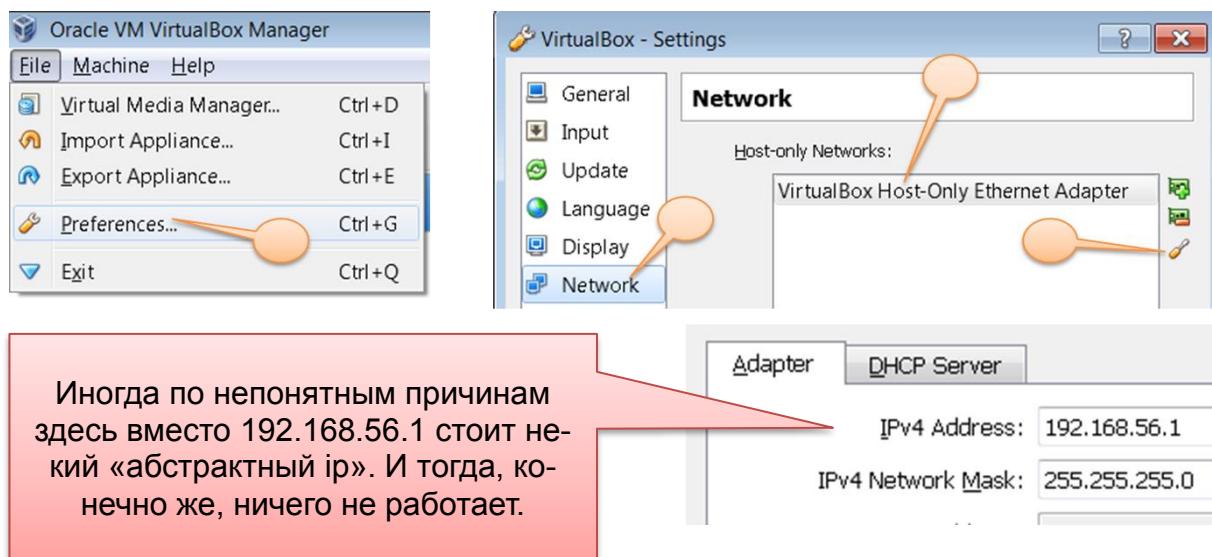
После этого в файле **/etc/network/interfaces** надо написать следующее:

```
auto eth0
iface eth0 inet dhcp

auto eth1
iface eth1 inet dhcp
```

Теперь ваша ВМ доступна с хостовой машины по ip наподобие 192.168.56.100 (используйте команду ifconfig для определения ip), и при этом на ВМ доступен интернет. Перезагрузите виртуальную машину.

**ВНИМАНИЕ!** Убедитесь, что в настройках VirtualBox прописан правильный ip-адрес VirtualBox Host-only Ethernet Adapter:

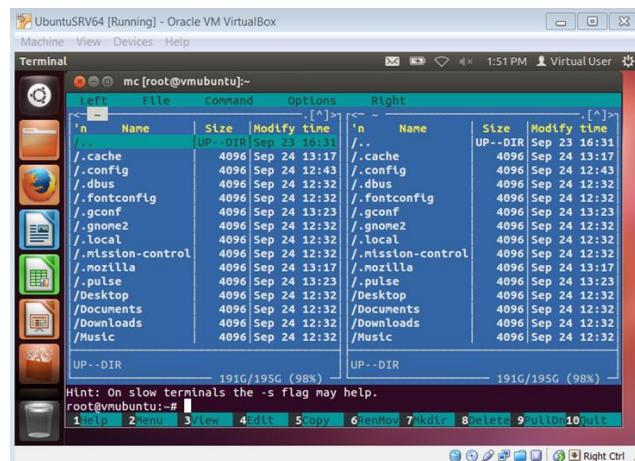


Ещё для удобства установим Midnight Commander:

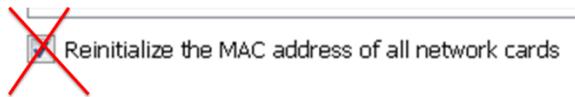
```
sudo apt-get install mc
```

После чего запустим mc от имени root:

```
sudo mc
```

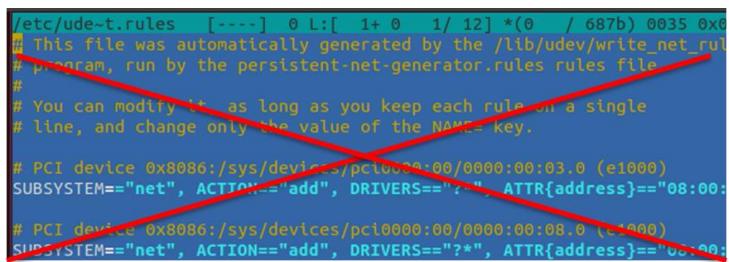


Теперь можно ещё раз сделать экспорт ВМ, импорт её на другом «физическом компьютере» и проверку работоспособности. Если при импорте вы укажете VirtualBox'у необходимость переинициализировать MAC-адреса сетевых адаптеров (а это НАДО сделать!), вы обнаружите неприятный «сюрприз» – сети под Ubuntu нет.



Это не баг, это фича Ubuntu: при обнаружении смены MAC-адреса адаптер ethN (который работал с этим адресом) выключается, и включается адаптер eth(N+1), который остаётся ненастроенным.

Чтобы вернуть всё в норму, надо удалить содержимое файла `/etc/udev/rules.d/70-persistent-net.rules` и перезагрузить Ubuntu.



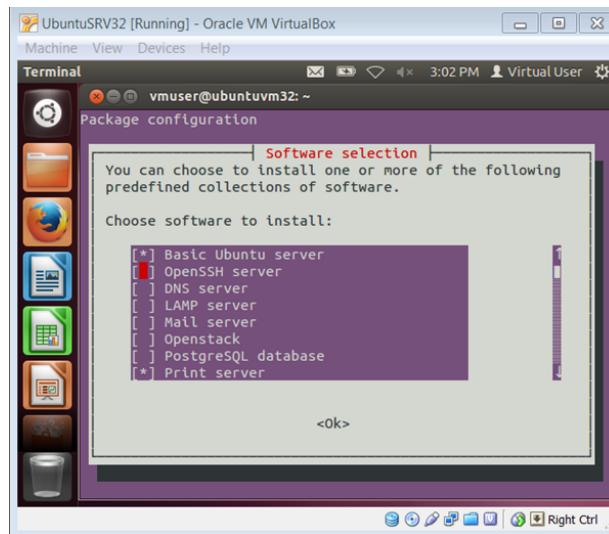
## 2.2. Установка, настройка и использование LAMP и LEMP

### 2.2.1. Упрощённый способ установки

Существует два способа установки LAMP – простой и рекомендуемый и «долгий, но интересный». Начнём с простого.

Выполним команды:

```
sudo apt-get install tasksel
sudo tasksel
```



Здесь можно сразу поставить «много всего».

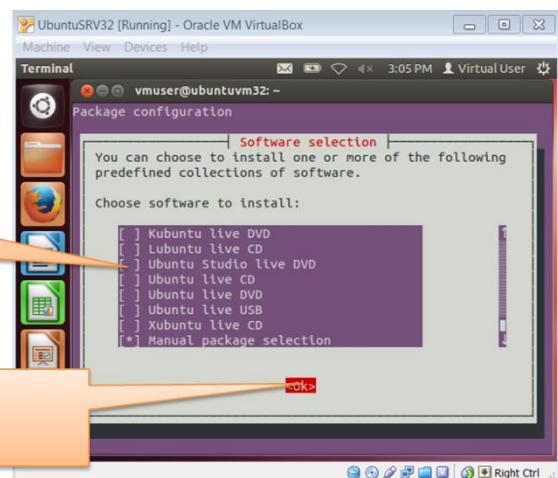
**ВАЖНО!** Не «снимайте галочки» там, где они уже есть! Это может привести к удалению элементов системы!

Выберем:

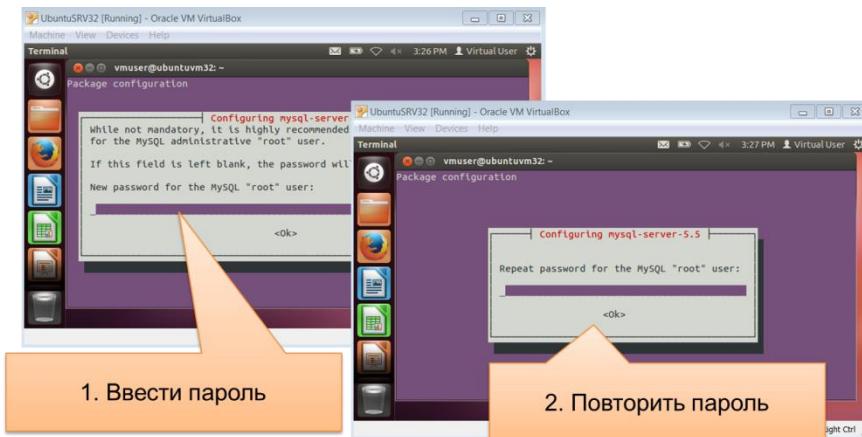
- LAMP Server
- Mail Server

1. Выбрать нужное

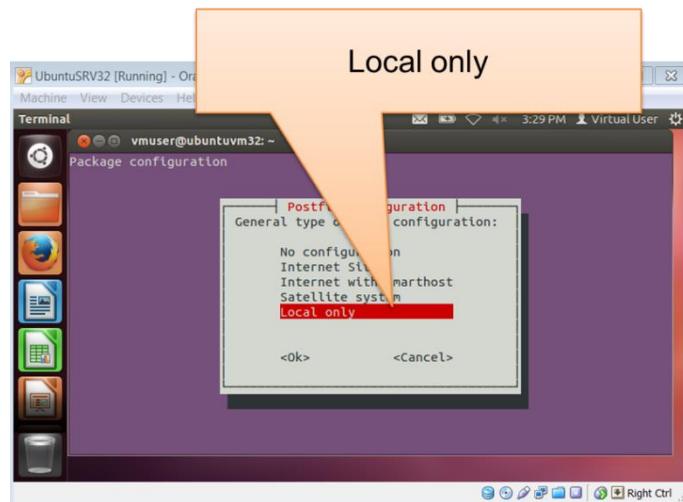
2. OK



По завершении установки система попросит вас ввести и повторить пароль для MySQL-пользователя root:

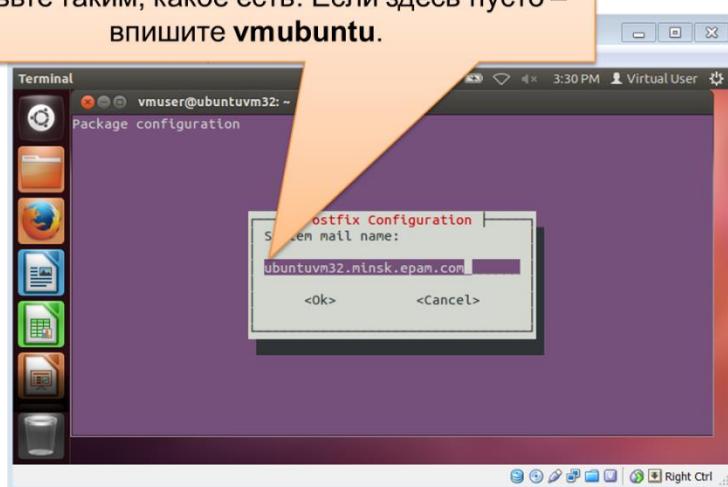


При настройке почтового сервера выберите «Local Only»:



Оставьте без изменений предложенное почтовое имя:

Оставьте таким, какое есть. Если здесь пусто – впишите **vmubuntu**.



Добавим phpMyAdmin для удобства работы с MySQL.

Загрузите архив **tar.gz** с <http://phpmyadmin.net> и распакуйте его содержимое в папку **/var/www/phpmyadmin** (для дальнейшей авторизации используйте: **root:123456**)

В самой же папке (/var/www) создайте файл **phpinfo.php** со следующим содержимым:

```
<?php
    phpinfo();
?>
```



Важно! В некоторых версиях Apache по умолчанию DocumentRoot указан как **/var/www/html**, в таком случае вам придётся вручную изменить его на **/var/www** в файлах

```
/etc/apache2/sites-available/000-default.conf
```

и

**/etc/apache2/sites-available/000-default-ssl.conf**, после чего нужно перезапустить Apache: **sudo service apache2 restart**

Добавим FTP-сервер:

```
sudo apt-get install vsftpd
```

Отредактируем файл **/etc/vsftpd.conf**

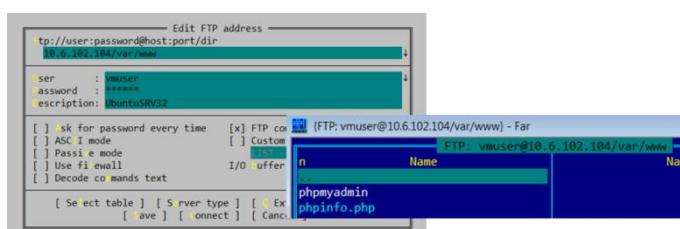
```
anonymous_enable=NO
local_enable=YES
write_enable=YES
chown_upload=YES
chown_username=vmuser
```

Подправим права на папку **/var/www** и перезапустим FTP-сервер:

```
sudo chown vmuser:vmuser /var/www
sudo service vsftpd restart
```

Создайте в любом FTP-клиенте соединение со следующими параметрами:

- Host: ip-адрес вашей ВМ
- Default folder: /var/www
- User: vmuser
- Password: 123456



## Добавим и настроим SSH-сервер:

```
sudo apt-get install openssh-server
```

В общем случае, всё заработает «сразу» 😊, но на всякий случай помним о конфигурационном файле:

```
/etc/ssh/sshd_config
```

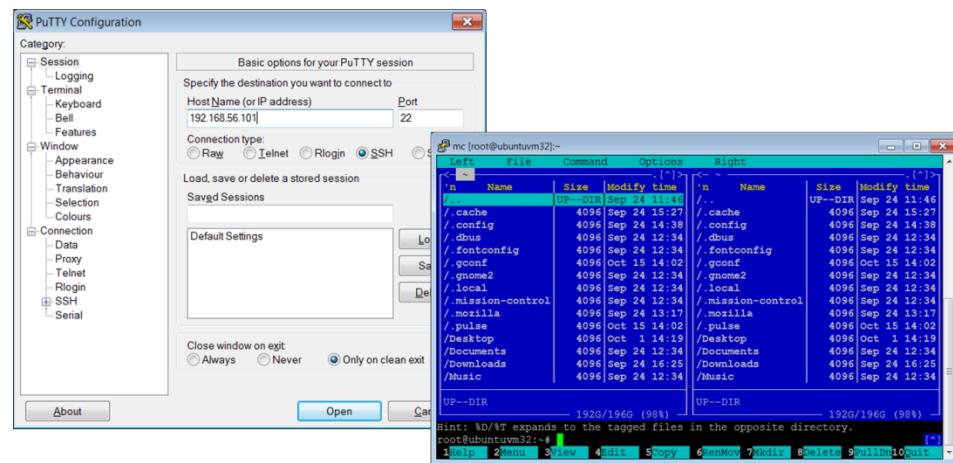
Подробности по конфигурированию:

<https://help.ubuntu.com/community/SSH/OpenSSH/Configuring>

В качестве клиента под Windows чаще всего используется PuTTY:

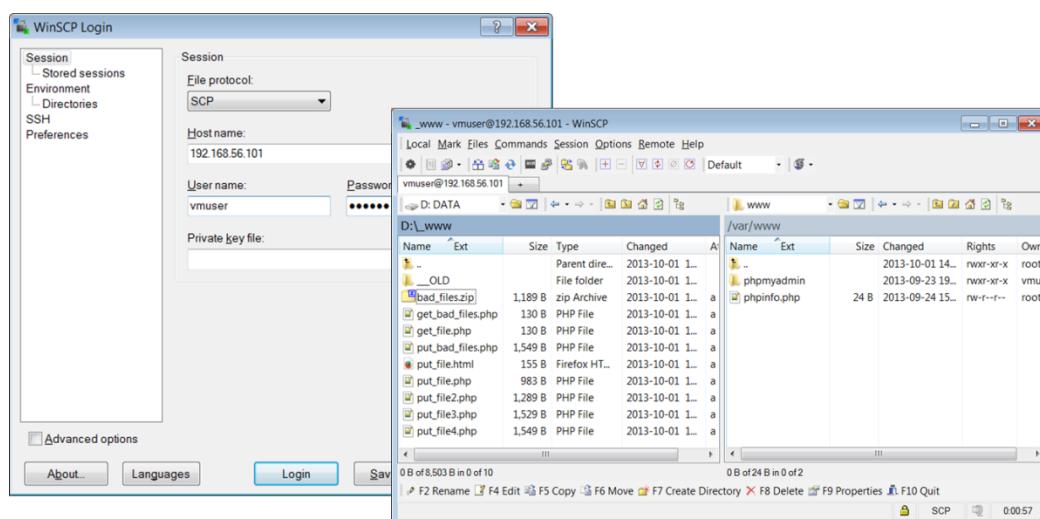
<http://www.putty.org/>

Установив PuTTY под свою windows-систему, проверьте работоспособность SSH-соединения:



Для безопасной передачи файлов вам может понадобиться WinSCP:

<http://winscp.net/eng/download.php>



Добавим и настроим samba-сервер:

```
sudo apt-get install samba
```

Отредактируем файл `/etc/samba/smb.conf`

```
workgroup = PHP
```

И в конец добавим:

```
[www]
comment = Apache Document Root
valid users = vmuser
path = /var/www
available = yes
public = yes
writable = yes
browseable = yes
create mask = 0777
directory mask = 0777
guest ok = no
read only = no
```

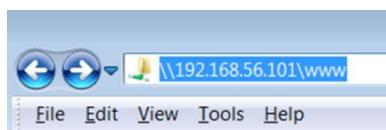
Настроим пароль samba-пользователя:

```
sudo smbpasswd -a vmuser
```

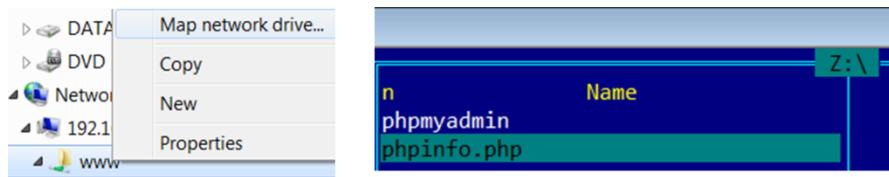
Перезапустим сервер:

```
sudo restart smbd
sudo restart nmbd
```

Теперь вы можете подключаться к ВМ по сети:



Используйте логин **vmuser** и пароль, который вы только что задали (на готовых учебных образах – **123456**). Вы также можете для удобства примонтировать эту сетевую папку как сетевой диск:



**ВСЁ!** Теперь вы можете работать с только что настроенным сервером «удалённо» – через браузер, SSH, FTP, SMB.

**Внимание!** Всегда при наличии такой возможности используйте протоколы связи, работающие на основе SSL/TLS, т.к. это снижает вероятность перехвата трафика между вашей машиной и сервером.

К защищённым протоколам относятся: https, sftp, ftps, scp.

Подробнее о принципах работы данных протоколов можно почитать здесь:  
<https://ru.wikipedia.org/wiki/TLS>

## 2.2.2. Более сложный путь установки

Сложный (пусть и не самый сложный, мы не будем собирать LAMP и т.д. из исходников) пусть состоит в использовании PPA (Personal Package Archives).

Нам понадобится:

<https://launchpad.net/~nathan-renniewaldock/+archive/apache2.4-php>

затем

<https://launchpad.net/~ondrej/+archive/mysql-5.6>

и

<https://launchpad.net/~nginx/+archive/stable>

По соответствующим ссылкам найдём имена PPA:

- ppa:nathan-renniewaldock/apache2.4-php
- ppa:ondrej/mysql-5.6
- ppa:nginx/stable

Подключим их как источники инсталляции в Ubuntu:

```
sudo add-apt-repository ppa:nathan-renniewaldock/apache2.4-php
sudo add-apt-repository ppa:ondrej/mysql-5.6
sudo add-apt-repository ppa:nginx/stable
sudo apt-get update
sudo apt-get upgrade
```

Может не устано-  
виться.

Если в процессе установки вы увидите сообщение «The following packages have been kept back: ...», попробуйте выполнить команду:

```
sudo apt-get dist-upgrade
```

Как вариант, можно подключить PPA с последней версией PHP:

- ppa:ondrej/php5

Теперь выполним команды:

```
sudo apt-get install apache2
sudo apt-get install php5
sudo apt-get install mysql-server php5-mysql
sudo apt-get install libapache2-mod-php5
```

Опять же: если в процессе установки вы увидите сообщение «The following packages have been kept back: ...», попробуйте выполнить команду:

```
sudo apt-get dist-upgrade
```

Если MySQL не установится с первого раза, можно попробовать повторить команду

```
sudo apt-get install mysql-server php5-mysql
```

Вообще, при данном подходе нет «универсального чёткого алгоритма», разве что есть один точно сработавший:

1. Ставим MySQL из стандартного репозитория (из PPA может не установиться).
  2. Добавляем PPA `ppa:nathan-renniewaldock/apache2.4-php` и выполняем `update/upgrade`.
  3. Добавляем PPA `ppa:ondrej/php5` и снова выполняем `update/upgrade`.
  4. Выполняем команды

```
sudo apt-get install apache2  
sudo apt-get install php5  
sudo apt-get install mysql-server php5-mysql  
sudo apt-get install libapache2-mod-php5
```

После этого, как правило, всё работает. Удачи ☺.

### 2.2.3. Установка и настройка nginx

Теперь пришло время установить nginx. Для начала остановим Apache, чтобы освободить 80-й порт:

```
sudo service apache2 stop
```

И выполним установку nginx:

```
sudo apt-get install php5-common php5-cli php5-fpm
sudo apt-get install nginx
sudo service nginx start
sudo service nginx stop
```

Внесём правки в **/etc/nginx/sites-available/default**

В секции server:

```
listen 81;
listen [::]:81 default ipv6only=on;

#root /usr/share/nginx/www;
root /var/www;
index index.php index.html index.htm;
```

В секции location ~ \.php\$ { :

```
location ~ \.php$ {
    fastcgi_split_path_info ^(.+\.php)(/.+)$;
    fastcgi_pass unix:/var/run/php5-fpm.sock;
    fastcgi_index index.php;
    include fastcgi_params;
}
```

Внесём правку в /etc/php5/fpm/php.ini :

```
cgi.fix_pathinfo=0
```

Внесём правку /etc/php5/fpm/pool.d/www.conf :

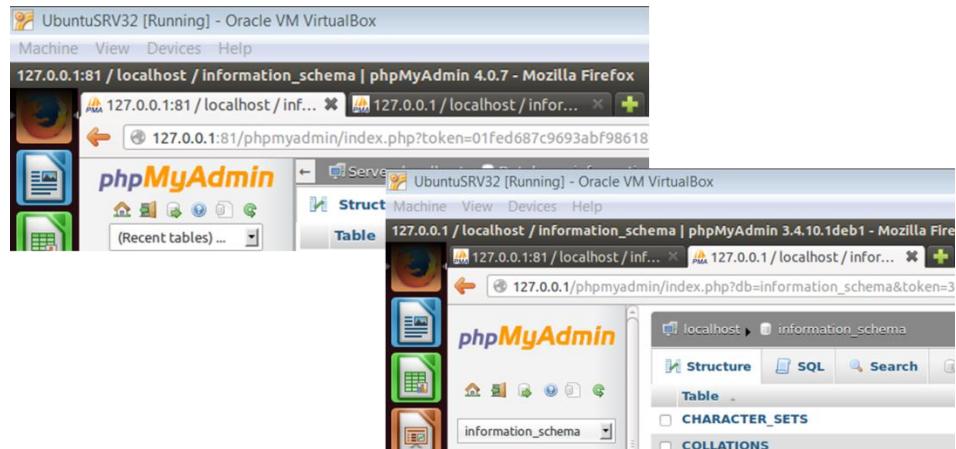
```
;listen = 127.0.0.1:9000
listen = /var/run/php5-fpm.sock
```

В новых версиях этот параметр может изначально иметь верное значение.

Перезапустим сервисы и запустим Apache:

```
sudo service php5-fpm restart
sudo service nginx restart
sudo service apache2 start
```

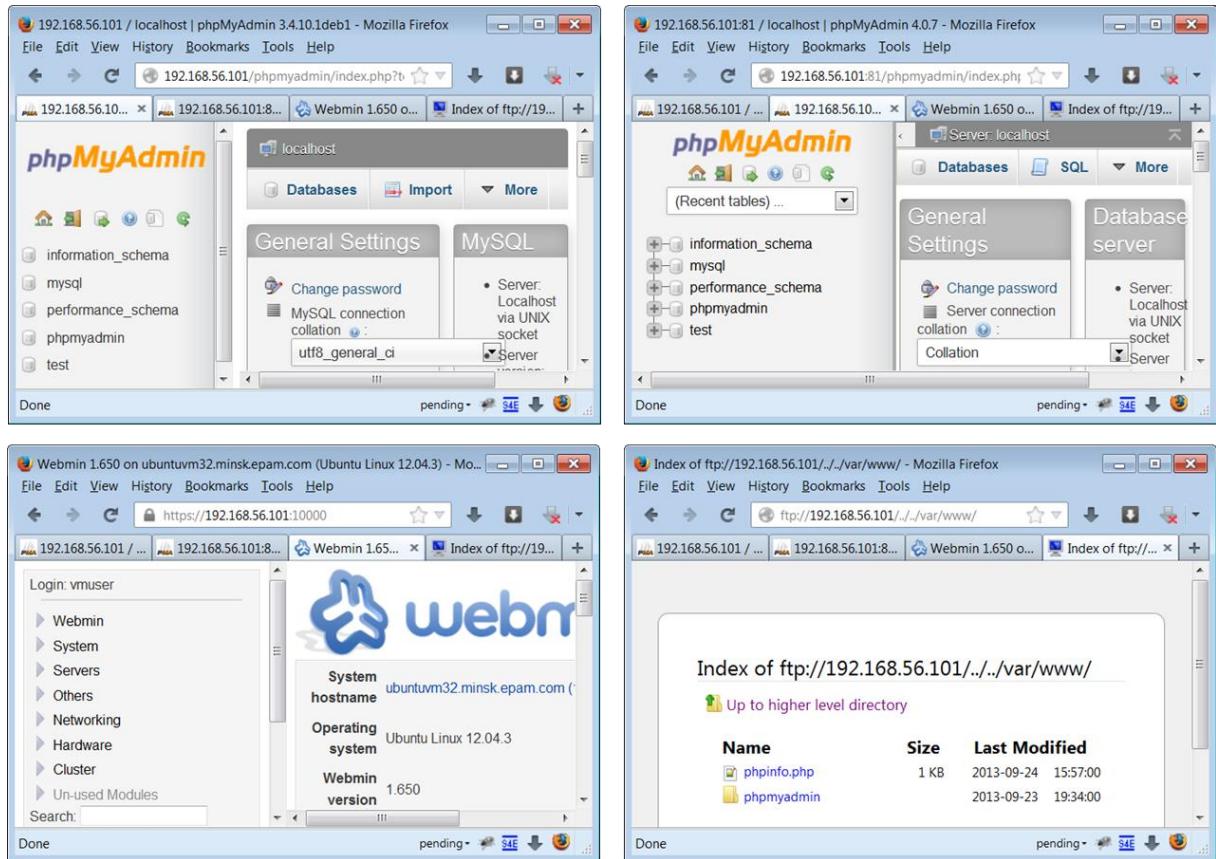
Теперь по порту 80 отвечает Apache, а по 81 – nginx:



## 2.2.4. Как теперь всем этим пользоваться

Если не вдаваться в подробности тонкого конфигурирования, сейчас мы получили вполне рабочий веб-сервер под управлением Linux. «Извне» его можно использовать так:

- FTP для передачи файлов в /var/www
- Браузер для проверки результатов работы приложений (по 80-му порту – Apache, по 81-му -- nginx).
- WebMin (по 10000-му порту для управления системой, HTTPS!!!).



## 2.3. Использование Windows в разработке на PHP

Если по каким-то причинам вам придётся настраивать WAMP/WEMP под Windows, то выглядит это примерно так.

Установку и настройку самой Windows пропустим, полагая это очевидной задачей. Допустим, у вас есть «свежая», только что установленная Windows 7.

**Внимание! Начиная с PHP 5.5.x поддержка Windows XP прекращена!**

Скачайте последние версии следующего ПО:

- PHP: <http://windows.php.net/download/>
- Apache: <http://www.apachelounge.com/download/>
- VC++ redistributable: <http://www.microsoft.com/en-us/download/details.aspx?id=30679>
- MySQL: <http://dev.mysql.com/downloads/>
- nginx: <http://nginx.org/en/download.html>
- PhpMyAdmin: <http://www.phpmyadmin.net>
- Notepad++: <http://notepad-plus-plus.org/download/>

Скачайте, установите и настройте Firefox и нужные вам плагины, FAR и прочее полезное с вашей точки зрения ПО.

Создайте следующее дерево каталогов (у вас могут быть другие номера версий, тут они указаны для простоты запоминания того, что у нас установлено):

```
C:/WebSoft
    Apache2410
    PHP560
    nginx161
    MySQL5620
    NPP668
    PhpStorm714
C:/WebData
    phpmyadmin
```

Установите VC++ redistributable, распакуйте дистрибутив Apache, и внесите следующие правки в файл **C:/WebSoft/Apache2410/conf/httpd.conf**

Замените все вхождения

C:/Apache24

на

C:/WebSoft/Apche2410

Замените все вхождения

C:/WebSoft/Apache2410/htdocs

на

C:/WebData

Замените все вхождения

AllowOverride None

на

AllowOverride All

Раскомментируйте строку

#LoadModule rewrite\_module modules/mod\_rewrite.so

Раскомментируйте строку

#Include conf/extrা/httpd-autoindex.conf

В строке  
DirectoryIndex index.html  
добавьте  
index.php  
перед  
index.html

В конец файла допишите строки:  
LoadModule php5\_module "C:/WebSoft/PHP560/php5apache2\_4.dll"  
AddHandler application/x-httpd-php .php  
PHPIniDir "C:/WebSoft/PHP560"

Распакуйте дистрибутив PHP, скопируйте файл **php.ini-development** под именем **php.ini** и внесите следующие правки:

Раскомментируйте параметр  
**;extension\_dir**  
И измените его значение на  
**extension\_dir = "C:/WebSoft/PHP560/ext"**

Раскомментируйте строку  
**;date.timezone**  
И изменить её значение на  
**date.timezone = "Europe/Minsk"**

Раскомментируйте строки  
**extension=php\_curl.dll**  
**extension=php\_gd2.dll**  
**extension=php\_mbstring.dll**  
**extension=php\_mysql.dll**  
**extension=php\_mysqli.dll**  
**extension=php\_pdo\_mysql.dll**  
**sendmail\_from = me@example.com**

Установите MySQL, Notepad++ и PhpStorm, следуя инструкциям инсталлятора (в качестве целевых каталогов для установки используйте соответствующие каталоги в **C:/WebSoft**). Распакуйте дистрибутив PhpMyAdmin в каталог **C:/WebData/phpmyadmin**.

При установке MySQL, возможно, понадобится установить .NET Framework:  
<http://www.microsoft.com/en-us/download/details.aspx?id=17851>

Также может понадобиться JRE:

<http://www.oracle.com/technetwork/java/javase/downloads/>

Поскольку в Windows 7 довольно часто 80-й порт занят системным процессом, выполните такую последовательность действий:

Запустите  
**regedit**  
Откройте ветку  
**HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services\HTTP**  
Добавьте новое значение  
**DWORD (32-bit)**  
С именем  
**NoRun**  
и значением  
**1**  
Перезагрузите ОС.

Распакуйте дистрибутив nginx, создайте папку **C:/WebSoft/nginx161/PHP560** и скопируйте в неё только что настроенный PHP.  
После чего исправьте в **php.ini** значение строки  
**extension\_dir**  
на  
**extension\_dir = "C:/WebSoft/nginx161/PHP560/ext"**

Откройте файл **C:/WebSoft/nginx161/conf/nginx.conf** и внесите следующие правки:

```
listen 81; 81-й порт
location / {
    root c:/webdata;
    index index.php index.html index.htm;
}
location ~ \.php$ {
    root c:/webdata;
    fastcgi_pass 127.0.0.1:9000;
    fastcgi_index index.php;
    fastcgi_param SCRIPT_FILENAME c:/webdata/$fastcgi_script_name;
    include fastcgi_params;
}
```

Новый корневой каталог.

Реакция на index.php.

Новый корневой каталог.

Передача PHP-файлов на обработку в PHP.

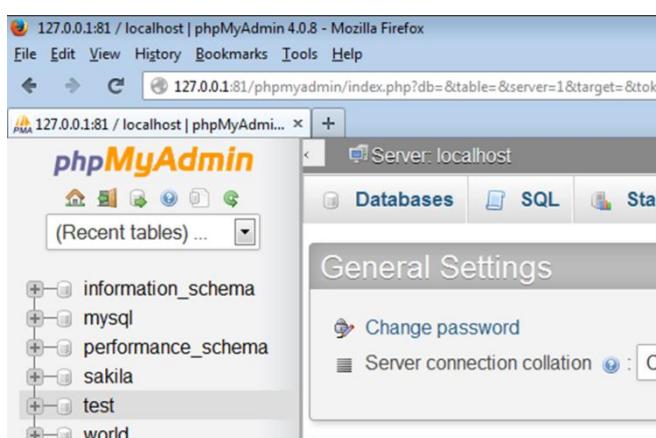
В папке **C:/WebData/nginx161/** создайте следующие cmd-файлы со следующим содержимым:

```
C:/WebSoft/nginx161/nginx.exe
C:/WebSoft/nginx161/PHP560/php-cgi.exe -b 127.0.0.1:9000
-c C:/WebSoft/nginx161/PHP560/php.ini
```

```
taskkill /f /IM nginx.exe
taskkill /f /IM php-cgi.exe
```

Одной строкой!

Проверим работоспособность nginx + PHP + MySQL. Откроем в браузере <http://127.0.0.1:81/phpmyadmin/> и авторизуемся (**root:123456**).



Осталось только установить сервис Apache. Из каталога с бинарными файлами Apache выполните команду (от имени администратора!):

```
httpd.exe -k install -n "Apache246"
```

Если не смотря ни на что Apache-x64 ругается, что PHP-x64 – не 32-битное приложение, скопируйте все dll из **C:/WebSoft/Apache2410/bin** в **C:/WebSoft/PHP560**.

Понять, «чего ему не хватает», очень помогает вот эта утилита:  
<http://www.dependencywalker.com/>.

Проверим работоспособность Apache + PHP + MySQL. Откроем в браузере <http://127.0.0.1/phpmyadmin/> и авторизуемся (**root:123456**).

## 2.4. Отладка и устранение проблем в PHP-приложениях и конфигурации окружения

Поскольку не существует гарантированных универсальных решений для любой потенциальной проблемы, здесь мы рассмотрим источники информации, которые могут нам помочь.

### Лог-файлы

Загляните в папку

/var/log

и поищите там подпапку того приложения, информацию о работе которого вы хотите узнать.

Вот, например, выдержка из лога MySQL:

```
130924 15:31:44 InnoDB: 5.5.32 started; log sequence number 1595675
ERROR: 1064 You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'ALTER TABLE user ADD column Show_view_priv enum('N','Y') CHARACTER SET utf8 NOT '
at line 1
130924 15:31:44 [ERROR] Aborting
```

### Полезные команды для быстрой работы с файлами

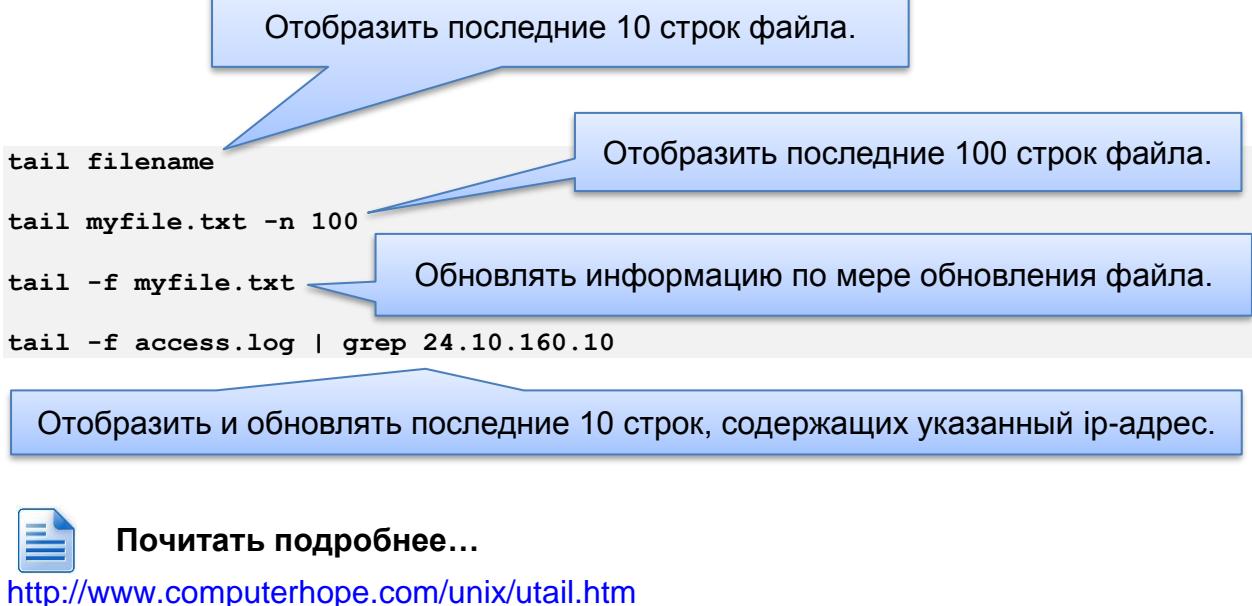
В быстрой работе с файлами помогают команды **cat** и **tail**:

|  |   |
|--|---|
| <pre>cat filename<br/>cat file1 file2 &gt; combinedfile<br/><br/>cat &gt; file.ext<br/>Пишем текст, потом нажимаем Ctrl-Z<br/><br/>cat -n filename</pre> | <p>Отобразить содержимое файла.</p> <p>Объединить содержимое файлов.</p> <p>Создать файл прямо из консоли.</p> <p>Отобразить содержимое файла с нумерацией строк.</p> |
|--|---|



Почитать подробнее...

<http://www.cyberciti.biz/faq/howto-use-cat-command-in-unix-linux-shell-script/>



## Утилиты диагностики

К большинству «серверных» приложений есть специальные утилиты, помогающие понять, что происходит. Например, команда

```
apachectl configtest
```

проверяет корректность конфигурационного файла Apache, а для nginx существует команда

```
nginx -t
```

## Проверка в нескольких окружениях

Здесь всё просто: запустите ваше PHP-приложение на другом сервере, под другой ОС, с другими версиями PHP/Apache/nginx/MySQL/библиотек. Возможно, проблема в совместимости, поддержке каких-то функций и т.д.

P.S. Помните о логах. Если там информации недостаточно – попробуйте увеличить детализацию логирования (debuglevel, loglevel) – многие приложения имеют такую возможность.

## Перезапуск серверов

Часть проблем решается перезапуском серверов.

Под Unix:

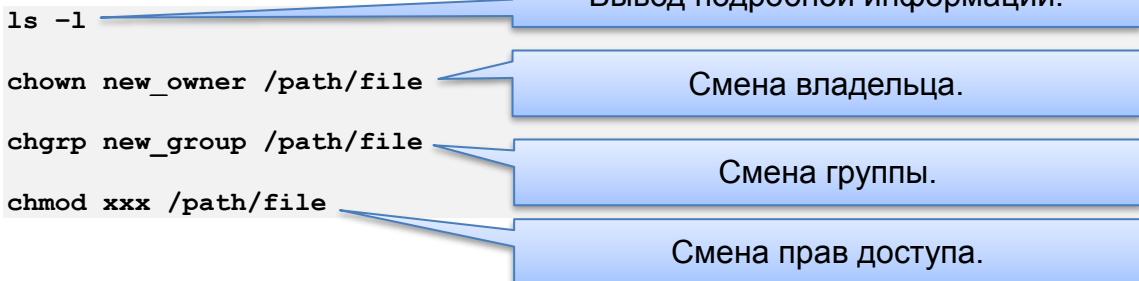
```
service apache2 restart  
service apache2 start  
service apache2 stop  
  
service mysql restart  
service mysql start  
service mysql stop
```

Под Windows:

```
net stop Apache2.4  
net stop MySQL56  
  
net start Apache2.4  
net start MySQL56
```

### Проверка прав доступа к файлам и каталогам

Чтобы узнать и изменить информацию о владельце и правах доступа к объекту файловой системы, используйте команды:



**Почитать подробнее...**

<https://help.ubuntu.com/community/FilePermissions>

### Правильный поиск информации

Иногда самым быстрым способом решения конкретной проблемы является поиск в гугле. Почему некоторые люди ничего не могут там найти? Потому, например, что вместо запроса “ubuntu nginx php bad gateway” они пишут что-то в стиле «не грузится страница».

Ещё хорошим способом поиска решения является «гугление» по конкретному тексту сообщения об ошибке (и номеру ошибки, если он известен).

### Использование отладчика в IDE

Несмотря на то, что встроенный отладчик в PHP обещают только в версии 5.6, уже долгое время существует широко используемый инструмент xdebug.

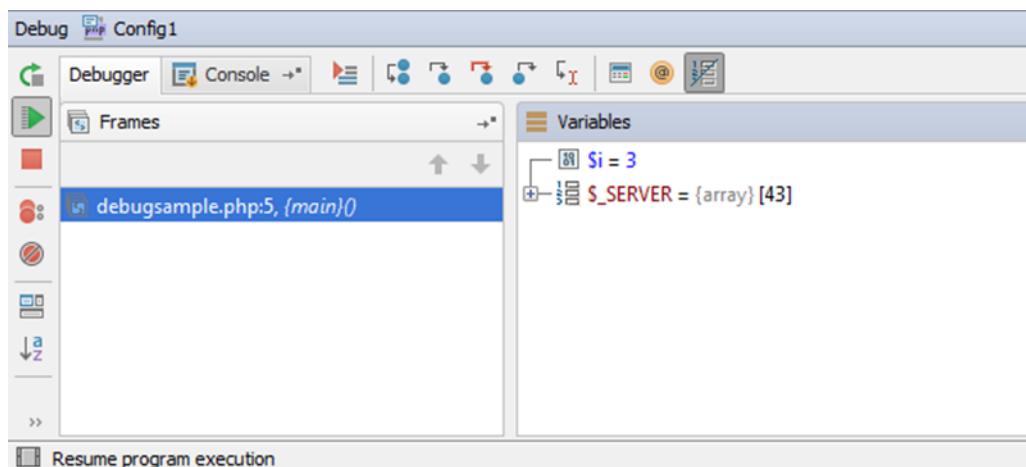
Для его применения скачайте сам xdebug с <http://xdebug.org/download.php>.

Скопируйте полученную DLL в **C:/WebSoft/PHP560/ext** и добавьте в php.ini строку:

```
zend_extension="C:/WebSoft/PHP560/ext/php_xdebug-2.2.5-5.6-vc11.dll"
```

Помните, что у вас может быть другая версия PHP и/или xdebug, и путь может отличаться.

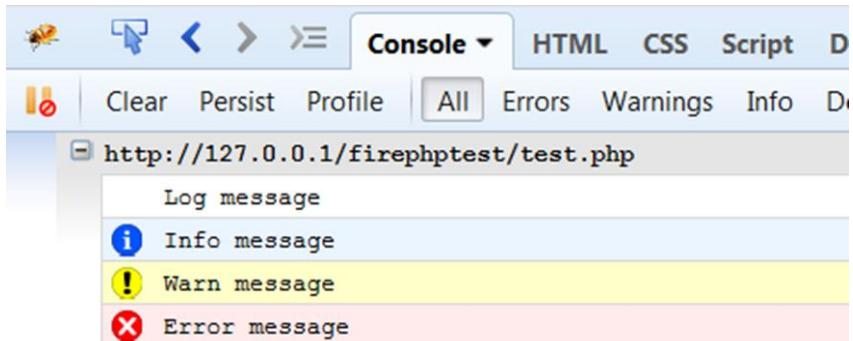
Использование отладки в PhpStorm вполне интуитивно и подобно аналогичной задаче в других IDE. Вот лишь простой визуальный пример:



Немного подробнее – см. в разделе, посвящённому настройке PhpStorm.

### Использование отладочных сообщений в браузере

Не менее широко распространён и ещё один инструмент – FirePHP, который состоит из расширения для Firefox (точнее, для расширения Firebug для Firefox, т.е. сначала надо установить Firebug) и небольшой библиотеки, подключаемой на стороне сервера.



Итак, что нужно сделать для использования FirePHP:

- Установить в Firefox расширение Firebug (оно бесплатное, официальный сайт: <https://getfirebug.com>).
- Установить в Firefox расширение FirePHP (оно тоже бесплатное, официальный сайт: <http://firephp.org>).
- Скачать серверную часть («FirePHPCore Server Library») (отсюда же <http://firephp.org>).
- Подключить основную библиотеку FirePHP:

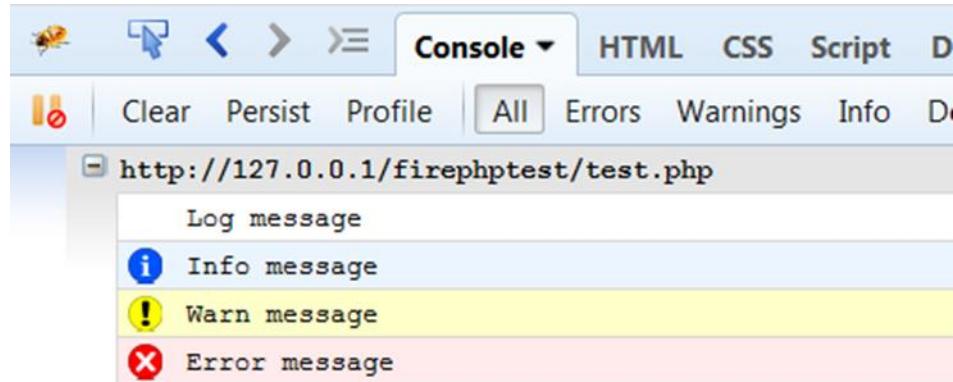
```
<?php
require_once('./FirePHPCore-0.3.2/lib/FirePHPCore/fb.php');
?>
```

Помните, что у вас путь может отличаться.

- Использовать внутри вашего кода следующие отладочные сообщения:

```
<?php  
FB::log('Log message');  
FB::info('Info message');  
FB::warn('Warn message');  
FB::error('Error message');  
?>
```

- Активировать в Firebug панели Console и Net и смотреть на поступающие сообщения:



## 2.5. Простейшие способы развёртывания PHP-приложений

### 2.5.1. Развёртывание веб-приложений

Стратегия развёртывания веб-приложений зависит от сотен факторов, но в общем можно выделить следующий обобщённый алгоритм.

- Выгрузка кода из репозитория на тестовый сервер, обновление БД.
- Прогон тестов. Если тесты проходят, продолжаем.
- Переключение рабочего сервера в режим «ждите, идут работы» ☺.
- Создание «точки восстановления» на резервном сервере (любым способом, позволяющим БЫСТРО отменить все изменения).
- Выгрузка кода из репозитория на рабочий сервер, обновление БД.
- Прогон тестов. Если тесты проходят, сервер переходит в рабочий режим. Если тесты не проходят, восстанавливаем предыдущую версию из бэкапа.

Для индивидуальной (в т.ч. учебной) работы алгоритм ещё проще:

- Создание резервной копии приложения и БД на сервере.
- Обновление БД.
- Обновление приложения.
- Проверка того, что новая версия работает.

### Создание резервной копии

ВСЕГДА создавайте резервные копии!

Проще всего запускать по расписанию (и, если надо, вне расписания вручную) файлы наподобие таких (сейчас рассмотрим).

Внимательно изучите, как работают утилиты создания дампов баз данных для вашей СУБД. Начать можно с mysqldump для MySQL.

Под Windows (CMD-скрипт):

```
set DBUSER=root
set DBPASSWORD=123456
set DBNAME=site
set BACKUPDESTINATION=z:/Backup/
set WEBROOTFOLDER=c:/www_pub/

set VDATE=%DATE:/=_%
set VDATE=%VDATE: .=_%
set VTIME=%TIME:~0,8%
set VTIME=%VTIME: :=_%
set VTIME=%VTIME: =_%
set DT=%VDATE%%VTIME%

set SQLDUMPNAME=daily_db_%DT%.sql
set SQLDUMPNAMEZIP=daily_db_%DT%.zip
set FILESDUMPZIP=daily_files_%DT%.zip

mysqldump -u%DBUSER% -p%DBPASSWORD% %DBNAME% > %SQLDUMPNAME%
7z.exe a %BACKUPDESTINATION% %SQLDUMPNAMEZIP% -tzip %SQLDUMPNAME%
del %SQLDUMPNAME%

7z.exe a %BACKUPDESTINATION% %FILESDUMPZIP% -ssw -tzip %WEBROOTFOLDER%
```

Под Linux (bash-скрипт):

```
#!/bin/bash

DBUSER="root"
DBPASSWORD="123456"
DBNAME="site"
BACKUPDESTINATION="/somewhere/somefolder"
WEBROOTFOLDER="/var/www"

SQLDUMPNAME=`date +'daily_db_%Y_%m_%d.sql'``
SQLDUMPNAMEZIP=`date +'daily_db_%Y_%m_%d.tar.gz'``
FILESDUMPZIP=`date +'daily_files_%Y_%m_4%d.tar.gz'``

mysqldump -u$DBUSER -p$DBPASSWORD $DBNAME > $SQLDUMPNAME
tar -czvf $BACKUPDESTINATION/$SQLDUMPNAMEZIP $SQLDUMPNAME
rm $SQLDUMPNAME

tar -czvf $BACKUPDESTINATION/$FILESDUMPZIP $WEBROOTFOLDER
```

## Обновление БД

В зависимости от конкретной ситуации, сложности БД, необходимости сохранить те или иные данные и прочих условий обновление БД может варьироваться от банального удаления, создания и импорта БД до выполнения сложного специально подготовленного SQL-скрипта, выполняющего множество операций.

В любом случае этот процесс также можно и нужно автоматизировать.

## Обновление приложения

Как и в случае с БД приложение можно просто «удалить и скопировать заново», «скопировать поверх» (не рекомендуется), а можно написать скрипт, удаляющий ненужное, копирующий нужное и т.д.

Ранее мы рассматривали разные варианты доступа к вашему серверу (FTP, SCP, SAMBA), и скоро рассмотрим ещё один интересный способ.

## Проверка того, что новая версия работает

Это тоже можно и нужно автоматизировать. К сожалению, наш курс не посвящён вопросам автоматизации тестирования веб-приложений, но одну рекомендацию дать можно: смотрите в направлении Selenium WebDriver и модульных тестов (phpunit):

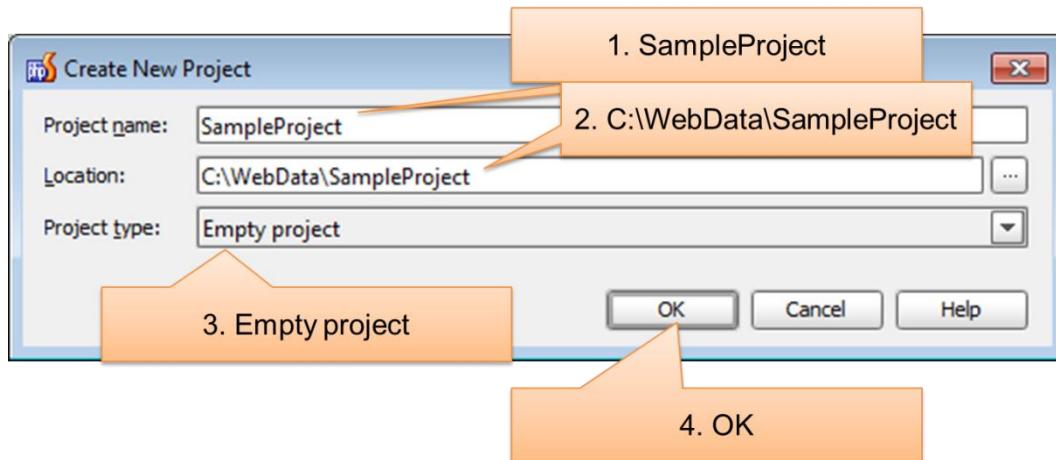
<http://seleniumhq.org>  
<http://phpunit.de>

## Настройка автоматического развертывания в PhpStorm

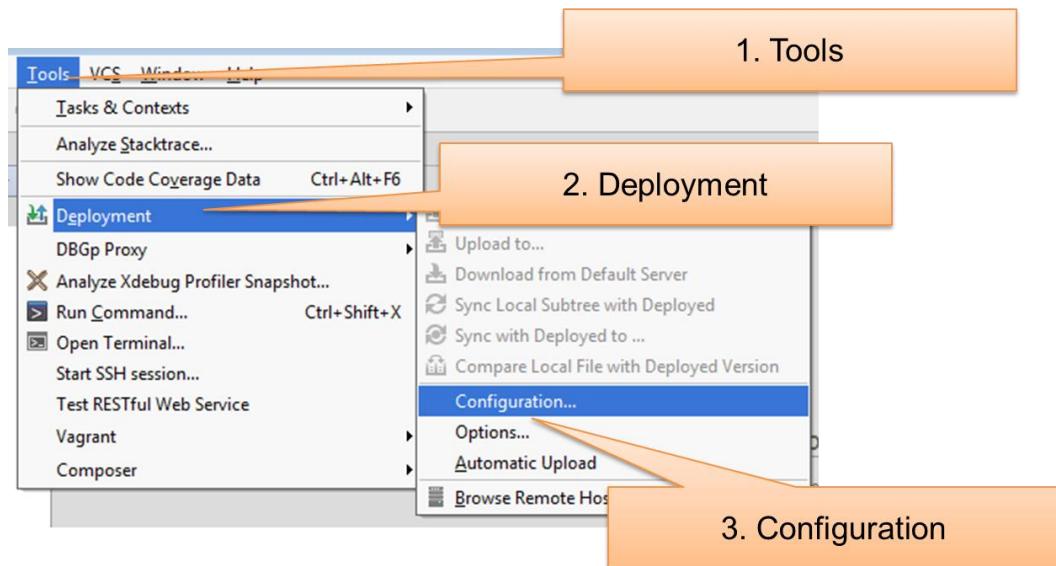
Коль скоро чаще всего разработка ведётся в PhpStorm, сейчас мы посмотрим, как настроить его для автоматического обновления приложения на вашем используемом для разработки сервере.

**Внимание!** НЕЛЬЗЯ настраивать выгрузку сразу на тестовый и тем более на рабочий сервер!

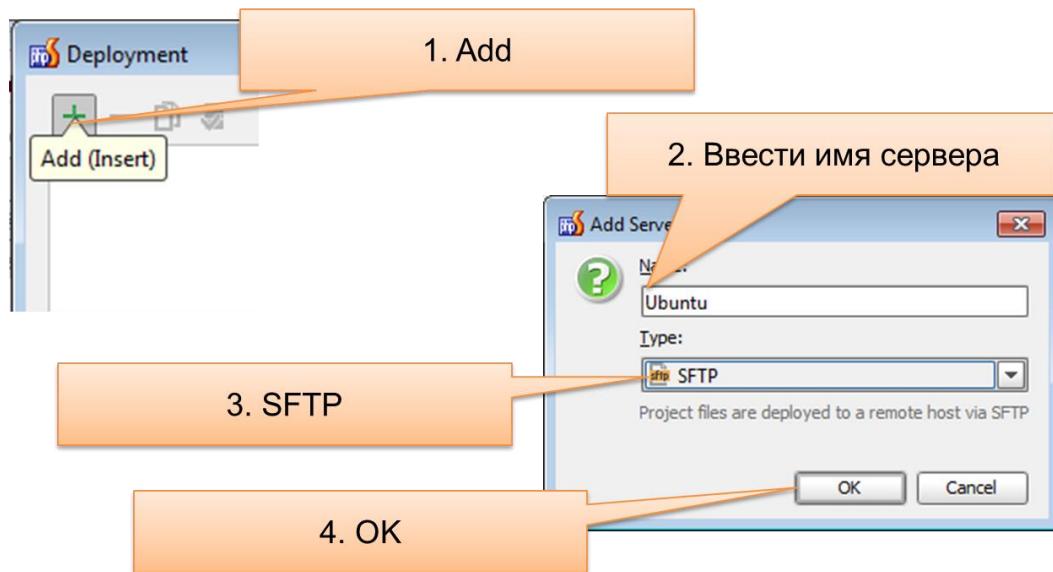
Допустим, у нас есть такое проект:



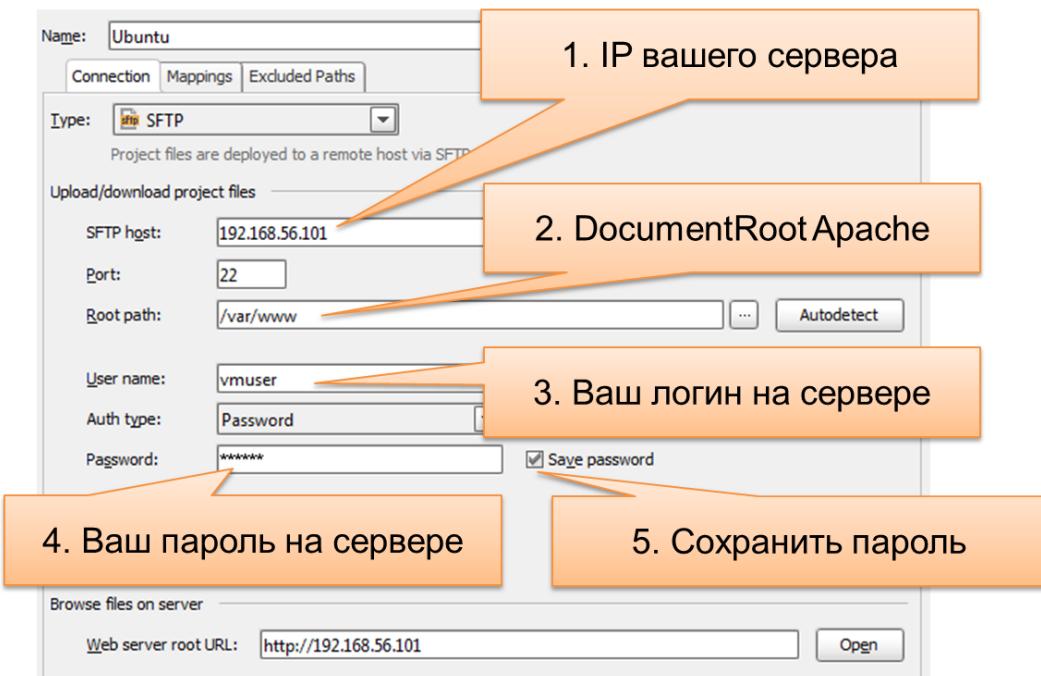
Откройте в меню Tools → Deployment → Configuration:

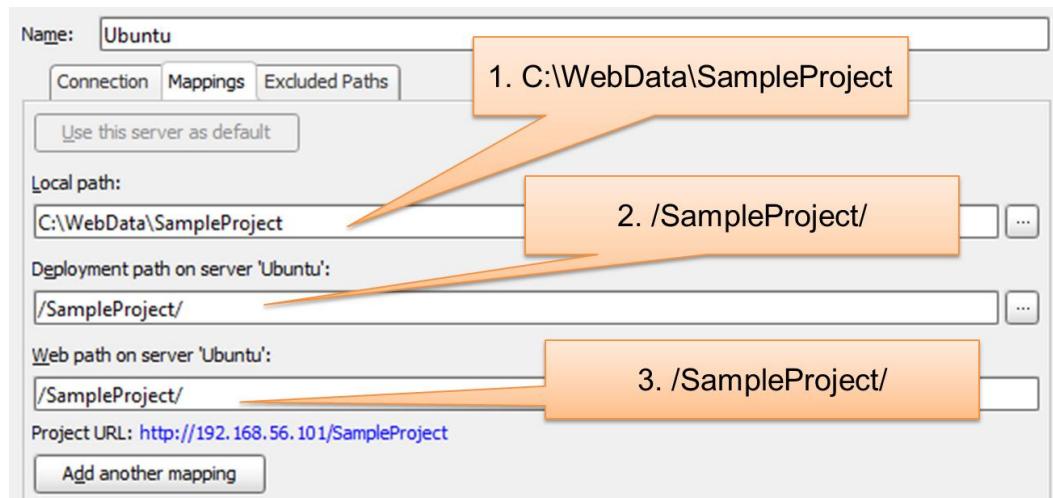


Добавьте новый сервер:

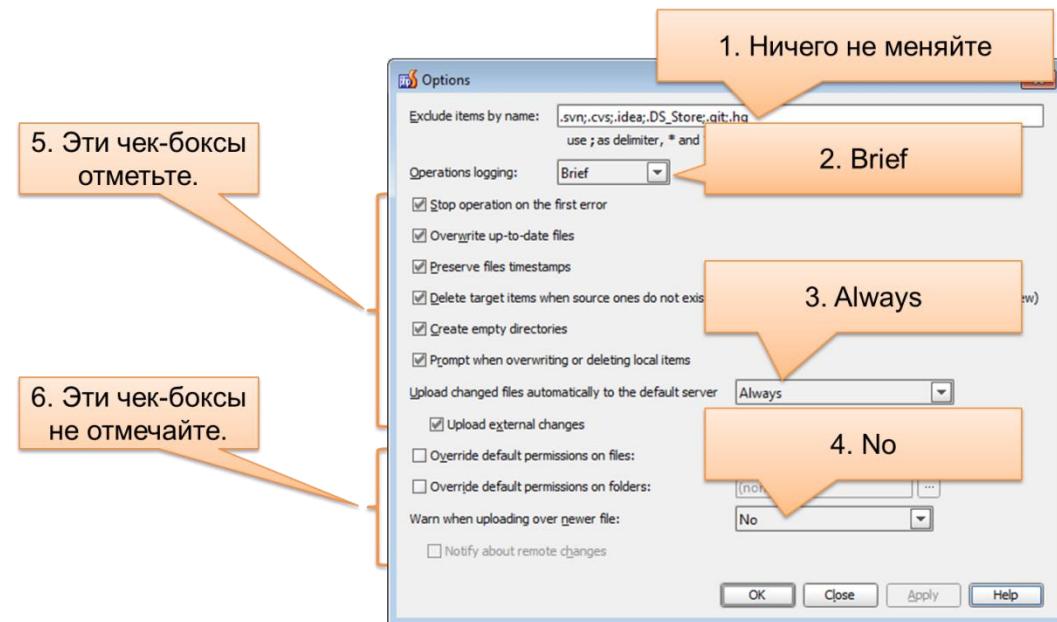


Укажите следующие настройки соединения:

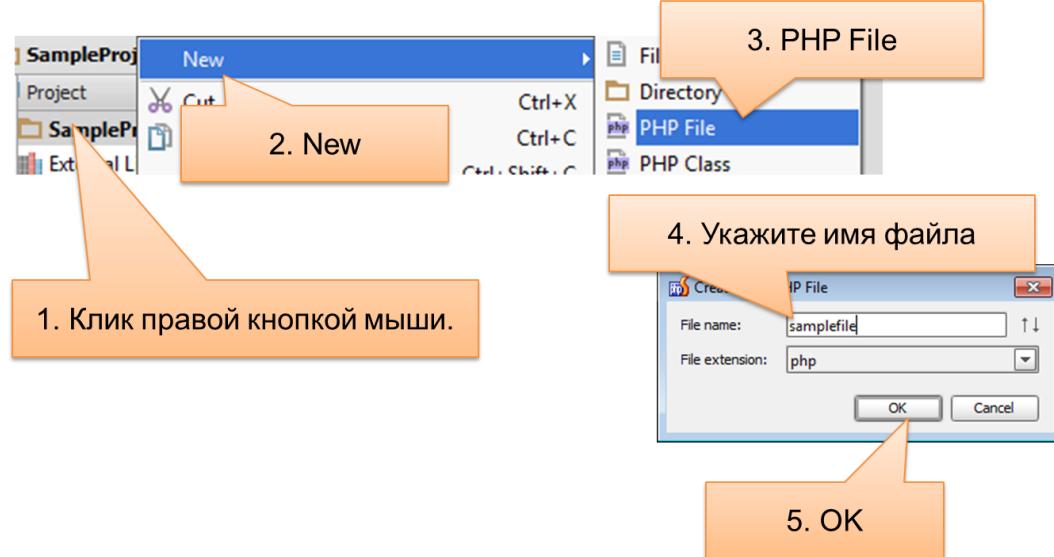




Откройте Tool → Deployment → Options и установите следующие настройки:



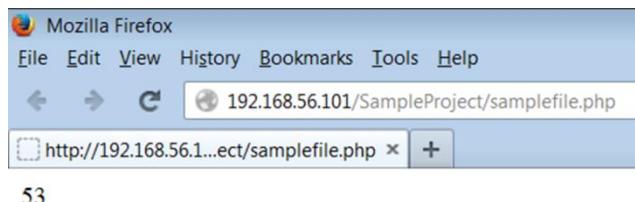
Теперь проверим, как это всё работает. Создадим в PhpStorm новый файл:



Как только мы сохраним наш файл, PhpStorm автоматически обновит его на сервере:

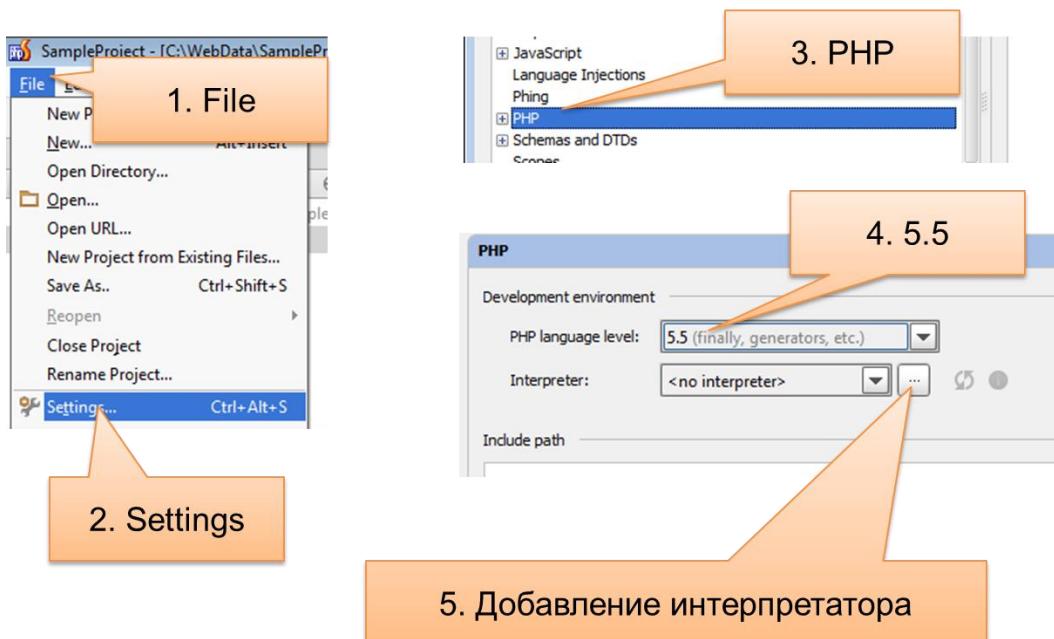
```
[10/23/13 10:58 AM] Automatic upload
[10/23/13 10:58 AM] Automatic upload completed in less than a minute: 1 file
transferred (1.5 Kb/s)
```

Убедимся, что всё работает:

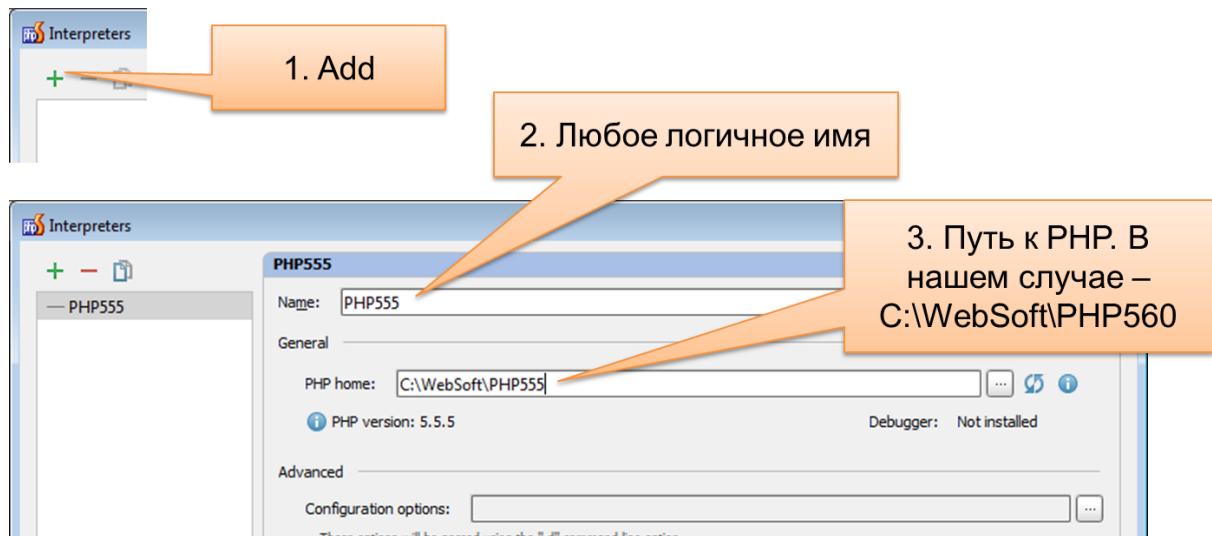


53

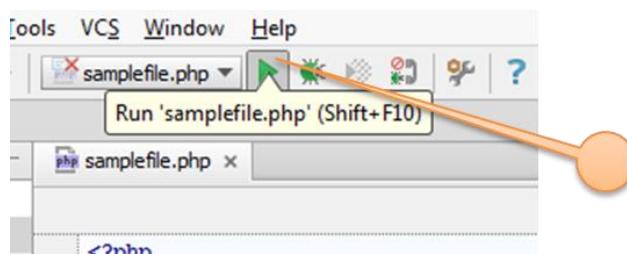
Для локального выполнения выполним такие настройки:



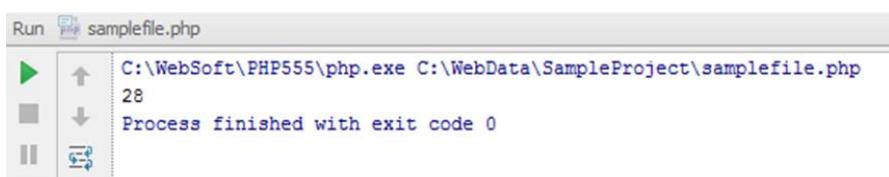
Продолжаем настройки.



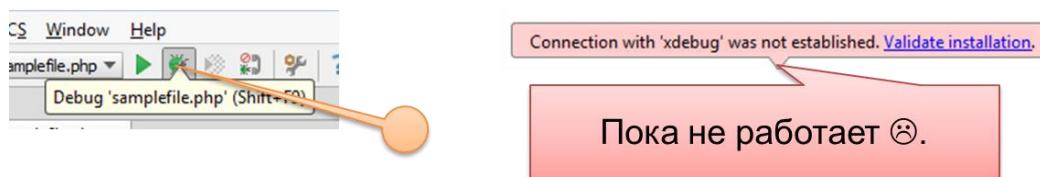
Теперь для выполнения текущего скрипта достаточно нажать Shift+F10 или кликнуть:



Работает:



Для того, чтобы воспользоваться отладчиком, придётся выполнить несколько дополнительных операций:

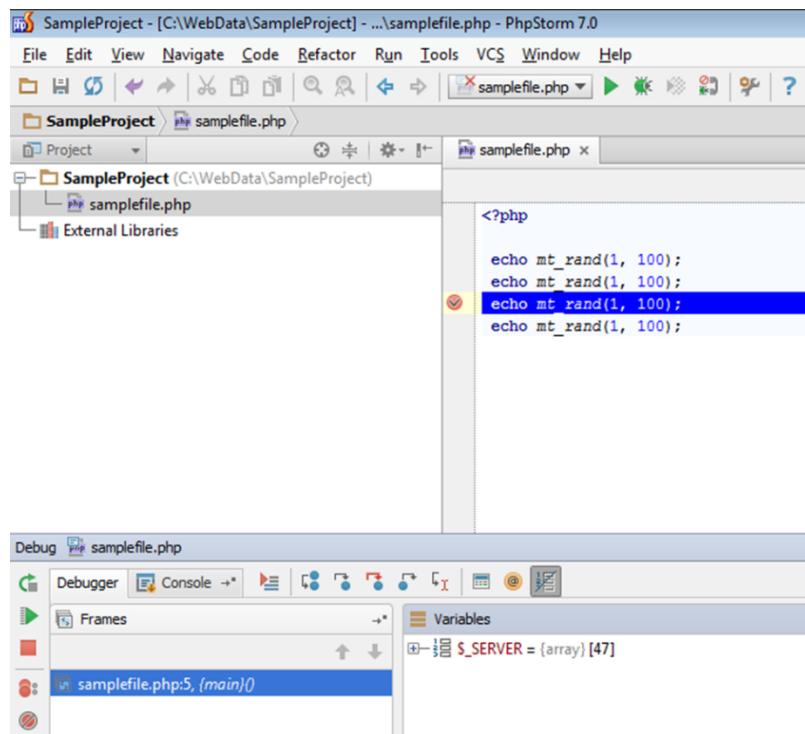


Скачайте xdebug с <http://xdebug.org/download.php>.

Скопируйте полученную DLL в **C:/WebSoft/PHP555/ext** и добавьте в **php.ini** строку:

```
zend_extension="C:/WebSoft/PHP555/ext/php_xdebug-2.2.3-5.5-vc11.dll"
```

Теперь отладка работает:



На этом мы завершаем обзор инфраструктуры для разработки на PHP. Дальше мы будем использовать её в том или ином виде при изучении самого PHP и решения с его помощью различных задач.

Главное напутствие: не пытайтесь относиться к настройке инфраструктуры как чёткому, единожды и на века выверенному алгоритму. Уже через месяц после создания этих слайдов что-то явно поменяется и будет работать иначе – просто думайте, правильно гуглите и экспериментируйте.

## 2.5.2. Типичные вопросы и ответы по настройке

Часто даже в случае, когда «всё понятно», остаются вопросы:

- А куда девать веб-приложение?
- А как настроить виртуальные хосты?
- А где конфигурируется веб-сервер?

Рассмотрим краткие ответы на подобные вопросы.

**Q:** А куда девать веб-приложение?

**A:** В общем случае – в `DocumentRoot` веб-сервера.

Под Windows (для Apache и nginx):

`C:/WebData`

Под Linux (для Apache и nginx):

`/var/www`

**ВНИМАНИЕ!** Эти папки таковы потому, что У НАС (см. ранее про настройку всей системы) так настроено. Они могут отличаться в других системах!

**Q:** А где это («папка для веб-приложения») настраивается?

**A:** В конфигурации веб-серверов.

Под Windows

Для Apache: параметр `DocumentRoot` в файле

`C:/WebSoft/Apache2410/conf/httpd.conf`

Для nginx: параметр `root` секции `location` в файле

`C:/WebSoft/nginx161/conf/nginx.conf`

Под Linux

Для Apache: параметр `DocumentRoot` в файле

`/etc/apache2/sites-available/default`

Для nginx: параметр `root` секции `location` (или `server`) в файле

`/etc/nginx/sites-available/default`

**Q:** Я положил своё веб-приложение (например, wordpress) в папку `C:/WebData/wordpress/`, набираю в браузере `http://127.0.0.1/WebData/wordpress/` и получаю ошибку «404 not found». Что не так?

**A:** Путь. После доменного имени надо писать «хвост», остающийся после пути к `DocumentRoot`, т.е. в вашем случае: `http://127.0.0.1/wordpress/` (без Web-Data).

**Q:** Я запустил виртуальную машину, всё настроил, теперь в браузере на хостовой ОС набираю `http://127.0.0.1` – и ничего не работает. Почему?

**A:** Потому, что `127.0.0.1` – ip вашей хостовой системы (когда вы работаете под ней). Если вы выполните этот же запрос из-под гостевой системы – всё заработает. Внимательно следите за тем, под какой системой вы выполняете действия. Для простоты представляйте себе, что это физически различные компьютеры.

**Q:** Как настроить виртуальные хосты?

**A:** OK, давайте настроим восемь виртуальных хостов из-под хостовой системы на Windows и Linux, Apache и nginx для `DocumentRoot` и для `phpmyadmin`. Тут будет много, но – приступим.

**Шаг 1:** настройка на хостовой ОС.

Допустим, что ip ваших виртуальных машин таковы:

- Windows: 192.168.56.102
- Linux: 192.168.56.101

Откройте файл

`C:/Windows/System32/drivers/etc/hosts`

и внесите туда следующее:

```
192.168.56.101      linaproot
192.168.56.101      linappma
192.168.56.101      linngroot
192.168.56.101      linngpma
192.168.56.102      winaproot
192.168.56.102      winappma
192.168.56.102      winngroot
192.168.56.102      winngpma
```

**Шаг 2:** настройка виртуальной машины под Windows.

Откройте файл

C:/WebSoft/Apache2410/conf/httpd.conf

и раскомментируйте строку:

#Include conf/extra/httpd-vhosts.conf

Откройте файл

C:/WebSoft/Apache2410/conf/extra/httpd-vhosts.conf

и разместите там такое содержимое:

```
<VirtualHost *:80>
    ServerAdmin mail@winaproot
    DocumentRoot "C:/WebData"
    ServerName winaproot
    ServerAlias winaproot
    ErrorLog "logs/winaproot-error.log"
    CustomLog "logs/winaproot-access.log" common
</VirtualHost>
<VirtualHost *:80>
    ServerAdmin mail@winappma
    DocumentRoot "C:/WebData/phpmyadmin"
    ServerName winappma
    ServerAlias winappma
    ErrorLog "logs/winappma-error.log"
    CustomLog "logs/winappma-access.log" common
</VirtualHost>
```

Перезапустите Apache.

Откройте файл

C:/WebSoft/nginx161/conf/nginx.conf

и разместите там такое содержимое (добавьте):

```
server {
    listen      81;
    server_name winngroot;

    location / {
        root   c:/webdata;
        index index.php index.html index.htm;
    }

    error_page  500 502 503 504  /50x.html;
    location = /50x.html {
        root   html;
    }

    location ~ \.php$ {
        root           c:/webdata;
        fastcgi_pass  127.0.0.1:9000;
        fastcgi_index index.php;
        fastcgi_param SCRIPT_FILENAME c:/webdata/$fastcgi_script_name;
        include        fastcgi_params;
    }
}

server {
    listen      81;
    server_name winngpma;

    location / {
        root   c:/webdata/phpmyadmin;
        index index.php index.html index.htm;
    }
}
```

```
error_page 500 502 503 504 /50x.html;
location = /50x.html {
    root html;
}

location ~ \.php$ {
    root c:/webdata/phpmyadmin;
    fastcgi_pass 127.0.0.1:9000;
    fastcgi_index index.php;
    fastcgi_param SCRIPT_FILENAME
c:/webdata/phpmyadmin/$fastcgi_script_name;
    include fastcgi_params;
}

}
```

Перезапустите nginx.

### Шаг 3: настройка виртуальной машины под Linux.

Скопируйте файл

/etc/apache2/sites-available/default  
под именами (в той же папке):

linaproto.conf  
linappma.conf

Создайте на только что полученные файлы символические ссылки в папке  
/etc/apache2/sites-enabled

Пример команды по созданию такой ссылки:

```
sudo ln -s /etc/apache2/sites-available/linappma.conf
/etc/apache2/sites-enabled/linappma.conf
```

В файле

/etc/apache2/sites-available/linaproto.conf  
замените все вхождения localhost на linaproto .

После строки

```
ServerAdmin webmaster@linaproto
добавьте строки
ServerName linaproto
ServerAlias linaproto
```

В файле

/etc/apache2/sites-available/linappma.conf  
замените все вхождения localhost на linappma И все вхождения /var/www на  
/var/www/phpmyadmin .

После строки

```
ServerAdmin webmaster@linappma
добавьте строки
ServerName linappma
ServerAlias linappma
```

Перезапустите Apache.

---

---

---

---

Скопируйте файл  
`/etc/nginx/sites-available/default`  
под именами (в той же папке):  
`linngroot`  
`linngpma`

Создайте на только что полученные файлы символические ссылки в папке  
`/etc/nginx/sites-enabled`

Пример команды по созданию такой ссылки:  
`sudo ln -s /etc/nginx/sites-available/linngpma /etc/nginx/sites-enabled/linngpma`

В файл  
`/etc/nginx/sites-available/linngroot`  
поместите следующее (замените имеющееся):

```
server {  
    listen 81; ## listen for ipv4; this line is default and implied  
  
    root /var/www;  
    index index.php index.html index.htm;  
  
    server_name linngroot;  
  
    location / {  
        try_files $uri $uri/ /index.html;  
    }  
  
    location /doc/ {  
        alias /usr/share/doc/;  
        autoindex on;  
        allow 127.0.0.1;  
        deny all;  
    }  
  
    location ~ \.php$ {  
        fastcgi_split_path_info ^(.+\.php)(/.+)$;  
        fastcgi_pass unix:/var/run/php5-fpm.sock;  
        fastcgi_index index.php;  
        include fastcgi_params;  
    }  
}
```

В файл  
`/etc/nginx/sites-available/linngpma`  
поместите следующее (замените имеющееся):

```
server {  
    listen 81; ## listen for ipv4; this line is default and implied  
  
    root /var/www/phpmyadmin;  
    index index.php index.html index.htm;  
  
    server_name linngpma;  
  
    location / {  
        try_files $uri $uri/ /index.html;  
    }  
  
    location /doc/ {  
        alias /usr/share/doc/;  
        autoindex on;  
        allow 127.0.0.1;
```

```
        deny all;
    }

location ~ \.php$ {
    fastcgi_split_path_info ^(.+\.php)(/.+)$;
    fastcgi_pass unix:/var/run/php5-fpm.sock;
    fastcgi_index index.php;
    include fastcgi_params;
}

}
```

Перезапустите nginx.

#### **Шаг 4: Проверка результата.**

На хостовой системе откройте браузер и в нём откройте следующие адреса:

```
http://winaproot/  
http://winappma/  
http://winngroot:81/  
http://winngpma:81/
```

```
http://linaproot/  
http://linappma/  
http://linngroot:81/  
http://linngpma:81/
```

Если вы всё сделали правильно, вы увидите содержимое DocumentRoot (или «403 forbidden» для nginx ☺) и главную страницу phpMyAdmin на двух виртуальных машинах под двумя веб-серверами.

## Тема 3. Основы PHP

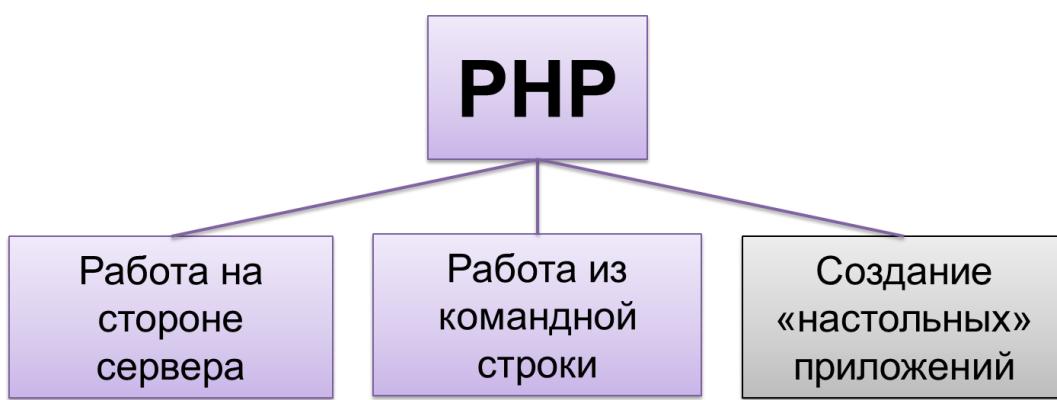
### **3.1. Общие сведения о PHP**

PHP (PHP – Hypertext Preprocessor) – язык программирования, предназначенный для разработки серверной части веб-приложений и решения иных задач широкого спектра.

## Основные факты:

- Программы на PHP хранятся в виде исходного текста.
  - Большая часть синтаксиса PHP пришла по наследию из языка С.
  - Основное преимущество PHP – в его простоте и широких возможностях по решению «типичных повседневных задач».

Программы на PHP могут выполняться тремя способами, причём создание настольных приложений является скорее экзотикой.



**Задание для самостоятельной проработки:** прочитайте описание PHP в Википедии и на официальном сайте PHP в разделе документации. Если возникнут вопросы – запишите их.

### 3.2. Что нужно для работы PHP

Для работы PHP понадобится:

- Сам PHP ☺: <http://php.net/downloads.php>
- Веб-сервер Apache: [http://projects.apache.org/projects/http\\_server.html](http://projects.apache.org/projects/http_server.html) или <http://www.apachelounge.com/download/>
- Текстовый редактор (например, Notepad++: <http://notepad-plus-plus.org/download/>) или среда разработки (например, “JetBrains PhpStorm”: <http://www.jetbrains.com/phpstorm/download/>)
- Для работы с БД проще всего использовать СУБД MySQL: <http://dev.mysql.com/downloads/>

Видеопособие по установке и настройке:

[http://svyatoslav.biz/education/work\\_environment\\_setup/](http://svyatoslav.biz/education/work_environment_setup/)

Напомним: в разделе «[Инфраструктура в разработке на PHP](#)» вопросы установки и настройки всего необходимого для работы с PHP рассмотрены намного более подробно.

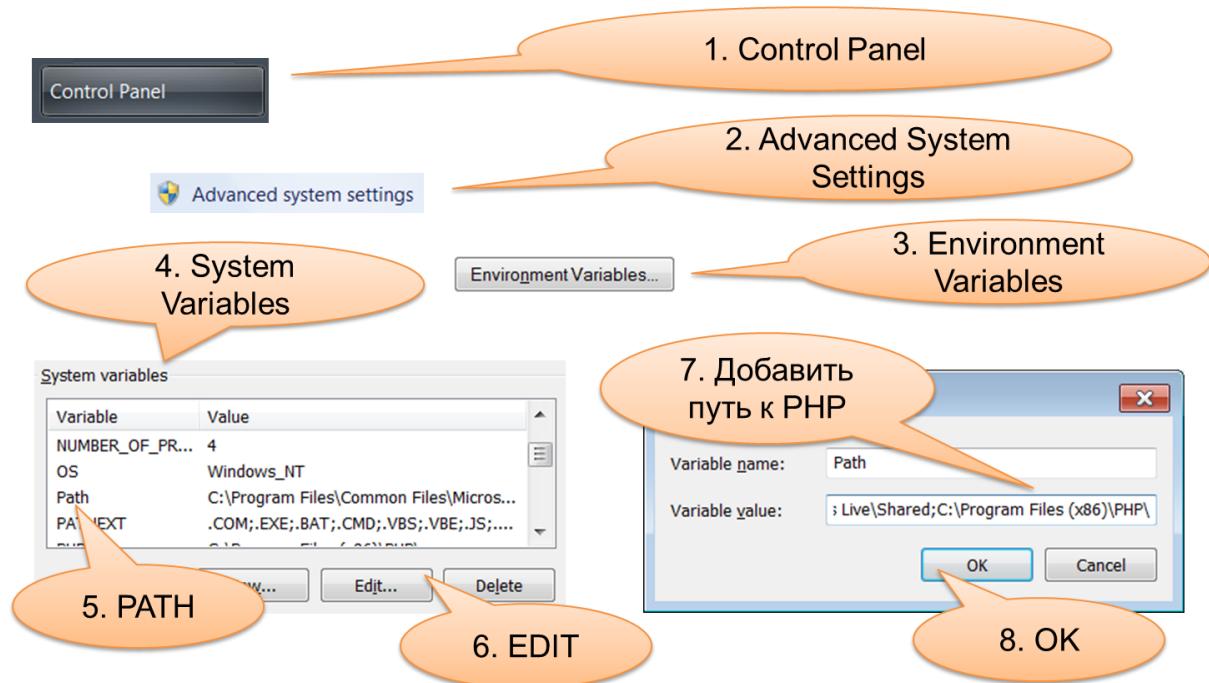
#### Упрощённый вариант работы - 1

Чтобы просто писать и запускать программы на PHP достаточно сделать следующее:

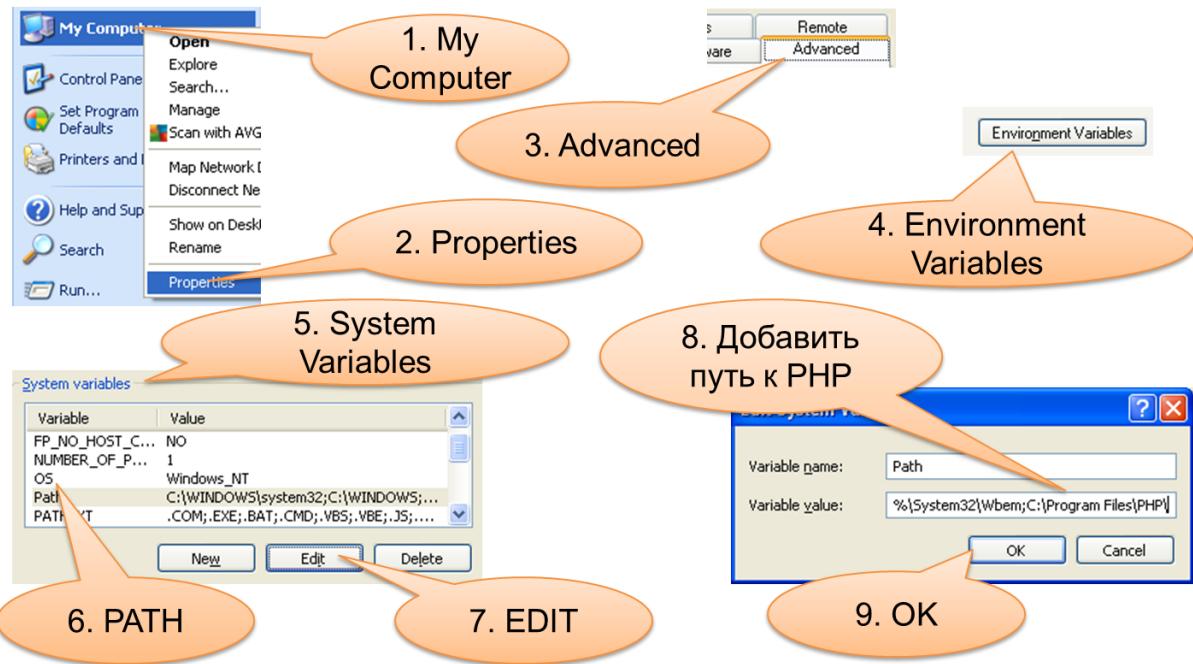
- Убедиться, что путь к CLI-интерпретатору прописан в PATH.
- Из командной строки (лучше всего использовать файловый менеджер FAR) выполнить команду

```
php имявашегоскрипта
```

Как добавить путь к PHP в PATH под Win7:



## Как добавить путь к PHP в PATH под WinXP:



Проверка работоспособности. В FAR для показа консоли используйте комбинацию клавиш Ctrl-O.

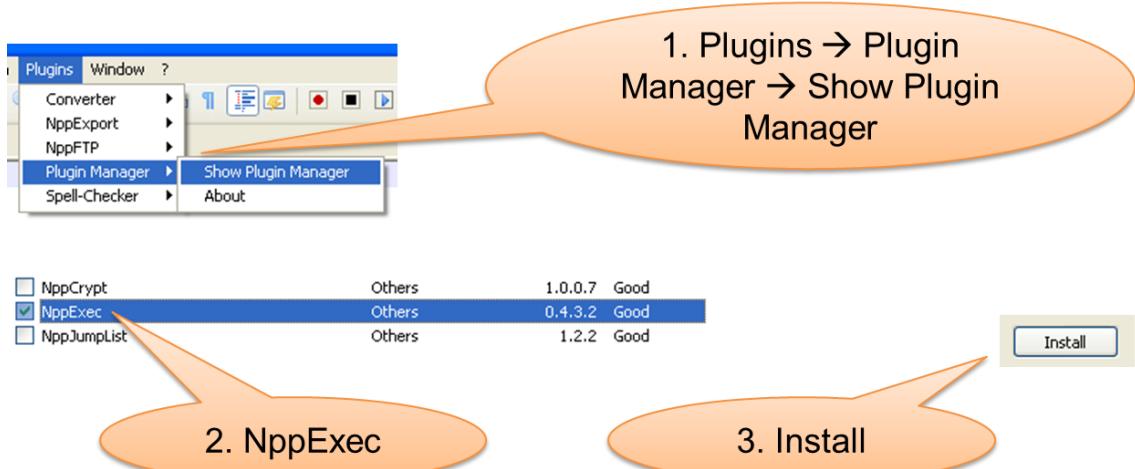
```
D:\_www>php helloworld.php
```

```
D:\_www>php helloworld.php
Hello, world!
```

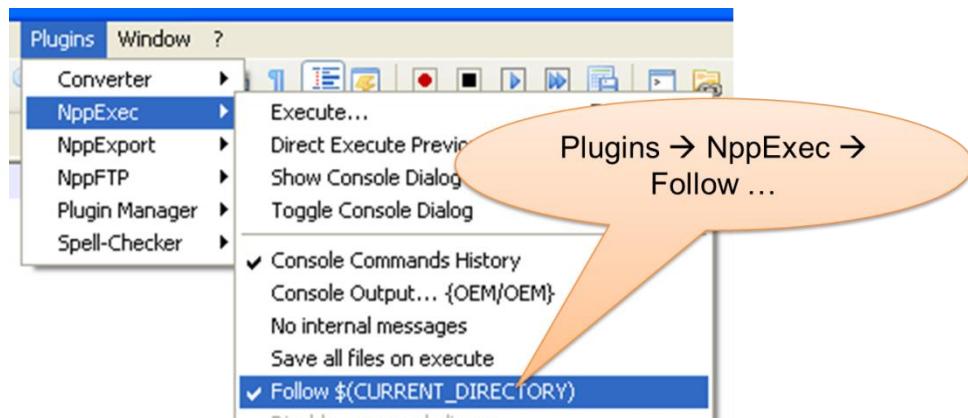
## Упрощённый вариант работы - 2

Поскольку мы будем использовать для работы Notepad++, имеет смысл настроить его для быстрого выполнения PHP-скриптов.

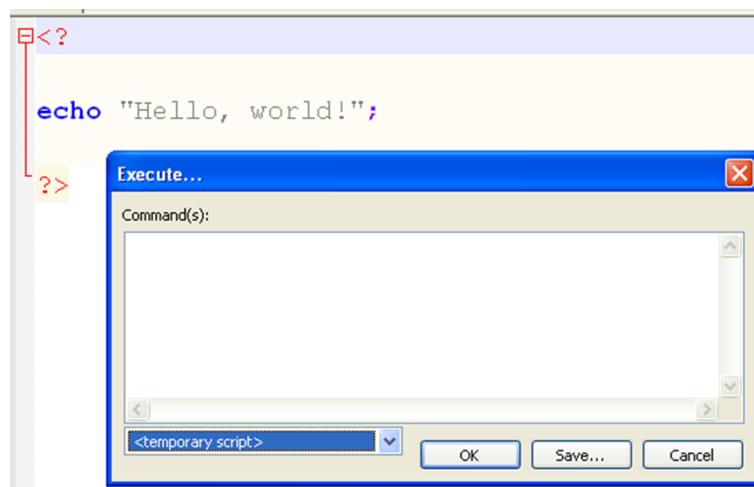
Для начала установим нужный плагин:



Теперь настроим плагин. Сначала обеспечим переход в текущий каталог перед запуском скрипта:



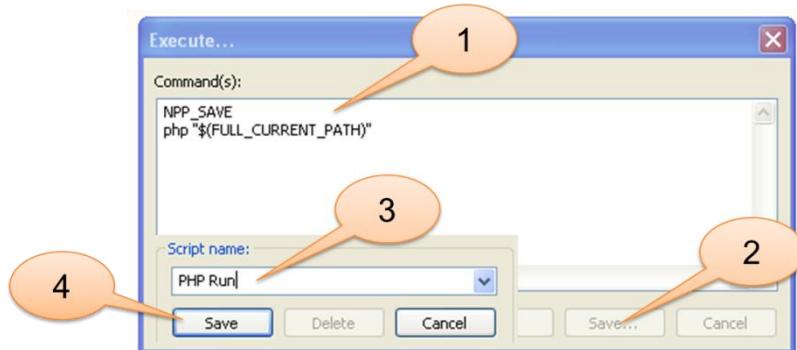
Откроем любой PHP-файл и нажмём F6 (вызов окна запуска):



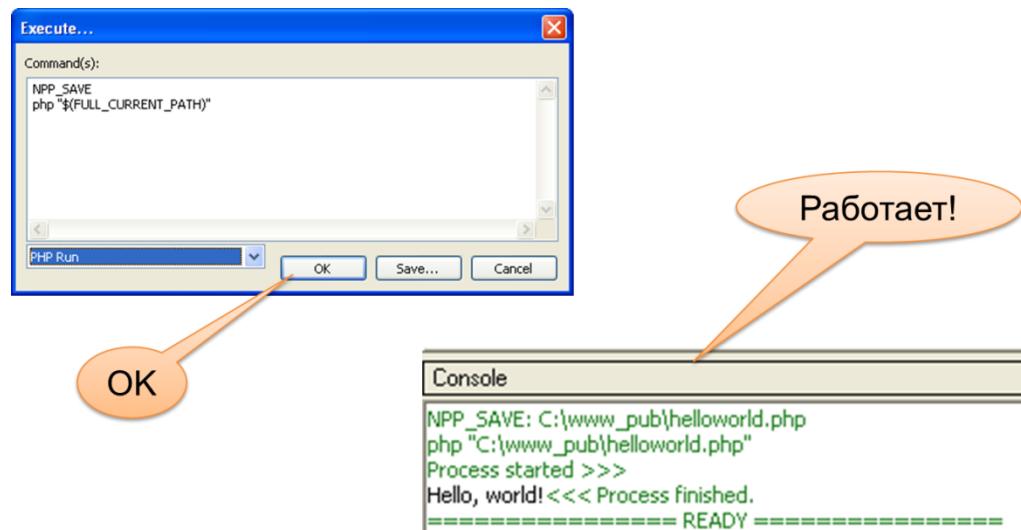
Впишем сюда следующие команды

```
NPP_SAVE
php "$(FULL_CURRENT_PATH)"
```

и сохраним скрипт.

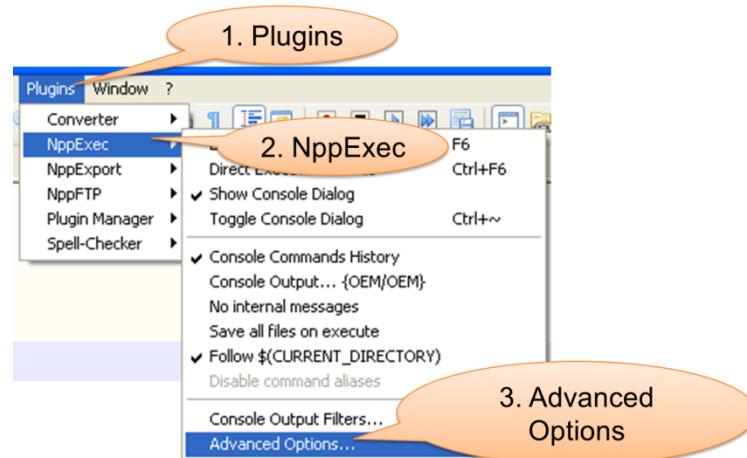


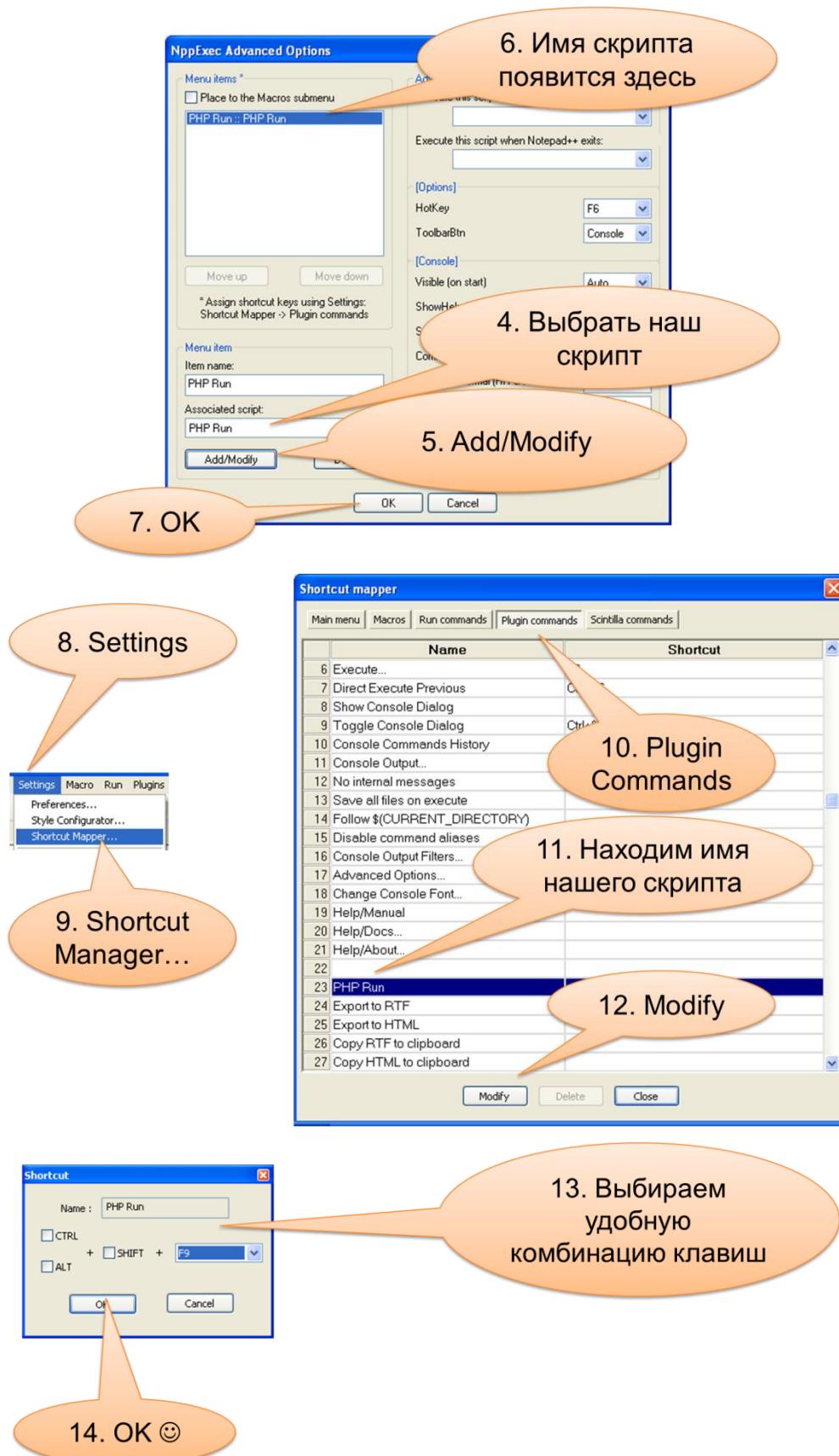
После нажатия «OK» вы уже увидите результат:



Чтобы запуск можно было осуществлять нажатием одной клавиши, сделаем ещё несколько настроек. Если по картинкам не до конца понятно, как это работает, рекомендую прочитать описание:

<http://stackoverflow.com/questions/9001046/how-can-i-integrate-php-python-interpreter-to-notepad>





Всё! Теперь для запуска на выполнение редактируемого скрипта достаточно нажать F9.

## Запуск в браузере

Если вы настроили PHP для работы в браузере (напоминаю, видеопособие – здесь: [http://svyatoslav.biz/education/work\\_environment\\_setup/](http://svyatoslav.biz/education/work_environment_setup/) ), то вам достаточно поместить свой скрипт в DocumentRoot веб-сервера, открыть в браузере <http://127.0.0.1> и кликнуть по имени скрипта:



Антрапедагогичное отступление ☺. Кто-то может сказать, что это «ни капельки не простые способы». Это – простые. Если не верите, поработайте с PhpStorm или Zend Studio – это отличные среды разработки, но их изучение требует больше времени.

В любом случае: если есть вопросы – запишите!

### 3.3. Общий синтаксис PHP

#### Команды PHP

Команды PHP заключаются в специальные теги, которые бывают четырёх видов:

- <?php ?>
- <? ?> (в некоторых версиях отключён по умолчанию в настройках)
- <script language = "php"></script>
- <% %> (отключён по умолчанию в настройках)

Мы будем использовать наиболее классический вариант:

```
<?php  
    ...  
?>
```

#### Комментарии в PHP

PHP предоставляет несколько способов вставки комментариев.

```
<?php  
    echo 'Hello, ' // комментарий  
    echo 'world!'; # комментарий  
/*  
    и это тоже комментарий  
*/  
?>
```

#### Регистрочувствительность

Большая часть синтаксиса PHP чувствительна к регистру. Да, есть исключения, но для простоты будем считать, что регистр надо учитывать ВЕЗДЕ:

```
<?php  
$var1 = 1;  
$vAr1 = 1;  
?>
```

Это – РАЗНЫЕ переменные!  
Пусть отличие и заключается  
всего лишь в регистре одной  
буквы.

### 3.4. Переменные и типы данных PHP

#### Переменные в PHP

PHP – нестроготипизированный язык: переменные в нём могут менять свой тип в процессе выполнения программы.

PHP не накладывает строгих ограничений на участие в выражениях переменных разных типов.

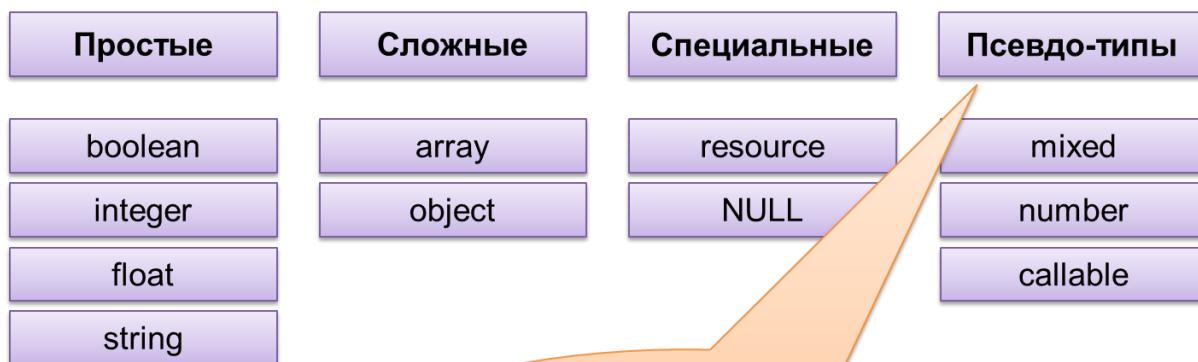
Переменные не нужно объявлять отдельно, они «объявляются» через инициализацию при первом использовании.

Имя переменной должно начинаться со знака \$, после которого идёт буква. Далее можно использовать буквы, цифры, знаки подчёркивания.

```
<?php  
$a = 1;  
$some_var = 1;  
$y2k_compatibility = TRUE;  
?>
```

#### Типы данных в PHP

PHP поддерживает следующие типы данных:



Эти типы данных  
используются ТОЛЬКО в  
документации. В самом  
языке их НЕТ!

## Простые типы данных: Boolean

**boolean** используется для хранения только двух логических значений: «истина» или «ложь».

```
<?php
$bool_var_1 = TRUE;
$bool_var_2 = true;
$bool_var_3 = FALSE;
$bool_var_4 = false;
?>
```

## Простые типы данных: integer

**integer** используется для хранения целочисленных значений.

```
<?php
$int_var_1 = 10;
$int_var_2 = -12;
?>
```

Переменные типа **integer** можно объявлять в четырёх системах счисления, но при первой же операции с этой переменной они будут приведены к 10-чной.

```
<?php
$int_var_1 = 10; // 10-чная СС
$int_var_2 = 0xFF; // 16-ричная СС
$int_var_3 = 0777; // 8-ричная СС
$int_var_4 = 0b10; // 2-ичная СС
?>
```

## Простые типы данных: float

**float (double)** используется для хранения дробных числовых значений.

```
<?php
$float_var_1 = 1.57;
$float_var_2 = -12.99;
$float_var_3 = 1.5e-1; // 0.15
?>
```

## Простые типы данных: string

**string** используется для хранения строк.

```
<?php
$string_var_1 = "Test";
$string_var_2 = 'Test';
?>
```

Строки в PHP могут заключаться в двойные (") и одинарные ('') кавычки.

В случае одинарных кавычек ('') строка рассматривается просто как набор символов, который никак дополнительно не обрабатывается.

Если внутрь строки надо поместить саму одинарную кавычку ('), она экранируется бэк-слешем (\).

```
<?php
$string_var_1 = 'Test';
$string_var_2 = 'Hotel \'Minsk\'';
?>
```

В случае двойных кавычек ("") строка анализируется.

В такой строке можно применять метасимволы и использовать имена переменных (об этом будет сказано отдельно в разделе про операторы PHP).

```
<?php
$string_var_1 = "Test";
$string_var_2 = "Hotel \"Minsk\"";
$string_var_3 = "His age is $age";
$string_var_4 = "The value is ${values['val']}";
$string_var_5 = "Two\nlines";
?>
```

Раз уж речь зашла о метасимволах, то вот краткий перечень:

- \n – новая строка
- \r – возврат каретки (в некоторых случаях переход на новую строку выполняется комбинацией \r\n)
- \t – табуляция

К отдельному символу строки в однобайтовой кодировке можно обратиться так:

```
<?php
$str = 'Just a string';
$chr1 = $str{3}; // t (нумерация идёт с 0)
$chr2 = $str[3]; // нерекомендуемый способ
?>
```

## Сложные типы данных: array

**array** используется для хранения наборов данных. В PHP массивы обладают следующими свойствами:

- Бывают только динамическими.
- Могут содержать одновременно данные любых типов.
- В качестве ключей (индексов) могут использовать как числа, так и строки.

```
<?php
$arr = array('Apple', 999, TRUE);
?>
```

Объявление массивов может выполняться с помощью ключевого слова **array** или с помощью синтаксиса квадратных скобок [ ].

```
<?php
$arr1 = array('Apple', 999, TRUE);
$arr2 = array('name' => 'Jonh', 'age' => 40);
$arr3['name'] = 'John';
$arr3['age'] = 40;
$arr4 = ['A', 'B', 'C']; // Начиная с PHP 5.4
$arr5 = [99 => 'A', 'B', 'C']; // Начиная с PHP 5.4
?>
```

Многомерные массивы объявляются аналогично:

```
<?php
$arr1 = array(
    'phone1' => array('code' => '123', 'number' => '1112233'),
    'phone2' => array('code' => '456', 'number' => '7778899')
);

$arr2['phone1']['code'] = '123';
$arr2['phone1']['number'] = '1112233';
$arr2['phone2']['code'] = '456';
$arr2['phone1']['number'] = '7778899';
?>
```

При объявлении массива можно не указывать ключ (индекс). Тогда PHP инкрементирует на 1 последний «самый большой» индекс и использует его.

```
<?php
$arr = array(50 => 'ABC', 'DEF');
?>
```

У этого элемента индекс = 51.

## Объявление многомерных массивов с пропуском ключей

В случае, если необходимо объявить многомерный массив с помощью [ ] и с пропуском ключей, следует помнить, что PHP будет инкрементировать ключи по всем уровням, в которых они пропущены.

```
<?php
$arr[10][15] = 'A';
$arr[10][] = 'B'; // $arr[10][16] = 'B';
$arr[][] = 'C'; // $arr[11][0] = 'C';
?>
```

## Обращение к элементам массива, возвращённого функцией

До PHP 5.4 возможность прямого обращения к элементу массива, возвращённого функцией, отсутствовала (необходимо было использовать промежуточную переменную). В PHP 5.4 такая возможность появилась.

```
<?php
// До PHP 5.4
$tmp = fnc();
$x = $tmp[$y];

// Начиная с PHP 5.4
$x = fnc()[$y];
?>
```

## Сложные типы данных: `object`

`object` представляет собой экземпляр класса. Подробнее об этом будет в теме про объектно-ориентированное программирование (ООП), а пока – просто пример того, «как получить `object`»:

```
<?php
class Test
{
    public $some_var;
}

$t = new Test(); // $t - object
?>
```

## Специальные типы данных: `resource`

`resource` используется для хранения информации о некотором ресурсе (дескрипторе файла, соединении с БД, создаваемом изображении и т.д.) В общем случае с ресурсами можно сделать только три вещи:

- получить;
- использовать;
- освободить.

```
<?php
$fd = fopen('1.txt', 'wb+'); // получение
fwrite($fd, 'OK!'); // использование
fclose($fd); // освобождение
?>
```

## Специальные типы данных: `NULL`

`NULL` означает «отсутствие значения». Переменная равна `NULL` в следующих трёх случаях:

- ей присвоено значение `NULL`;
- она ещё не была проинициализирована;
- она была «удалена» функцией `unset()`.

```
<?php
$var1 = NULL;
unset($var_x);
?>
```

## Псевдо-типы: mixed, number, callable (callback)

Псевдо-типы mixed, number и callback используются только в документации и обозначают:

- mixed – используется, чтобы не перечислять имена нескольких разных типов;
- number – используется, чтобы не перечислять все числовые типы;
- callable (callback) – показывает, что функция, имя которой будет приведено в указанном месте, будет вызвана автоматически.

```
mixed print_r ( mixed $expression [, bool $return = false ] )  
number abs ( mixed $number )  
bool usort ( array &$array , callable $cmp_function )
```

## Константы в PHP

Помимо переменных PHP также поддерживает константы. Их значение изменять нельзя. Имена констант НЕ начинаются со знака \$.

```
<?php  
  
define('myconst', 'ABC'); // установка значения константы  
  
if (defined('myconst')) // проверка существования константы  
{  
    echo myconst; // использование константы  
}  
?>
```

### 3.5. Определение и преобразование типов данных

#### Определение типа переменной

Для определения типа переменной используется функция

```
string gettype ( mixed $var )
```

которая возвращает тип переменной в виде строки:

- boolean;
- integer;
- double (НЕ float);
- string;
- array;
- object;
- resource;
- NULL;
- unknown type.

Рассмотрим пример:

```
<?php
$a = 10;
$b = 11.1;
$c = TRUE;
$d = 'ABC';

echo gettype($a); // integer
echo gettype($b); // double
echo gettype($c); // boolean
echo gettype($d); // string
?>
```

Для проверки принадлежности переменной к некоторому типу данных существует следующий набор функций:

```
bool is_bool(mixed $var)
bool is_numeric(mixed $var)
bool is_float(mixed $var)
bool is_int(mixed $var)
bool is_string(mixed $var)
bool is_object(mixed $var)
bool is_array(mixed $var)
bool is_null(mixed $var)
```

Реагирует в т.ч. на числа, представленные строками.

Да, этих функций больше. Но мы пока рассматриваем только самые основные.

## Преобразование типа переменной

Для преобразования типа переменной используется функция

```
bool settype ( mixed &$var , string $type )
```

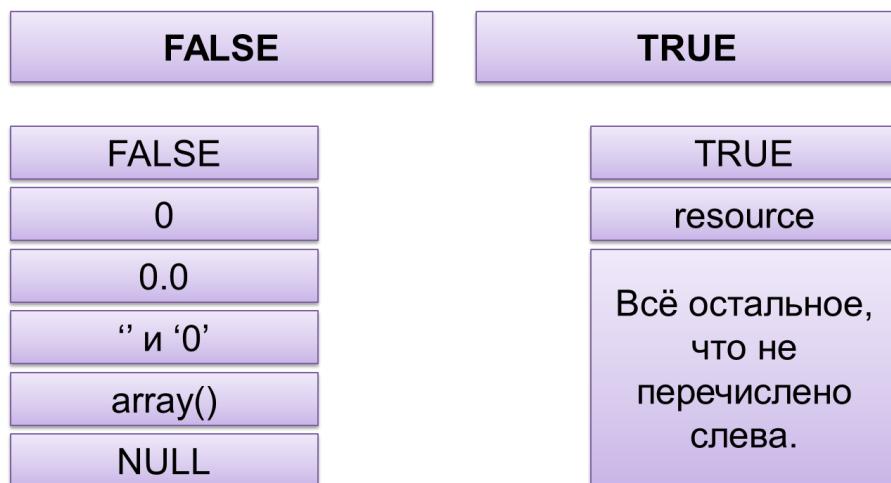
или «С-образный синтаксис»

```
$var = (newtype) $var;
```

В обоих случаях в качестве имени типа используются те же строки, которые возвращает gettype().

**Внимание! В большинстве случаев преобразование типа МЕНЯЕТ значение переменной!**

### Преобразование к Boolean



### Преобразование к integer



**Преобразование из других типов НЕ ОПРЕДЕЛЕНО.**

## Преобразование к integer (и float)

### Из string

Учитываются символы в начале строки, соответствующие синтаксису записи чисел.

```
1 + "10.5";           // float (11.5)
1 + "-1.3e3";         // float (-1299)
1 + "bob-1.3e3";      // integer (1)
1 + "bob3";           // integer (1)
1 + "10 cats";        // integer (11)
4 + "10.2 dogs";      // float (14.2)
"10.0 birds " + 1;    // float (11)
"    10.0 birds " + 1.0; // float (11)
```

## Преобразование к float

### Из boolean

FALSE → 0.0  
TRUE → 1.0

### Из float

Значение не меняется ☺.

### Из string

См. ранее ☺.

### Из NULL

NULL → 0.0

**Преобразование из других типов НЕ ОПРЕДЕЛЕНО.**

## Преобразование к string

### Из boolean

FALSE → ""  
TRUE → '1'

### Из array, resource, object

array → 'Array'  
resource → 'Resource id # N'  
object → 'Object'

### Из NULL

NULL → "

Неявное преобразование к string происходит при выводе переменной в выходной поток.

## Преобразование к array

### Из простых типов и resource

Массив с одним элементом – исходным значением.

### Из object

Результат достаточно нетривиален. См. документацию ☺.

### Из NULL

### Пустой массив

## Преобразование к object

### Из любого типа, кроме NULL

Объект с одним свойством **scalar** с исходным значением.

### Из NULL

Объект без свойств.

## Преобразование к resource

Не имеет смысла

Ресурсы содержат такую информацию как дескрипторы открытых файлов, информацию о соединениях с БД и т.п. Эта информация имеет смысл только в привязке к конкретным внешним ресурсам, используемым PHP.

## Преобразование к NULL

Не существует

Фактически, преобразование к NULL – это «удаление» переменной. Такой же эффект даёт `unset($var)`.

## Преобразование и переключение типов

В PHP существует два механизма изменения типов данных:

- преобразование – переменная меняет тип (и, возможно, значение);
- переключение – копия переменной подвергается преобразованию, а сама переменная остаётся незатронутой.

### Преобразование

```
$a = 10;
$a = (string)$a;
```

### Переключение

```
$a = 10;
$b = TRUE;
$c = $a + $b;
```

## Важные функции и их реакция на разные типы и значения данных

| Выражение         | gettype() | empty() | is_null() | isset() | boolean:<br>if(\$x) |
|-------------------|-----------|---------|-----------|---------|---------------------|
| \$x = "";         | string    | TRUE    | FALSE     | TRUE    | FALSE               |
| \$x = null        | NULL      | TRUE    | TRUE      | FALSE   | FALSE               |
| var \$x;          | NULL      | TRUE    | TRUE      | FALSE   | FALSE               |
| \$x не определено | NULL      | TRUE    | TRUE      | FALSE   | FALSE               |
| \$x = array();    | array     | TRUE    | FALSE     | TRUE    | FALSE               |
| \$x = false;      | boolean   | TRUE    | FALSE     | TRUE    | FALSE               |
| \$x = true;       | boolean   | FALSE   | FALSE     | TRUE    | TRUE                |
| \$x = 1;          | integer   | FALSE   | FALSE     | TRUE    | TRUE                |
| \$x = 42;         | integer   | FALSE   | FALSE     | TRUE    | TRUE                |
| \$x = 0;          | integer   | TRUE    | FALSE     | TRUE    | FALSE               |
| \$x = -1;         | integer   | FALSE   | FALSE     | TRUE    | TRUE                |
| \$x = "1";        | string    | FALSE   | FALSE     | TRUE    | TRUE                |
| \$x = "0";        | string    | TRUE    | FALSE     | TRUE    | FALSE               |
| \$x = "-1";       | string    | FALSE   | FALSE     | TRUE    | TRUE                |
| \$x = "php";      | string    | FALSE   | FALSE     | TRUE    | TRUE                |
| \$x = "true";     | string    | FALSE   | FALSE     | TRUE    | TRUE                |
| \$x = "false";    | string    | FALSE   | FALSE     | TRUE    | TRUE                |

**Гибкое (мягкое) сравнение (==)**

|         | TRUE  | FALSE | 1     | 0     | -1    | "1"   | "0"   | "-1"  | NULL  | array() | "php" | ""    |
|---------|-------|-------|-------|-------|-------|-------|-------|-------|-------|---------|-------|-------|
| TRUE    | TRUE  | FALSE | TRUE  | FALSE | TRUE  | TRUE  | FALSE | TRUE  | FALSE | FALSE   | TRUE  | FALSE |
| FALSE   | FALSE | TRUE  | FALSE | TRUE  | FALSE | FALSE | TRUE  | FALSE | TRUE  | TRUE    | FALSE | TRUE  |
| 1       | TRUE  | FALSE | TRUE  | FALSE | FALSE | TRUE  | FALSE | FALSE | FALSE | FALSE   | FALSE | FALSE |
| 0       | FALSE | TRUE  | FALSE | TRUE  | FALSE | FALSE | TRUE  | FALSE | TRUE  | FALSE   | TRUE  | TRUE  |
| -1      | TRUE  | FALSE | FALSE | FALSE | TRUE  | FALSE | FALSE | TRUE  | FALSE | FALSE   | FALSE | FALSE |
| "1"     | TRUE  | FALSE | TRUE  | FALSE | FALSE | TRUE  | FALSE | FALSE | FALSE | FALSE   | FALSE | FALSE |
| "0"     | FALSE | TRUE  | FALSE | TRUE  | FALSE | FALSE | TRUE  | FALSE | FALSE | FALSE   | FALSE | FALSE |
| "-1"    | TRUE  | FALSE | FALSE | FALSE | TRUE  | FALSE | FALSE | TRUE  | FALSE | FALSE   | FALSE | FALSE |
| NULL    | FALSE | TRUE  | FALSE | TRUE  | FALSE | FALSE | FALSE | FALSE | TRUE  | TRUE    | FALSE | TRUE  |
| array() | FALSE | TRUE  | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE | TRUE  | TRUE    | FALSE | FALSE |
| "php"   | TRUE  | FALSE | FALSE | TRUE  | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE   | TRUE  | FALSE |
| ""      | FALSE | TRUE  | FALSE | TRUE  | FALSE | FALSE | FALSE | FALSE | TRUE  | FALSE   | FALSE | TRUE  |

**Жёсткое (строгое) сравнение (==)**

|         | TRUE  | FALSE | 1     | 0     | -1    | "1"   | "0"   | "-1"  | NULL  | array() | "php" | ""    |
|---------|-------|-------|-------|-------|-------|-------|-------|-------|-------|---------|-------|-------|
| TRUE    | TRUE  | FALSE   | FALSE | FALSE |
| FALSE   | FALSE | TRUE  | FALSE   | FALSE | FALSE |
| 1       | FALSE | FALSE | TRUE  | FALSE   | FALSE | FALSE |
| 0       | FALSE | FALSE | FALSE | TRUE  | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE   | FALSE | FALSE |
| -1      | FALSE | FALSE | FALSE | FALSE | TRUE  | FALSE | FALSE | FALSE | FALSE | FALSE   | FALSE | FALSE |
| "1"     | FALSE | FALSE | FALSE | FALSE | FALSE | TRUE  | FALSE | FALSE | FALSE | FALSE   | FALSE | FALSE |
| "0"     | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE | TRUE  | FALSE | FALSE | FALSE   | FALSE | FALSE |
| "-1"    | FALSE | TRUE  | FALSE | FALSE   | FALSE | FALSE |
| NULL    | FALSE | TRUE  | FALSE   | FALSE | FALSE |
| array() | FALSE | TRUE    | FALSE | FALSE |
| "php"   | FALSE   | TRUE  | FALSE |
| ""      | FALSE   | FALSE | TRUE  |

**Небольшая задача для закрепления материала**

В обоих вариантах вам предлагается выяснить, чему равно значение переменной \$c после выполнения кода.

**Вариант 1:**

```
$a="10 cats";
$b='5$a dogs';
$c=$a/$b;
```

Чему (и почему) равна переменная \$c?

**Вариант 2:**

```
$a="10 cats";
$b='5$a dogs';
$c=$a/$b;
```

Чему (и почему) равна переменная \$c?

## 3.6. Основные функции PHP, с которых надо начать

### Функции, которые нужны всегда

В PHP существует несколько функций (и конструкций), которые нужны буквально с первого дня программирования. Рассмотрим их:

```
<?php
echo $var;
print_r($var);
isset($arr[$elem]);
unset($var, $arr[$elem]);
include($filename);
exit('Some message...');
?>
```

### Вывод данных в выходной поток

Для вывода данных в выходной поток используется конструкция echo. Обратите внимание: это – конструкция языка, а НЕ функция. Её параметры НЕ заключаются в скобки!

```
<?php
$a=10;
echo $a;      // 10
echo $a,$a;   // 1010
?>
```

### Отладочный вывод данных

Конструкция echo «не умеет» выводить сложные типы данных. Потому для отладки программ очень удобно использовать функции print\_r() и var\_dump():

```
<?php
$arr=array(10=> 'B', 20 => 'C');
print_r($arr);
// Array
// (
// [10] => B
// [20] => C
// )
```

```
<?php
$arr=array(10=> 'B', 20 => 'C');
var_dump($arr);
// array(2)
// {
// [10] => string(1) 'B'
// [20] => string(1) 'C'
// }
```

### Проверка существования

Проверка существование элемента массива или переменной (или нескольких переменных и нескольких элементов массива) осуществляется функцией iset():

```
<?php
$a=10;
$b=20;
$arr[12]=5;
if (isset($a)) echo 'OK';           // OK
if (isset($a,$b)) echo 'OK';       // OK
if (isset($arr[12])) echo 'OK';    // OK
if (isset($c)) echo 'OK';          // не существует
if (isset($arr[27])) echo 'OK';    // не существует
?>
```

## Удаление переменных и элементов массива

Удаление переменных и элементов массива осуществляется функцией `unset()`:

```
<?php  
$a=10;  
$b=20;  
$c=30;  
$arr[12]=5;  
  
unset($a);  
unset($a,$b);  
unset($arr[12]);  
?>
```

## Досрочное прекращение выполнения скрипта

Для досрочного прекращения выполнения скрипта используется функция `exit()` (или её синоним – `die()`). В случае указания текстового параметра, он выводится в выходной поток перед завершением работы скрипта.

```
<?php  
  
if ($argc < 3)  
{  
    exit('USAGE: php scan.php SOURCEDIR DESTDIR');  
}  
  
?>
```

## Сборка скрипта из нескольких файлов

Для того, чтобы разместить код скрипта в различных файлах, а при выполнении программы «объединить» его в один скрипт, применяются следующие функции:

```
<?php  
include($filename);  
include_once($filename);  
require($filename);  
require_once($filename);  
?>
```

В случае отсутствия файла `include` генерирует предупреждение, а `require` – сообщение об ошибке и останавливает выполнение скрипта. Варианты без `_once` позволяют подключить файл много раз, а с `_once` – подключат указанный файл только один раз.

## Ещё раз, кратко

Итак, основное, что нам потребуется:

- `echo` – для вывода информации «на экран» (читается как «эхо» или «екоу», но НЕ как «эчо»);
- `print_r()` и `var_dump()` – для отладки;
- `isset()` и `unset()` для проверки существования и удаления переменных и элементов массива;
- `exit()` или `die()` – для досрочного завершения выполнения скрипта;
- `include()` и его «собратья» – для «сборки» скрипта из отдельных файлов.

### 3.7. Операторы и управляющие конструкции PHP

### 3.7.1. Операторы PHP

## Что такое оператор

Оператор представляет собой символическое обозначение некоторого действия, выполняемого с операндами в выражении.

Следует помнить, что PHP выполняет автоматическое переключение типов операндов на основании типа оператора.

## Виды операторов в PHP

## Операторы в PHP бывают:

- Унарными – с одним операндом:

```
$b = !$b;  
$i++;
```

- Бинарными – с двумя операндами (таких – абсолютное большинство):






Какие вопросы у вас возникли после прочтения материала по указанной ссылке?

## Оператор присваивания

Оператор присваивания обозначается знаком = и активно используется для инициализации переменных и присваивания переменным результатов вычислений:

```
<?php  
$a = 5;  
$b = $a * 12;  
?>
```

PHP допускает и такую форму присваивания, но её использование **не рекомендуется**, т.к. это сильно усложняет читаемость кода:

```
<?php  
$a = ($b = 4) + 5; // $a присвоено 9, $b присвоено 4  
?>
```

## Сокращённая форма записи оператора присваивания

Для операций + - \* / % . (конкатенация строк) поддерживается сокращённая форма записи оператора присваивания:

```
<?php  
$a = 3;  
$a += 5; // аналогично записи: $a = $a + 5;  
  
$b = 'Hello';  
$b .= ' world!'; // аналогично записи: $b = $b . ' world!';  
?>
```

## Присваивание по ссылке

PHP поддерживает присваивание нескольким переменным ссылки на одну и ту же область памяти. Фактически, переменные становятся «синонимами» друг друга:

```
<?php  
$a = 3;  
$b = &$a;  
$b = 10; // значение $a тоже меняется на 10  
?>
```

## Арифметические операторы

Операция деления всегда возвращает дробный тип, даже если оба операнда были целочисленными (или строками, которые преобразуются в целые числа). Остаток  $\$a \% \$b$  будет отрицательным для отрицательных значений  $\$a$ .

| Оператор          | Действие                       | Пример                   |
|-------------------|--------------------------------|--------------------------|
| $\$a= - \$a;$     | Смена знака                    | $\$a=5; \$a=-\$a; // -5$ |
| $\$c=\$a + \$b;$  | Сложение                       | $\$c=3+$x;$              |
| $\$c=\$a - \$b;$  | Вычитание                      | $\$c=17.6-\$z;$          |
| $\$c=\$a * \$b;$  | Умножение                      | $\$z=\$n*\$x*76;$        |
| $\$c=\$a / \$b;$  | Деление                        | $\$x=\$n/\$y;$           |
| $\$c=\$a \% \$b;$ | Деление по модулю<br>(остаток) | $\$c=5\%2; // 1$         |

## Поразрядные операторы

Эта группа операторов работает с битовыми представлениями значений целочисленных operandов. В основном эти операторы применяются для создания т.н. «битовых масок», для решения задач криптографии и при генерации изображений.

| Оператор          | Действие                          |
|-------------------|-----------------------------------|
| $\$c=\$a \& \$b;$ | Поразрядное И (AND)               |
| $\$c=\$a   \$b;$  | Поразрядное ИЛИ (OR)              |
| $\$c=\$a ^ \$b;$  | Поразрядное исключающее ИЛИ (XOR) |
| $\$c= \sim \$a;$  | Поразрядное отрицание (NOT)       |
| $\$c=\$a << 1;$   | Поразрядный сдвиг влево           |
| $\$c=\$a >> 2;$   | Поразрядный сдвиг вправо          |

**Дополнительное задание** для тех, кто знает, как работают логические операторы, или тех, кто не поленится это узнать. Чему (и почему) равно  $16 | 3$ ?

Ваш ответ: \_\_\_\_\_

Поясните свой ответ:

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

## Логические операторы

Эта группа операторов используется для определения логического значения выражения (в т.ч. с несколькими operandами). Подробнее мы рассмотрим их в разделе, посвящённом управляющим конструкциям.

| Оператор                             | Действие                         |
|--------------------------------------|----------------------------------|
| <code>\$c=\$a and \$b;</code>        | Логическое И (AND)               |
| <code>\$c=\$a or \$b;</code>         | Логическое ИЛИ (OR)              |
| <code>\$c=\$a xor \$b;</code>        | Логическое исключающее ИЛИ (XOR) |
| <code>\$c= ! \$a;</code>             | Логическое отрицание (NOT)       |
| <code>\$c=\$a &amp;&amp; \$b;</code> | Логическое И (AND)               |
| <code>\$c=\$a    \$b;</code>         | Логическое ИЛИ (OR)              |

Смысл двух разных вариантов для операторов «and» и «ог» в том, что они работают с различными приоритетами: or > xor > and > || > && > !.

В логических выражениях true можно заменить на любое отличное от 0 число (чаще всего – на 1), а false – на 0, при условии, что для сравнения не используется оператор жёсткого сравнения (==).

## Операторы сравнения

Операторы сравнения позволяют сравнивать между собой значения переменных.

Обратите внимание: при «мягком» сравнении («==») будет происходить переключение типов.

| Оператор   | Название                      | Результат  |
|--|-------------------------------|--|
| <code>\$a == \$b</code>                                  | Равно                         | TRUE, если \$a равно \$b                         |
| <code>\$a === \$b</code>                                 | Равно по типу и значению      | TRUE, если \$a равно \$b по типу и значению      |
| <code>\$a != \$b</code><br><code>\$a &lt;&gt; \$b</code> | Не равно                      | TRUE, если \$a НЕ равно \$b                      |
| <code>\$a !== \$b</code>                                 | Не равно по типу или значению | TRUE, если \$a НЕ равно \$b по типу ИЛИ значению |
| <code>\$a &lt; \$b</code>                                | Меньше                        | TRUE, если \$a < \$b                             |
| <code>\$a &gt; \$b</code>                                | Больше                        | TRUE, если \$a > \$b                             |
| <code>\$a &lt;= \$b</code>                               | Меньше либо равно             | TRUE, если \$a <= \$b                            |
| <code>\$a &gt;= \$b</code>                               | Больше либо равно             | TRUE, если \$a >= \$b                            |

Рассмотрим пример с использованием операторов == и !=:

```
<?php
$a=1;
$b='1';
if ($a==$b) echo 'OK'; // выведется OK
if ($a===$b) echo 'OK'; // ничего не выведется
if ($a!===$b) echo 'Yes'; // выведется Yes
?>
```

## Оператор маскировки сообщений об ошибках

PHP следует идеологии «если в случае возникновения ошибки можно продолжить работу – работа будет продолжена». При этом сообщение об ошибке будет отображено, что в случае работающего проекта:

- негативно влияет на имидж разработчика;
- может облегчить задачу злоумышленнику.

Отключить вывод сообщений об ошибках можно двумя способами.

**Способ 1:** использовать оператор маскировки сообщений об ошибках (он воздействует только на то выражение, в котором используется).

```
<?php
// Маскировка ошибки при работе с файлами
$my_file = @file ('non_existent_file');

// @ работает и для выражений, а не только для функций
$value = @$arr[$k];
// В случае если ключа $k нет в массиве $arr,
// сообщение об ошибке не будет отображено
?>
```

**Внимание:** оператор `@` не подавляет сообщения о синтаксических ошибках в коде!

**Способ 2:** использовать функцию `error_reporting()` для управления глобальными настройками вывода сообщений об ошибках.

```
<?php
error_reporting(0); // Вывод сообщений об ошибках отключен
?>
```

Действие такого отключения распространяется на всю дальнейшую работу кода до завершения скрипта или вызова функции `error_reporting()` с другими параметрами. Как и `@`, этот способ не подавляет сообщения о синтаксических ошибках в коде.

## Оператор исполнения

PHP поддерживает оператор исполнения: обратные кавычки ``. Обратите внимание, что это не одинарные кавычки, а именно обратные (на клавиатуре расположены под клавишей Esc).

PHP пытается выполнить строку, заключённую в обратные кавычки, как консольную команду, и возвращает полученный в консоли вывод:

```
<?php
$output = `dir`;
echo '<pre>' . $output . '</pre>';
?>
```

Подобного эффекта можно достичь использованием функции `system ( string command [, int &return_var] )`

Эта функция сразу же выводит результат выполнения команды в выходной поток или помещает его в переменную, которая может быть задана вторым аргументом.

**Внимание:** попытка с помощью этого способа запустить GUI-приложение приводит, как правило, к зависанию Apache.

## Операторы инкремента и декремента

PHP поддерживает префиксные и постфиксные операторы инкремента и декремента числовых переменных.

Инкрементирование или декрементирование логических переменных не меняет их значение.

| Оператор            | Название              | Результат  |
|---------------------|-----------------------|--|
| <code>++\$a;</code> | Префиксный инкремент  | Увеличивает значение переменной на 1 и возвращает её значение (новое)          |
| <code>--\$a;</code> | Префиксный декремент  | Уменьшает значение переменной на 1 и возвращает её значение (новое)            |
| <code>\$a++;</code> | Постфиксный инкремент | Возвращает значение переменной (старое) и увеличивает значение переменной на 1 |
| <code>\$a--;</code> | Постфиксный декремент | Возвращает значение переменной (старое) и уменьшает значение переменной на 1   |

Несколько примеров для пояснения:

```
<?php
$a = 5;
echo $a++; // 5
echo $a; // 6

echo ++$a; // 7
echo $a; // 7

echo $a--; // 7
echo $a; // 6

echo --$a; // 5
echo $a; // 5
?>
```

PHP позволяет применять операцию инкремента (но НЕ декремента) к строковым данным:

```
<?php
$i = 'W';
for ($n=0; $n<6; $n++) echo ++$i . " ", ";

// Результат работы будет следующим:
// X, Y, Z, AA, AB, AC,
?>
```

Небольшая задача для закрепления материала: что получится в результате выполнения такого кода?

```
<?php  
    $i=2;  
    $i += $i++ + ++$i;  
    echo $i;  
?>
```

Ваш ответ: \_\_\_\_\_

Поясните, как происходит вычисление:

Проверьте, какой результат даёт аналогичный код в других языках программирования:

## Строковые операторы

В PHP есть один строковый оператор – точка (.). Это оператор конкатенации (склеивания) строк. Он поддерживает сокращённую запись (конкатенацию с присваиванием).

```
<?php
$a = 'Hello';
$b = $a . ' world! ';
// $b содержит строку 'Hello world!'

$a = 'Hello';
$a .= ' world!';
// $a содержит строку 'Hello world!'
?>
```

## Операторы работы с массивами

Подробнее о массивах мы поговорим в отдельной теме, а сейчас просто рассмотрим операторы работы с массивами, чтобы знать, что такие существуют:

| Оператор                 | Название                                       | Результат  |
|--------------------------|--|--|
| \$a + \$b                | Объединение массивов                           | Объединение массивов   |
| \$a == \$b               | Проверка на равенство массивов                 | TRUE, если массивы содержат одинаковые элементы                                    |
| \$a === \$b              | Проверка на тождественное равенство массивов   | TRUE, если массивы содержат одинаковые элементы в том же порядке                   |
| \$a != \$b<br>\$a <> \$b | Проверка на неравенство массивов               | TRUE, если массивы содержат неодинаковые элементы                                  |
| \$a !== \$b              | Проверка на тождественное неравенство массивов | TRUE, если массивы содержат неодинаковые элементы или порядок элементов отличается |

Рассмотрим несколько примеров:

```
<?php
$arr1=array(10 => 'A', 20 => 'B', 30 => 'C');
$arr2=array(10 => 'aaa', 20 => 'bbb', 90 => 'ccc');
$arr3=$arr1+$arr2;
print_r($arr3);

// Array ( [10] => A [20] => B [30] => C [90] => ccc )
?>
```

Обратите внимание, что элементы первого массива с ключами, совпадающими с ключами элементов второго массива, не были перезаписаны.

```
<?php
$arr1=array(10 => 'A', 20 => 'B', 30 => 'C');
$arr2=array(10 => 'A', 20 => 'B', 30 => 'C');
if ($arr1==$arr2) echo 'OK'; // выводится OK

$arr1=array(10 => 'A', 20 => 'B', 30 => 'C');
$arr2=array(10 => 'A', 20 => 'B', 900 => 'C');
if ($arr1==$arr2) echo 'Yes'; // ничего не выводится

$arr1=array(10 => 'A', 20 => 'B', 30 => 'C');
$arr2=array(10 => 'A', 20 => 'B', 900 => 'C');
if ($arr1==$arr2) echo 'Yes'; // ничего не выводится
?>
```

## Оператор проверки принадлежности классу

Оператор `instanceof` используется для определения того, является ли текущий объект экземпляром указанного класса (или его подкласса).

```
<?php
class A {}
class B extends A {}
class C {}
$a = new A();
$b = new B();
$c = new C();
if ($a instanceof A) echo 'A=OK'; // выведется A=OK
if ($b instanceof A) echo 'B=OK'; // выведется B=OK
if ($c instanceof A) echo 'C=OK'; // ничего не выведется
?>
```

Какие вопросы у вас остались после изучения этого раздела?

### 3.7.2. Управляющие конструкции PHP

#### Оператор условия

Оператор условия (if) может использоваться в PHP в нескольких формах.  
Самый простой вариант:

```
<?php  
  
if ($var == 999)  
{  
    // операторы  
}  
  
?>
```

Оператор условия может содержать необязательные секции – elseif и else.

```
<?php  
if ($var == 999)  
{  
    // операторы  
}  
elseif ($var>999)  
{  
    // операторы  
}  
else  
{  
    // операторы  
}  
?>
```

Оператор условия может быть вложенным (в любой секции).

```
<?php  
if ($var == 999)  
{  
    if ($a == 12)  
    {  
        // операторы  
    }  
}
```

Условия могут быть сложными (составными), здесь пригодятся уже рассмотренные нами ранее логические операторы. Обратите внимание: в круглые скобки берётся как всё сложное условие целиком, так и каждое простое условие в его составе по отдельности:

```
<?php  
if (($a!=5) || ($c>12) || ($str=='OK'))  
{  
    // операторы  
}  
?>
```

## Оператор переключения

Оператор переключения (switch) является наиболее удобным средством для организации т.н. «мультиветвления».

```
<?php
switch ($a) // Любой тип данных, кроме массивов и объектов
{
    case 10:
        echo 'Ten';
    break; // Наличие break обязательно в конце каждого case
    case 'ZZZ':
        echo 'Some string';
    break; // Наличие break обязательно в конце каждого case
    default: // Эта секция может отсутствовать
        echo 'Something unknown';
}
?>
```

## Цикл с предусловием

Цикл с предусловием может не выполниться ни разу, т.к. условие проверяется перед выполнением тела цикла:

```
<?php
$i = 0;
while(++$i <= 5)
{
    echo $i.'<br />';
}
?>
```

## Цикл с постусловием

Цикл с постусловием выполнится хотя бы один раз, т.к. условие проверяетсѧ после выполнения тела цикла:

```
<?php
$i = 0;
do
{
    echo $i.'<br />';
}
while (++$i <= 5)
?>
```

## Итерационный цикл

Итерационный цикл выполняется заданное количество раз, причём его синтаксис включает инициализацию счётчика, условие выхода и правило изменения счётчика:

```
<?php
for ($i=0;$i<5;$i++)
{
    echo $i.'<br />';
}
?>
```

Специальная модификация итерационного цикла (foreach) позволяет проходить по любому массиву (по одному уровню для многомерных массивов), на каждом шаге извлекая значение элемента или ключ и значение элемента:

```
<?php
$a = array ('fio' => 'Ivanov I.I', 'age' => 50);
foreach ($a as $k => $v)
{
    echo 'Key = '.$k.' Value = '.$v.'<br />';
}
foreach ($a as $v)
{
    echo 'Value = '.$v.'<br />';
}
?>
```

### Пропуск остатка итерации и выход из цикла

PHP позволяет пропустить выполнение части тела цикла и сразу перейти на следующую итерацию с помощью оператора continue. Досрочный выход из цикла выполняется оператором break.

```
<?php
for ($i=0; $i<999; $i++)
{
    if ($i<500) continue;
    echo $i.'<br />';
    break;
}
?>
```

### Тернарный оператор

Управление выполнением программы можно реализовывать и с помощью тернарного оператора – сокращённого аналога оператора if:

```
<?php
// Пример использования тернарного оператора
$action = (!empty($_POST['action'])) ? 'default' : $_POST['action'];

// Приведённый выше код аналогичен следующему
if (!empty($_POST['action']))
{
    $action = 'default';
}
else
{
    $action = $_POST['action'];
}
?>
```

### 3.8. Математические функции PHP

## Общие сведения

Несмотря на то, что PHP редко используется для решения математических задач, есть ряд функций, использование которых может оказаться полезным.

Полный список математических функций в PHP (на время написания материала – 48 функций) можно увидеть здесь:

<http://www.php.net/manual/en/ref.math.php>

Настоятельно рекомендуется ознакомиться с этим списком функций – хотя бы для того, чтобы знать, что они есть, и «не изобретать велосипеды».

## Округление чисел

`float round ( float $val [, int $precision = 0 [, int $mode = PHP_ROUND_HALF_UP ]] )` – округление числа с заданной точностью (смысл последнего параметра см. в документации).

```
<?php
    echo round(3.4);           // 3
    echo round(3.5);           // 4
    echo round(3.6);           // 4
    echo round(3.6, 0);        // 4
    echo round(1.95583, 2);    // 1.96
    echo round(1241757, -3);   // 1242000
    echo round(5.045, 2);      // 5.05
    echo round(5.055, 2);      // 5.06
?>
```

`float ceil ( float $value )` – округление числа «вверх» до ближайшего целого.  
`float floor ( float $value )` – округление числа «вниз» до ближайшего целого.

```
<?php
    echo ceil(4.3);           // 5
    echo ceil(9.999);         // 10
    echo ceil(3.0);           // 3

    echo floor(4.3);          // 4
    echo floor(9.999);        // 9
    echo floor(5.0);          // 5
?>
```

## Получение случайных чисел

`int mt_rand ( void )` – генерирует целое случайное число в диапазоне от 0 до int `mt_getrandmax(void)`.

`int mt_rand ( int $min , int $max )` – генерирует целое случайное число в заданном диапазоне.

```
<?php
    echo mt_rand();           // например, 23745
    echo mt_rand(1, 999);     // например, 743
?>
```

## Перевод чисел между системами счисления

`string dechex ( int $number )` – выполняет преобразование числа из 10-тичной в 16-ричную СС.

`number hexdec ( string $hex_string )` – выполняет преобразование числа из 16-ричной в 10-тичную СС.

```
<?php
    echo dechex(255); // ff
    echo hexdec('ff'); // 255
?>
```

`string decbin ( int $number )` – выполняет преобразование числа из 10-тичной в 2-ичную СС.

`number bindec ( string $binary_string )` – выполняет преобразование числа из 2-ичной в 10-тичную СС.

```
<?php
    echo decbin(3); // 11
    echo bindec('11'); // 3
?>
```

`string decoct ( int $number )` – выполняет преобразование числа из 10-тичной в 8-ричную СС.

`number octdec ( string $octal_string )` – выполняет преобразование числа из 8-ричной в 10-тичную СС.

```
<?php
    echo decoct(8); // 10
    echo octdec('10'); // 8
?>
```

`string base_convert ( string $number , int $frombase , int $tobase )` – выполняет преобразование числа из \$frombase в \$tobase СС (от 2 до 36).

```
<?php
    echo base_convert('AB', 16, 2); // 10101011
?>
```

## Определение минимального и максимального значения

`mixed min ( array $values )` – определение минимального числа (в массиве чисел)

`mixed min ( mixed $value1 , mixed $value2 [, mixed $... ] )` – определение минимального числа среди набора чисел.

```
<?php
$a = array(5, 3, 9, 2);
echo min($a);           // 2
echo min(9, 12, 6, 8); // 6
?>
```

`mixed max ( array $values )` – определение максимального числа (в массиве чисел)

`mixed max ( mixed $value1 , mixed $value2 [, mixed $... ] )` – определение максимального числа среди набора чисел.

```
<?php
$a = array(5, 3, 9, 2);
echo max($a);           // 9
echo max(9, 12, 6, 8); // 12
?>
```

## И ещё несколько полезных функций

Здесь – просто перечисление ещё нескольких математических функций в PHP:

`number abs ( mixed $number )` – возвращает модуль числа.

`float fmod ( float $x , float $y )` – возвращает дробный остаток от деления.

`float log ( float $arg [, float $base = M_E ] )` – возвращает логарифм числа.

`float pi ( void )` – возвращает значение числа Pi.

`float sqrt ( float $arg )` – возвращает квадратный корень.

## Определение корректности чисел

В PHP многие операции с числами могут привести к переполнению разрядной сетки. Проверить результат выполнения операций с числами на корректность можно с помощью функций:

`bool is_finite ( float $val )` – возвращает TRUE, если в результате операции получилось корректное число.

`bool is_infinite ( float $val )` – возвращает TRUE в случае, если результатом числовой операции является бесконечность (как в случае с `log(0)`) или если произошло переполнение разрядной сетки.

`bool is_nan ( float $val )` – возвращает TRUE, если результат операции не может быть интерпретирован как число.

### 3.9. Функции PHP , определяемые пользователем

#### Небольшое введение

Прежде чем продолжить рассмотрение библиотечных функций PHP, нужно научиться писать свои собственные функции.

PHP следует такой логике при работе с функциями:

- Тип возвращаемых значений не указывается и может быть любым.
- Тип предаваемых параметров не указывается и может быть любым.
- Поддерживается передача параметров по ссылке и по значению.
- Поддерживается переменное количество параметров и параметры со значением по умолчанию.

#### Объявление и вызов функции

В самом простом случае объявление и вызов функции выглядит так:

```
<?php
function sqr($x)
{
    return $x*$x;
}

$y = sqr(2);
echo $y; // 4
?>
```

#### Область видимости переменных

Переменные, объявленные вне функций, недоступны внутри функций (и наоборот). Это же относится и к переданным по значению параметрам.

```
<?php
function fnc($x)
{
    $x = 99;
}

$x = 55;
fnc($x);
echo $x; // 55
?>
```

The diagram shows the scope of variables in the provided PHP code. It uses blue curly braces to group different parts of the code and red text to label specific instances of the variable \$x. The first brace groups the entire function definition (from the start of the function to the closing brace). The second brace groups the line '\$x = 55;' and the line 'echo \$x; // 55'. The third brace groups the line '\$x = 99;' inside the function. Labels are placed near these braces: '\$x здесь' (here) is placed next to the inner \$x in the function, and 'РАЗНЫЕ \$x' (different \$x) is placed next to the outer \$x. This illustrates that the inner \$x is a local variable, while the outer \$x is a separate variable.

## Передача параметров по значению

При передаче параметров по значению создаётся копия переменной; любые изменения параметра внутри функции не влияют на исходную переменную вне функции.

```
<?php
function fnc($x)
{
    $x = $x + 10;
    return $x;
}
$x = 10;
$y = fnc($x);
echo 'X='.$x.' Y='.$y; // X=10 Y=20
?>
```

## Передача параметров по ссылке

При передаче параметров по ссылке работа ведётся с одним и тем же адресом памяти, потому изменения параметра внутри функции приводят к изменению «внешней» переменной.

```
<?php
function fnc(&$x)
{
    $x = $x + 10;
}
$x = 10;
fnc($x);
echo 'X='.$x; // X=20
?>
```

## Параметры со значением по умолчанию

При объявлении параметра можно указывать значение, которое он примет, если не будет передан при вызове функции. Параметры со значением по умолчанию должны объявляться СПРАВА (после обычных).

```
<?php
function fnc($x, $y = 99)
{
    echo 'x=' . $x . ' y=' . $y; // x=10 y=99
}

fnc(10);
?>
```

## Передача переменного количества параметров

В PHP в функцию можно передать больше параметров, чем указано при её объявлении. Внутри функции к параметрам можно получить доступ с помощью трёх сервисных функций:

`int func_num_args ( void )` – возвращает количество переданных при вызове функции параметров.

`mixed func_get_arg ( int $arg_num )` – возвращает параметр по его номеру (нумерация идёт с 0).

`array func_get_args ( void )` – возвращает массив из всех переданных в функцию параметров.

Особо отметим, что передавать по ссылке можно только явным образом объявленные параметры. Пример переменного количества параметров – ниже:

```
<?php
function fnc()
{
    echo func_num_args(); // 3
    echo func_get_arg(1); // B
    print_r(func_get_args()); // Array([0] => A [1] => B [2] => C)
}

fnc('A', 'B', 'C');
?>
```

## Рекурсия

Понятие рекурсии относится к базовым понятиям информатики. Но если надо его напомнить – говорите. Рассмотрим и запишем.

## Анонимные функции и замыкания

**Анонимные функции** (anonymous functions) – функций, которые объявляются в месте использования и не получают уникального идентификатора для доступа к ним.

**Замыкания** (closures) – функции, в теле которых присутствуют (не в качестве параметров) ссылки на переменные, объявленные вне тела этой функции. Такие функции ссылаются на свободные переменные в своём контексте.



[Почитать подробнее...](#)

<http://php.net/manual/en/functions.anonymous.php>  
<http://habrahabr.ru/post/147620/>  
<http://habrahabr.ru/company/mailru/blog/103983/>  
<http://www.ibm.com/developerworks/ru/library/os-php-5.3new2/>

С анонимными функциями всё просто: вы можете описывать функцию там, где можно (нужно) было указывать имя функции:

```
// Использование анонимной функции.
$words = array ('ABC', 'A', 'LongWord');
usort($words, function($a, $b){return (strlen($a)>strlen($b) ? 1 : -1);});
print_r($words);

/*
Array
(
    [0] => A
    [1] => ABC
    [2] => LongWord
)
*/
```

С замыканиями самую малость сложнее. Общий принцип: мы создаём функцию, которая «запоминает» некоторые параметры в своём экземпляре.



[См. примеры в папке...](#)

[01\\_anonymous\\_functions\\_and\\_closures](#)

```
// Использование самого простого замыкания.
$my_rsor_function = function()
{
    return function($a, $b)
    {
        return (strlen($a)>strlen($b) ? -1 : 1);
    }; // Здесь НУЖНА ;
}; // Здесь НУЖНА ;

usort($words, $my_rsor_function());
print_r($words);
```

```

// Использование более сложного замыкания.
// Фактически, здесь -- "фабрика функций".
// В момент "вызыва" $my_sort_function_fabric
// с некоторым параметром возвращается
// функция, которая дальше в своей работе опирается
// на этот параметр.
$my_sort_function_fabric = function($method)
{
    return function($a, $b) use ($method)
    {
        if ($method == 'alphabet')
        {
            return ($a>$b) ? 1 : -1;
        }
        if ($method == 'codesums')
        {
            $cs_a = 0;
            $cs_b = 0;
            for ($i=0; $i<strlen($a); $i++)
            {
                $cs_a += ord($a[$i]);
            }
            for ($i=0; $i<strlen($b); $i++)
            {
                $cs_b += ord($b[$i]);
            }
            return ($cs_a>$cs_b) ? -1 : 1;
        }
        return 0;
    };
};

// Сначала будем получать функцию прямо в месте,
// где надо было бы указывать её имя:
$words = array ('A', 'Z', 'C');

usort($words, $my_sort_function_fabric('alphabet'));
print_r($words);

usort($words, $my_sort_function_fabric('codesums'));
print_r($words);

// И ещё раз, теперь "положим функции в переменные":
$words = array ('A', 'Z', 'C');
$alphabet_sorter = $my_sort_function_fabric('alphabet');
$codesums_sorter = $my_sort_function_fabric('codesums');

usort($words, $alphabet_sorter);
print_r($words);

usort($words, $codesums_sorter);
print_r($words);

/*
var_dump($alphabet_sorter);

object(Closure)#3 (2) {
    ["static"]=>
    array(1) {
        ["method"]=>
        string(8) "alphabet"
    }
    ["parameter"]=>
    array(2) {
        ["$a"]=>
        string(10) "<required>"
        ["$b"]=>
        string(10) "<required>"
    }
}
*/

```

## Пространства имён

**Пространства имён** (namespaces) – способ разрешения конфликтов имён между пользовательским кодом и внутренним кодом PHP или между кодом, написанным разными разработчиками.

Если «совсем на пальцах», то пространства имён позволяют однозначно указать, какой класс/функцию/константу мы имели в виду, если имя используемого элемента не уникально.

Пространство имён определяется в начале файла до любого другого кода:

```
<?php
namespace SomeProject;

const SOME_CONSTANT = 1;
class SomeClass { /* ... */ }
function some_function() { /* ... */ }

?>

<?php
namespace SomeProject\SomeSubproject\SomeSubSubProject;

const SOME_CONSTANT = 1;
class SomeClass { /* ... */ }
function some_function() { /* ... */ }

?>
```

Как это использовать:



См. примеры в папке...  
02\_namespaces

```
<?php
namespace SomeProject\SomeLibrary\EasyURL;

class URL
{
    public $JustForTest_EeasyURL = 1;
}

?>

<?php
namespace SomeProject\SomeLibrary\ComplexURL;

class URL
{
    public $JustForTest_ComplexURL = 1;
}

?>

<?php

require_once('some_library_1.php');
require_once('some_library_2.php');

use \SomeProject\SomeLibrary\EasyURL as Easy;
```

```
use \SomeProject\SomeLibrary\ComplexURL as Complex;

$easy_url_class = new Easy\URL();
$complex_url_class = new Complex\URL();
var_dump($easy_url_class);
var_dump($complex_url_class);

/*
object(SomeProject\SomeLibrary\EasyURL\URL)#1 (1) {
    ["JustForTest_EesyURL"]=>
    int(1)
}
object(SomeProject\SomeLibrary\ComplexURL\URL)#2 (1) {
    ["JustForTest_ComplexURL"]=>
    int(1)
}
*/
?>
```



**Почитать подробнее...**

<http://habrahabr.ru/post/72033/>  
<http://habrahabr.ru/post/72097/>  
<http://habrahabr.ru/post/72150/>  
<http://www.ibm.com/developerworks/ru/library/os-php-5.3new3/>  
<http://www.php.net/manual/ru/language.namespaces.faq.php>

## 3.10. Работа с массивами в PHP

### Общие сведения

Если немного расширить ранее озвученный набор фактов о массивах в PHP, получим, что они:

- Бывают только динамическими (причём меняться может и мерность массива).
- Могут содержать одновременно данные любых типов.
- В качестве ключей (индексов) могут использовать как числа, так и строки.
- Могут быть использованы для хранения таких классических структур как очереди, стеки, деревья и т.д., хотя готовых встроенных инструментов для управления этими структурами в PHP нет.

Сейчас мы рассмотрим основные функции по работе с массивами в PHP.

Полный их перечень можно увидеть здесь:

<http://php.net/manual/en/ref.array.php>

Всего сейчас есть 77 функций по работе с массивами.

Важно! Большинство функций при работе с многомерными массивами обрабатывает только один уровень. Для обработки всего массива в этом случае, надо написать свою рекурсивную функцию.

### Определение количества элементов в массиве

`int count ( mixed $var [, int $mode = COUNT_NORMAL ] )` – определяет количество элементов в массиве. Если необходимо посчитать ВСЕ элементы в многомерном массиве, следует вызвать функцию со вторым параметром, равным COUNT\_RECURSIVE или 1.

```
<?php
$arr = array( array ('A', 'B'), array('C', 'D') );

echo count($arr); // 2
echo count($arr, COUNT_RECURSIVE); // 6
?>
```

### Поиск элемента в массиве

Для поиска элемента по ключу используется ранее рассмотренная функция `isset()`. Для поиска элемента по значению используется функция

`bool in_array ( mixed $needle , array $haystack [, bool $strict = FALSE ] )`

Обратите внимание, что по умолчанию эта функция при поиске элементов использует нестрогое сравнение, что может привести к неожиданным результатам.

```
<?php
$arr = array('A', 99, TRUE);
$x = in_array(99, $arr);           // true
$x = in_array(99999, $arr);        // true (!?!?!?!)
$x = in_array(99999, $arr, TRUE); // false
?>
```

`mixed array_search ( mixed $needle , array $haystack [, bool $strict = false ] )` – производит поиск элемента в массиве и возвращает его ключ (в случае, если такой элемент есть) или FALSE, если элемента нет.

```
<?php
$arr = array('A', 99, TRUE);
$x = array_search(99, $arr);           // 1
$x = array_search(99999, $arr, TRUE); // false
?>
```

## Получение списка ключей массива

При решении некоторых задач возникает необходимость получить список ключей массива в виде элементов другого массива. Это делается с помощью функции

`array array_keys ( array $input [, mixed $search_value = NULL [, bool $strict = false ] ] )` – в случае указания 2-го (и 3-го) параметров, функция вернёт только ключи искомых элементов.

```
<?php
$arr = array('A', 99, TRUE);
$k = array_keys($arr); // array(0, 1, 2)
?>
```

## «Сброс» ключей массива

Часто возникает необходимость привести массив к «строгоиндексированному» виду (с ключами вида 0, 1, 2, 3, 4, ...) Это позволяет сделать функция

```
array array_values ( array $input )

<?php
$arr = array(100 => 'A', 123 => 99, 555 => TRUE);
$k = array_values($arr); // array(0 => 'A', 1 => 99, 2 => TRUE)
?>
```

## Сортировка массива

Основные функции сортировки массива (все они принимают первый параметр по ссылке):

```
bool sort ( array &$array [, int $sort_flags = SORT_REGULAR ] )
```

 – сортирует массив по возрастанию, не сохраняя ключи.  

```
bool rsort ( array &$array [, int $sort_flags = SORT_REGULAR ] )
```

 – сортирует массив по убыванию, не сохраняя ключи.  

```
bool asort ( array &$array [, int $sort_flags = SORT_REGULAR ] )
```

 – сортирует массив по возрастанию, сохраняя ключи.  

```
bool arsort ( array &$array [, int $sort_flags = SORT_REGULAR ] )
```

 – сортирует массив по убыванию, сохраняя ключи.  

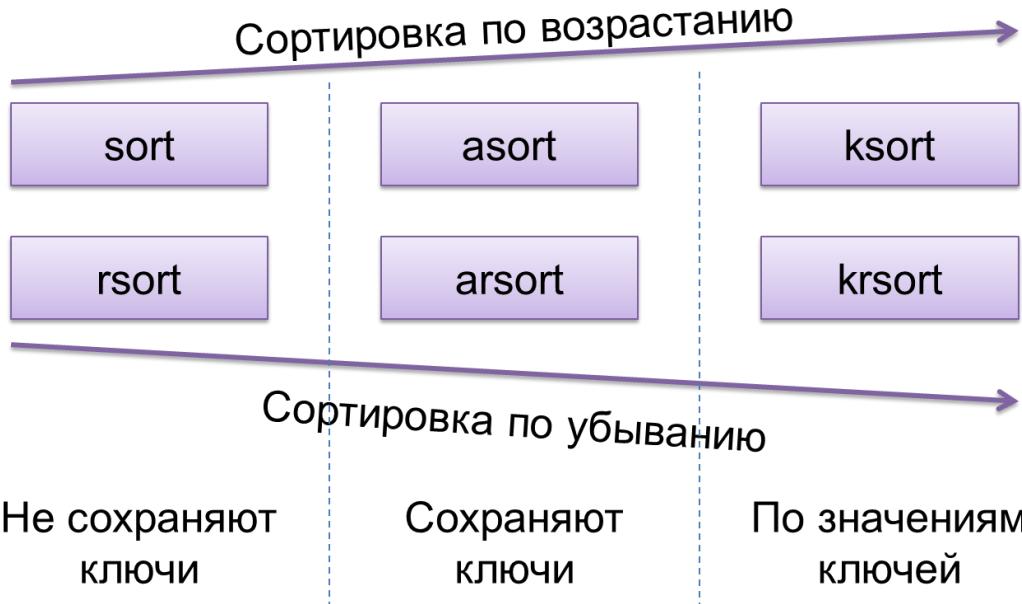
```
bool ksort ( array &$array [, int $sort_flags = SORT_REGULAR ] )
```

 – сортирует массив по возрастанию значений ключей.  

```
bool krsort ( array &$array [, int $sort_flags = SORT_REGULAR ] )
```

 – сортирует массив по убыванию значений ключей.

Для простоты запоминания покажем эти функции на картинке:



Существует ещё одна крайне полезная функция, которая позволяет «не изобретать велосипед», если массив надо отсортировать, принимая какое-то нетривиальное решение о соотношении элементов:

```
bool usort ( array &$array , callable $cmp_function )
```

 – сортирует массив, передавая два сравниваемых элемента в пользовательскую функцию, которая должна возвращать значения: 1 – если первый элемент больше, -1 – если первый элемент меньше, 0 – если элементы равны.

Также см. uasort() и uksort().

Допустим, у нас есть многомерный массив, в котором каждый подмассив – это набор цен товаров в заказе. Надо отсортировать заказы по их стоимости. Решение может выглядеть так:

```
<?php
$orders = array
(
    array(10, 34, 34, 67),
    array(1, 2, 6),
    array(2000, 90),
    array(15, 67),
    array(34, 87, 34)
);

function orders_compare($a, $b)
{
    $sum_a = 0;
    $sum_b = 0;

    foreach ($a as $v)
    {
        $sum_a+=$v;
    }

    foreach ($b as $v)
    {
        $sum_b+=$v;
    }

    if ($sum_a>$sum_b)
    {
        return 1;
    }
    elseif ($sum_a<$sum_b)
    {
        return -1;
    }
    else
    {
        return 0;
    }
}

print_r($orders);
usort($orders, 'orders_compare');
print_r($orders);
?>
```

## Реверс и перемешивание массива

Чтобы изменить порядок элементов массива на обратный используется функция

```
array array reverse ( array $array [, bool $preserve_keys = false ] )
```

которая по умолчанию не сохраняет ключи.

Для перемешивания элементов массива в случайном порядке используется функция

```
bool shuffle ( array &$array )
```

которая всегда сбрасывает (не сохраняет) ключи.

## Извлечение из массива случайных элементов

Для извлечения из массива случайных элементов используется функция

```
mixed array_rand ( array $input [, int $num_req = 1 ] )
```

которая возвращает ключ или массив ключей случайных элементов массива.

P.S. Настоятельно рекомендуется ознакомиться с этим разделом документации, в котором приведено много дополнительной информации о функциях сортировки массивов: <http://www.php.net/manual/en/array.sorting.php>

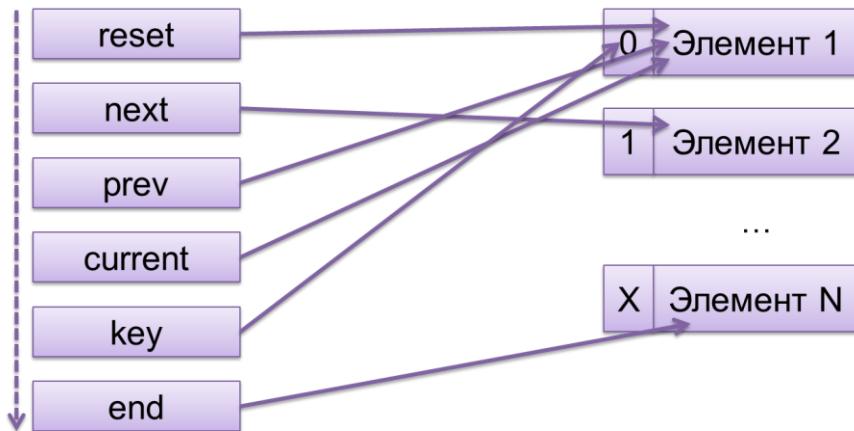
Какие вопросы у вас появились после изучения материала по приведённой ссылке?

## Поэлементная навигация по массиву

PHP предоставляет возможность поэлементно перемещаться по массиву с помощью следующих функций:

`mixed reset ( array &$array )` – возвращает первый элемент массива.  
`mixed end ( array &$array )` – возвращает последний элемент массива.  
`mixed next ( array &$array )` – возвращает следующий элемент массива.  
`mixed prev ( array &$array )` – возвращает предыдущий элемент массива.  
`mixed current ( array &$array )` – возвращает текущий элемент массива.  
`mixed key ( array &$array )` – возвращает ключ текущего элемента массива.

Для запоминания рассмотрим эти функции на картинке:



## Суперглобальные массивы и переменные

В PHP существует некоторое количество предопределённых массивов и переменных, область видимости которых не ограничена ничем, т.е. из любой точки кода к ним можно получить доступ.

Они хранят множество полезной информации, используемой при решении прикладных технических задач. Подробности см. здесь:

<http://www.php.net/manual/en/language.variables.superglobals.php>

Рассмотрим эти массивы и переменные.

### Суперглобальные массивы и переменные: \$GLOBALS

В суперглобальном массиве \$GLOBALS хранятся все переменные, объявленные в глобальной области видимости, что позволяет обращаться к ним изнутри функций и методов.

Подробности: <http://www.php.net/manual/en/reserved.variables.globals.php>

```
<?php
function fnc()
{
    echo $GLOBALS['x']; // 99
}
$x = 99;
fnc();
?>
```

Это – плохой стиль программирования! Не злоупотребляйте!

### Суперглобальные массивы и переменные: \$\_SERVER

В суперглобальном массиве \$\_SERVER хранится разнообразная техническая информация: пути, адреса, параметры клиента и т.п. Этот массив наполняется веб-сервером, потому его содержимое может сильно меняться. Подробности см. здесь: <http://www.php.net/manual/en/reserved.variables.server.php>

```
<?php
echo $_SERVER['REMOTE_ADDR']; // ip клиента
echo $_SERVER['HTTP_USER_AGENT']; // браузер клиента
echo $_SERVER['PHP_SELF']; // путь к скрипту
?>
```

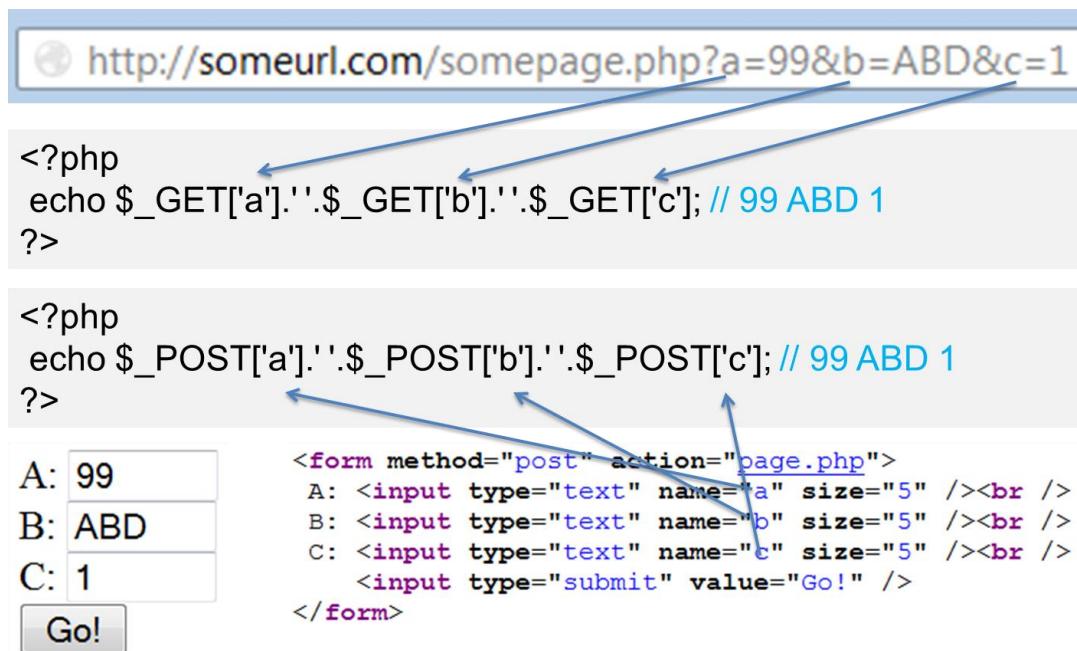
## Суперглобальные массивы и переменные: \$\_GET и \$\_POST

В суперглобальные массивы \$\_GET и \$\_POST попадают данные, переданные в HTTP-запросе методами GET и POST соответственно.

Подробности см. здесь:

<http://www.php.net/manual/en/reserved.variables.get.php>

<http://www.php.net/manual/en/reserved.variables.post.php>



## Суперглобальные массивы и переменные: \$\_FILES

В суперглобальном массиве `$_FILES` содержится информация о переданных на сервер файлах.

Подробности см. здесь:

<http://www.php.net/manual/en/reserved.variables.files.php>

Внимание! Структура массива `$_FILES` отличается в случае передачи одного и нескольких файлов. Отличия предлагаются изучить самостоятельно, поучившись передавать файлы на сервер.

The screenshot shows a web page with a file upload form and its corresponding PHP output. On the left, there is a file upload interface with two fields: 'File 1:' containing 'C:\1.txt' and 'File 2:' containing 'C:\2.html'. Both have 'Browse...' buttons and a 'Go!' button. To the right, a speech bubble icon contains three exclamation marks ('!!!'). Below the form, the raw HTML code is shown:

```
<form method="post" action="page.php" enctype="multipart/form-data">
    File 1: <input type="file" name="f1" size="5" /><br />
    File 2: <input type="file" name="f2" size="5" /><br />
    <input type="submit" value="Go!" />
</form>
```

Below the code, the PHP output is displayed in a light gray box. It shows the contents of the `$_FILES` superglobal array:

```
<?php
print_r($_FILES);
?>
```

```
Array
(
    [f1] => Array
        (
            [name] => 1.txt
            [type] => text/plain
            [tmp_name] => C:\Windows\temp\php132.tmp
            [error] => 0
            [size] => 258
        )
    ...
)
```

## Суперглобальные массивы и переменные: \$\_ENV

В суперглобальном массиве \$\_ENV содержится информация о переменных окружения.

Подробности см. здесь:

<http://www.php.net/manual/en/reserved.variables.environment.php>

Чтобы этот массив был не пустым, в php.ini надо указать его имя в директиве

```
variables_order = "GPCSE"
```

Т.к. регистрация суперглобальных массивов влияет на производительность вместо обращения к \$\_ENV рекомендуется использовать функцию getenv().

`string getenv ( string $varname )` – возвращает значение переменной окружения по её имени.

```
<?php  
echo getenv('PATH'); // C:\WINDOWS\system32;  
?>
```

---

---

---

---

---

---

---

## Суперглобальные массивы и переменные: \$\_SESSION

В суперглобальном массиве \$\_SESSION содержится информация текущей сессии.

Подробности см. здесь:

<http://www.php.net/manual/en/reserved.variables.session.php>

Помимо данного массива для работы сессий используется несколько вспомогательных функций:

```
bool session_start ( void ) – начало сессии.  
bool session_destroy ( void ) – завершение сессии.
```

Вообще, в PHP существует 22 функции по работе с сессиями, но мы пока ограничимся этими двумя, чтобы просто увидеть принцип.

В PHP переменные скрипта существуют только во время его выполнения. Однако, часто возникает необходимость сохранить их значения для последующих запусков скрипта или доступа из других скриптов.

Для достижения этого эффекта используется механизм сессий – поименованных соединений, с которыми ассоциирован некоторый набор данных, хранящихся на сервере. На клиент передаётся только идентификатор сессии (с целью последующего возврата его серверу и продолжения сессии).

---

---

---

---

---

---

---

Примерно такой алгоритм реализуется в каждом скрипте, в котором происходит работа с сессиями.



Сейчас рассмотрим для наглядности исходный код.

Скрипт 1:

```
<?php
session_start();
$_SESSION['x']=999;
?>
```

Скрипт 2:

```
<?php
session_start();
echo $_SESSION['x']; // 999
?>
```

## Суперглобальные массивы и переменные: \$\_COOKIE

В суперглобальном массиве `$_COOKIE` содержится информация о присланных браузером куки.

Подробности см. здесь:

<http://www.php.net/manual/en/reserved.variables.cookies.php>

Помимо данного массива для работы с куки используется функция

```
bool setcookie ( string $name [, string $value [, int $expire = 0 [, string  
$path [, string $domain [, bool $secure = false [, bool $httponly = false  
]]]]]] )
```

которая позволяет управлять установкой, поведением и удалением куки.

Скрипт 1:

```
<?php  
setcookie('CookieName', 999, time() + 3600); // установка на час  
?>
```

Скрипт 2:

```
<?php  
echo $_COOKIE['CookieName']; // 999  
setcookie('CookieName', 0, time() - 3600); // удаление  
?>
```

---

## Куки и сессии: момент установки и запуска

Важно понимать, что `session_start()` и `set_cookie()` модифицируют заголовки HTTP-ответа сервера. Потому эти функции должны быть вызваны ДО вывода чего бы то ни было в выходной поток.

Вывод в выходной поток в PHP начинается автоматически при использовании любой конструкции вывода (`echo`, `print()`, `print_r()`, `var_dump()` и т.д.) или в момент появления в скрипте участка, не обрамлённого «тегами» `<?php ?>`.

Одним из вариантов решения этой проблемы является буферизация вывода: <http://www.php.net/manual/en/ref.outcontrol.php>

---

## Суперглобальные массивы и переменные: `$_REQUEST`

Этот массив собирает в себе всю потенциально опасную (переданную от клиента) информацию – из массивов `$_GET`, `$_POST`, `$_COOKIE`.

Подробности см. здесь:

<http://www.php.net/manual/en/reserved.variables.request.php>

За последовательность импорта массивов `$_GET`, `$_POST`, `$_COOKIE` в массив `$_REQUEST` отвечает настройка PHP `request-order`.

См.: <http://www.php.net/manual/en/ini.core.php#ini.request-order>

---

## Суперглобальные массивы и переменные: `$argc` и `$argv`

Эти массивы содержат в себе информацию о количестве параметров, переданных из командной строки (`$argc`) и сами параметры (`$argv`). 0-м параметров в `$argv` всегда является имя запущенного скрипта:

```
php test.php A B C

<?php
echo $argc; // 4
print_r($argv); // Array ([0] => test.php [1] => A [2] => B [3] => C)
?>
```

## Суперглобальные массивы и переменные: и ещё пара полезных

Для полноты картины рекомендуется самостоятельно рассмотреть следующие предопределённые переменные PHP:

```
$php_errormsg
$HTTP_RAW_POST_DATA
$http_response_header
```

Они нечасто используются в повседневной жизни, но понимать логику их работы полезно.

## Пример для закрепления материала

К настоящему моменту мы уже изучили достаточно, чтобы решать относительно простые задачи.

Сначала будет два примера решения задачи со следующей формулировкой: «Написать функцию, увеличивающую на заданное значение все числовые элементы массива».

Постарайтесь сначала написать своё решение, а только потом смотреть предложенное ниже.

Передача массива по значению:

```
<?php
function array_inc_v($arr, $inc=1)
{
    if (!is_array($arr))
    {
        return $arr;
    }

    foreach ($arr as $k => $v)
    {
        if (is_array($v))
        {
            $arr[$k] = array_inc_v($v, $inc);
        }
        elseif ((is_double($v)) || (is_integer($v)))
        {
            $arr[$k] = $v + $inc;
        }
    }
    return $arr;
}
?>
```

Передача массива по ссылке:

```
<?php
function array_inc(&$arr, $inc=1)
{
    if (!is_array($arr))
    {
        return NULL;
    }

    foreach ($arr as $k => $v)
    {
        if (is_array($v))
        {
            array_inc($v, $inc);
            $arr[$k]=$v;
        }
        elseif ((is_double($v)) || (is_integer($v)))
        {
            $v = $v + $inc; // $v+=$inc;
            $arr[$k]=$v;
        }
    }
}
?>
```

### Задачи для закрепления материала

И теперь вам рекомендуется самостоятельно решить следующие задачи:

1. Определить, что PHP-скрипт запущен из командной строки, а не посредством веб-сервера.
2. Отсортировать многомерный массив по верхнему уровню, рассчитывая «вес» его подмассивов так: каждый числовой элемент даёт 1 «балл», каждый строковый – 2 «балла», элементы иных типов на «вес» не влияют.
3. Определить, сколько раз в многомерном массиве встречается каждое из присутствующих в нём значений элементов простых типов.

### 3.11. Работа со строками в PHP

#### Общие сведения

Если немного расширить ранее озвученный набор фактов о строках в PHP, получим, что:

- на длину строки нет никаких ограничений (в PHP строка может быть такой длины, какой объём памяти может выделить операционная система, естественно, с учётом ограничений, установленных для отдельно выполняющегося скрипта);
- PHP умеет рассматривать строки как массивы символов;
- PHP рассматривает как строки любые последовательности байт (так, например, прочитанный в память графический файл будет с точки зрения PHP строкой).

Сейчас мы рассмотрим основные функции по работе со строками в PHP.

Полный их перечень можно увидеть здесь:

<http://www.php.net/manual/en/ref.strings.php>

Всего сейчас есть 98 функций по работе со строками в однобайтовых кодировках и 55 функций по работе со строками в мультибайтовых кодировках.

В PHP огромное количество задач решается путём выполнения операций над строками, потому рассмотрим это подробно.

#### Однобайтовые и мультибайтовые кодировки строк

В общем случае, если вы знаете функцию для однобайтовых кодировок, а нужна для мультибайтовых, поищите функцию с именем `mb_имяизвестнойвамфункции`. Скорее всего, вы сразу найдёте искомое.

Важно! Внимательно следите за тем, с какими кодировками вы работаете. Огромное количество проблем у начинающих разработчиков появляется из-за того, что в разных местах используются разные кодировки.

#### Кодировки должны совпадать!

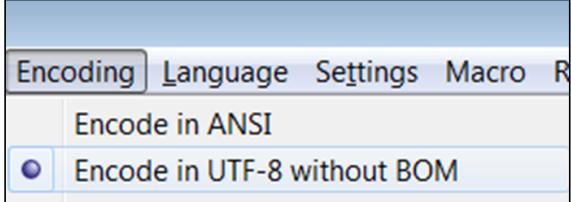
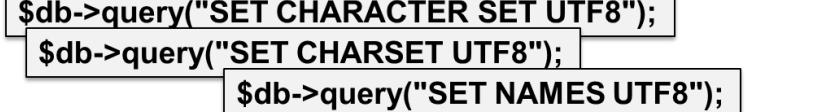
Кодировки должны совпадать в:

- Исходниках.
- Шаблонах.
- Конфигурационных файлах.
- Заголовке генерируемых HTML-страниц.
- Параметрах соединения с СУБД.
- В БД: на уровне БД, таблиц, полей, параметров сортировки.
- ВЕЗДЕ!

Иначе будет вот так

Документо 02 19:15  
ДД2Ñ,Д3/4Д1/2Д3/4Д1/4Д1/2Д°Ñ  
Д°Ñ€Д3/4ÑÑД»Д»Д°Ñ,Ñ,Д3/4Ñ€Д1/4ДµД1/2Д1/2Д°Ñ  
Д1/4Д3/4Д1/2Д3/4Д»Д,Ñ,Д1/2Д°Ñ  
Д°Ñ€Д3/4Д3Ñ€Д°Д1/4Д1/4Д° Д1/2Д° Java  
JAVA\*, C++, C\*  
Д»ÑŽД±Д»ÑŽ desktop-ДžÑ€Д,Д»Д%Д¶ДµД½Д,Ñ. ДÝÑ€Д,Д·Д½Д°Д°Ñ,ÑœÑÑ Д°  
ÑÑ,Д%Д½Д½Ñ·Дµ, Д¢Д%Ñ...Д%Д¶Дµ, ÑÑ,Ñ·Д'Д½ДµДµ, Ñ‡ДµД½Д° ÑД°ÑÐ-ÑÑ...

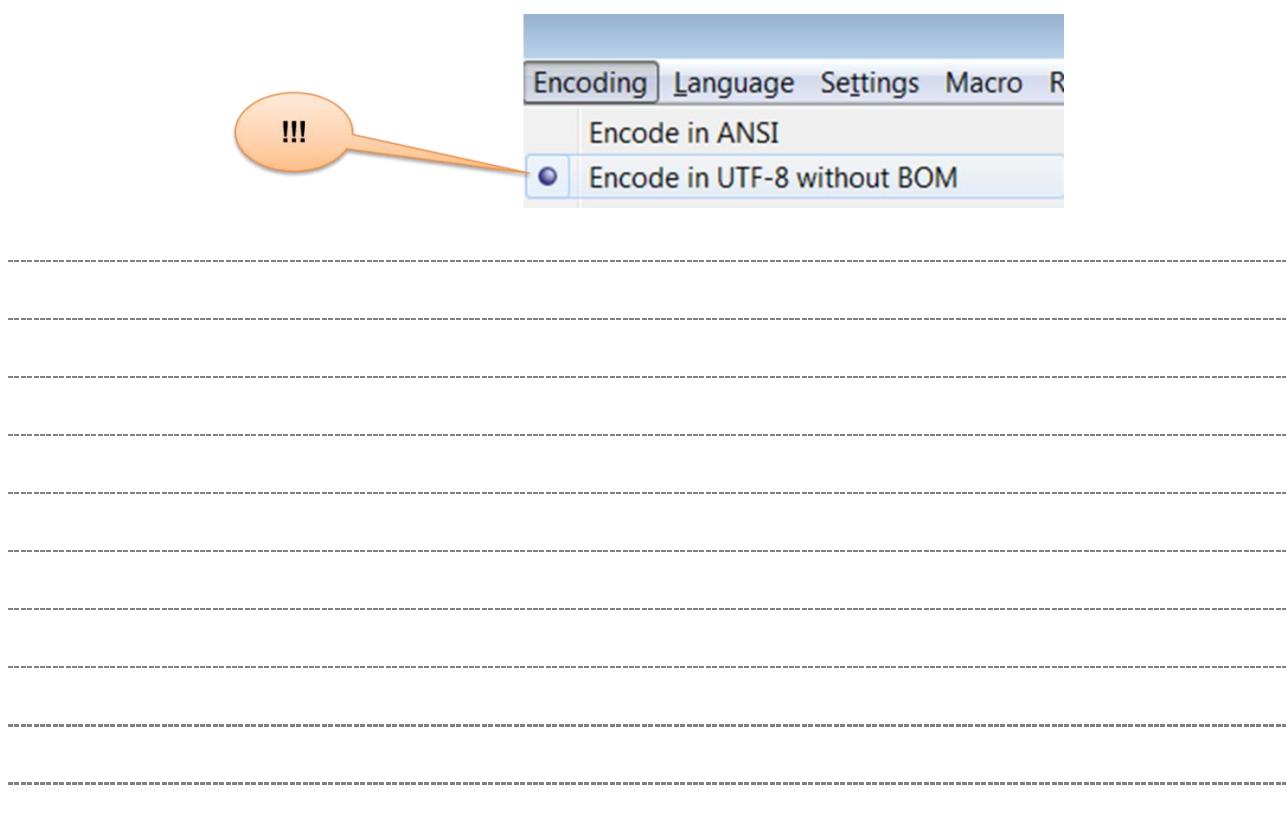
Кодировки должны совпадать в...

|            |  |
|------------|--|
| Исходниках |    |
| Шаблонах   |    |
| «Конфигах» |  |
| Заголовках |  |
| Соединении |  |
| БД         |    |

Если вы сомневаетесь, какую кодировку выбирать, выбирайте UTF8 (или utf8\_general\_ci для значения параметра collation в БД).

Но! Будьте особенно внимательны при сохранении конфигурационных файлов СУБД, веб-серверов и т.п.

Используйте для их правки редакторы, которые умеют сохранять файлы в UTF-8 БЕЗ Byte Order Mark байта (BOM) в начале документа. Иначе возможны проблемы.



## Экранирование символов

В некоторых случаях некоторые символы строк должны быть экранированы, чтобы строка могла быть использована в нужном контексте.

Для этого существует группа функций:

- addslashes / stripslashes
- htmlentities / html\_entity\_decode
- htmlspecialchars / htmlspecialchars\_decode
- preg\_quote
- реализации mysql\_real\_escape\_string
- strip\_tags

Функция

```
string addslashes ( string $str )
```

экранирует символом \ (бэк-слэш) символы: ', ", \, NUL.

Функция

```
string stripslashes ( string $str )
```

убирает такое экранирование.

Допустим, у нас есть такой текст. Обработаем его.

ThisNUL \ is 'a " textNUL

```
<?php
$t1 = file_get_contents('slashes.txt');
echo $t2 = addslashes($t1);    // This\0 \\ is \'a \" text\0
echo $t3 = stripslashes($t2); // This \ is 'a " text
?>
```

Функция

```
string htmlentities ( string $string [, int $flags = ENT_COMPAT | ENT_HTML401
[, string $encoding = 'UTF-8' [, bool $double_encode = true ]]] )
```

подготавливает текст к использованию в HTML-контексте, производя замену недопустимых или непечатных символов их HTML-представлением, например:

```
© → &copy;
< → &lt;
> → &gt;
```

См.: [http://en.wikipedia.org/wiki/List\\_of\\_XML\\_and\\_HTML\\_character\\_entity\\_references](http://en.wikipedia.org/wiki/List_of_XML_and_HTML_character_entity_references)

### Функция

```
string html_entity_decode ( string $string [, int $flags = ENT_COMPAT | ENT_HTML401 [, string $encoding = 'UTF-8' ]] )
```

выполняет обратное преобразование («инвертирует» изменения, сделанные функцией htmlentities()), т.е.:

```
&copy; → ©  
&lt; → <  
&gt; → >
```

### Функции

```
string htmlspecialchars ( string $string [, int $flags = ENT_COMPAT | ENT_HTML401 [, string $encoding = 'UTF-8' [, bool $double_encode = true ]]] )
```

и

```
string htmlspecialchars_decode ( string $string [, int $flags = ENT_COMPAT | ENT_HTML401 ] )
```

работают аналогично только что рассмотренным htmlentities() и html\_entity\_decode() за исключением того, что обрабатывают только следующие символы:

```
& " ' < >
```

Рассмотрим применение этих функций на простом примере. В качестве значения элемента формы передаётся строка «Hotel “Minsk”».

Результат без обработки:

The screenshot shows a simple HTML form element. It consists of a rectangular input field with a blue border containing the text "Hotel". Below the input field is a line of code in red and purple text: "value="Hotel "Minsk""". This demonstrates that the double quotes in the string were not converted by the browser's rendering engine.

Результат с обработкой:

The screenshot shows the same input field and code as the previous one, but with a noticeable difference. The text "Hotel "Minsk"" is now displayed correctly with single quotes around "Minsk", indicating that the htmlspecialchars function has converted the double quotes in the original string into single quotes.

## Функция

```
string preg_quote ( string $str [, string $delimiter = NULL ] )
```

экранирует с помощью \ (бэк-слэш) следующие мета-символы регулярных выражений:

```
. \ + * ? [ ^ ] $ ( ) { } = ! < > | : -
```

Эта функция применяется, если в образец для поиска необходимо передать данные из внешнего источника, чтобы содержащиеся в таких данных вышеназванные символы не исказили обработку образца для поиска.

Подробнее мы поговорим об этом в теме, посвящённой регулярным выражениям.

## Методы

```
string mysqli::real_escape_string ( string $escapestr )
```

и

```
string PDO::quote (string $string [, int $parameter_type = PDO::PARAM_STR ])
```

используются для обработки данных перед передачей их в запрос к MySQL. Эти методы (и процедурный аналог `mysql_real_escape_string()`) используют API MySQL для обработки данных.

Подробности:

<http://www.php.net/manual/en/mysqli.real-escape-string.php>

<http://www.php.net/manual/en/pdo.quote.php>

## Функция

```
string strip_tags ( string $str [, string $allowable_tags ] )
```

используется для удаления HTML-тегов из документа.

**Внимание!** Эта функция не проверяет корректность HTML, а потому в некоторых случаях может повреждать данные.

## Работа с ASCII (символами и их байтовым представлением)

Для преобразования символов ASCII-таблицы в их номера (значения байтов) и обратно используются функции:

```
int ord ( string $string )
```

и

```
string chr ( int $ascii )
```

```
<?php  
echo ord('A'); // 65  
echo chr(65); // A  
?>
```

Подробнее про ASCII-таблицу см. здесь:

<http://en.wikipedia.org/wiki/ASCII>

---

---

---

---

## Разбиение и склеивание строк

Часто для решения прикладных задач над строками приходится выполнять серию операций, которые можно условно назвать «разбиение» и «склеивание». Основными функциями PHP в этом контексте являются:

- explode / implode
- chunk\_split
- str\_split
- str\_getcsv
- nl2br
- parse\_str
- wordwrap

### Функции

```
array explode ( string $delimiter , string $string [, int $limit ] )
```

и

```
string implode ( string $glue , array $pieces )
```

используются для разбиения строки по некоторому разделителю в массив подстрок и склеивания строки из массива строк.

```
<?php
$t1 = 'c:/dir1/dir2/file.ext';
$arr = explode('/', $t1);
$t2 = implode('\\\\\\', $arr);

print_r($arr);
// Array ( [0] => c: [1] => dir1 [2] => dir2 [3] => file.ext )

echo $t2; // c:\\dir1\\dir2\\file.ext
?>
```

### Функция

```
string chunk_split ( string $body [, int $chunklen = 76 [, string $end =
"\r\n" ]] )
```

используется для разбиения длинной строки на отдельные строки длиной \$chunklen символов, после которых идёт указанный признак конца строки.

В основном эта функция применяется для подготовки аттачей к почтовым сообщениями. Потому её использование мы подробнее рассмотрим в соответствующей теме.

### Функция

```
array str_split ( string $string [, int $split_length = 1 ] )
```

используется для разбиения строки в массив подстрок указанной длины.

Эта функция – один из самых удобных способов посимвольного разбиения однобайтовых строк.

```
<?php
$t = 'ABC';
$arr = str_split($t, 1);
print_r($arr); // Array ( [0] => A [1] => B [2] => C )
?>
```

### Функция

```
array str_getcsv ( string $input [, string $delimiter = ',' [, string $enclosure = '"' [, string $escape = '\\\' ]]] )
```

используется для разбиения строки CSV-файла в массив подстрок.

```
<?php
$t = 'coumn 1,column 2,column 3';
$arr = str_getcsv($t);
print_r($arr);
// Array ( [0] => coumn 1 [1] => column 2 [2] => column 3 )
?>
```

### Функция

```
string nl2br ( string $string [, bool $is_xhtml = true ] )
```

используется для подготовки текста, содержащего «классические» признаки конца строки (\n, \r, \r\n, \n\r) к отображению в HTML, где признаком конца строки является тег <br />.

```
<?php
$t = "AB\nCD";
echo nl2br($t);
// AB<br />
// CD
?>
```

### Функция

```
void parse_str ( string $str [, array &$arr ] )
```

используется для автоматического представления т.н. «строки запроса» (GET-запроса) в виде массива, ключами которого являются переменные из этой строки, а значениями – значения этих переменных.

```
<?php
$t = 'a=99&b=55&c=1';
parse_str($t, $arr);
print_r($arr); // Array ( [a] => 99 [b] => 55 [c] => 1 )
?>
```

---

---

---

---

## Функция

```
string wordwrap ( string $str [, int $width = 75 [, string $break = "\n" [, bool $cut = false ]]] )
```

используется для разбиения длинной строки на строки заданной длины с заданным признаком конца строки и возможностью разрезать или не разрезать слова, оказавшиеся «на границе».

**Внимание!** Эта функция НЕ осуществляет «перенос слов по слогам».

```
<?php
$t = 'This is a long string';
echo wordwrap($t, 5, '<br />', FALSE);
// This<br />is a<br />long<br />string

echo wordwrap($t, 5, '<br />', TRUE);
// This<br />is a<br />long<br />strin<br />g
?>
```

## Задачи для закрепления материала

- В тексте даны даты в виде: «01-01-2013 12-12-12 04/04/2012». Показать первую и последнюю даты.
- С помощью функций explode/implode убрать в тексте все повторяющиеся пробелы (оставить один пробел вместо нескольких).
- Длинный текст, представленный одной строкой, отформатировать для вывода в HTML с переносами строк, причём каждая строка должна быть не длиннее 60 символов. Слова на границе строк не разрезать.

## Управление регистром

Для управления регистром символов в PHP существуют следующие функции:

- strtolower / strtoupper
- ucfirst / lcfirst
- ucwords

## Функции

```
string strtolower ( string $str )
```

и

```
string strtoupper ( string $string )
```

используются для перевода всей строки в нижний и верхний регистры соответственно.

```
<?php
echo strtolower('just a TEST string'); // just a test string
echo strtoupper('just a TEST string'); // JUST A TEST STRING
?>
```

## Функции

```
string lcfirst ( string $str )
```

И

```
string ucfirst ( string $str )
```

используются для перевода первого символа строки в нижний и верхний регистры соответственно.

```
<?php
    echo lcfirst('JUST a TEST string'); // JUST a TEST string
    echo ucfirst('just a TEST string'); // Just a TEST string
?>
```

## Функция

```
string ucwords ( string $str )
```

используется для перевода первого символа каждого слова строки верхний регистр.

```
<?php
    echo ucwords('just a TEST string'); // Just A TEST String
?>
```

## Задачи для закрепления материала

- Дан текст (о регистре символов в котором мы ничего не знаем).
  - Привести текст к виду: “Каждая Первая Буква Слова В Верхнем Регистре, Остальные В Нижнем”.
  - Привести текст к виду: “Только первая буква – в верхнем регистре, остальные – в нижнем”.
  - Привести текст к виду: “КАЖДАЯ пЕРВАЯ БУКВА сЛОВА в нИЖНЕМ рЕГИСТРЕ, оСТАЛЬНЫЕ в ВЕРХНЕМ”.

## Длина и символы/слова

Для определения длины строки, набора символов, из которых состоит строка, и количества слов в строке используются функции:

- `strlen`
- `count_chars`
- `str_word_count`

Функция

```
int strlen ( string $string )
```

используется для определения длины строки в байтах.

```
<?php  
echo strlen('ABC'); // 3  
?>
```

Функция

```
mixed count_chars ( string $string [, int $mode = 0 ] )
```

используется для определения того, какие символы и в каком количестве встречаются в строке. Эта функция может возвращать множество результатов, см.:

<http://www.php.net/manual/en/function.count-chars.php>

```
<?php  
$arr = count_chars('test', 1);  
print_r($arr); // Array ( [101] => 1 [115] => 1 [116] => 2 )  
?>
```

Функция

```
mixed str_word_count (string $string [,int $format = 0 [,string $charlist ]])
```

используется для определения количества слов в строке. Эта функция может возвращать различные результаты. См. подробности здесь:

<http://www.php.net/manual/en/function.str-word-count.php>

```
<?php  
echo str_word_count('Just a test'); // 3  
?>
```

## Задачи для закрепления материала

- НЕ используя функцию `count_chars()` определить, сколько раз в строке встречается каждый из присутствующих там символов (например, в слове `test`: `t = 2, e = 1, s = 1`).
- Определить длину самого короткого и самого длинного слова в строке, вывести все самые короткие и самые длинные слова.
- Определить среднюю длину слова в строке.

## Удаление концевых пробелов

Для удаления концевых пробелов (стоящих в начале или конце строки) используются функции:

- trim
- ltrim
- rtrim

Их поведение совершенно тривиально, потому рассмотрим их все сразу.

Синтаксис данных функций таков:

```
string trim ( string $str [, string $charlist = " \t\n\r\x0B" ] )
string ltrim ( string $str [, string $charlist = " \t\n\r\x0B"] )
string rtrim ( string $str [, string $charlist = " \t\n\r\x0B"] )
```

Они удаляют пробелы с обеих сторон, слева и справа соответственно.

```
<?php
echo ['.trim('    Just a test      ') .']'; // [Just a test]
echo ['.ltrim('   Just a test      ') .']'; // [Just a test      ]
echo ['.rtrim('   Just a test      ') .']'; // [   Just a test]
```

## Хэш-функции

Наиболее распространённые функции PHP для получения хэшей от строк (массивов байт) таковы:

- md5
- md5\_file
- sha1
- sha1\_file

Сначала рассмотрим работу этих функций. Функции md5 и sha1 (без слова \_file) обрабатывают строку и возвращают её MD5- и SHA1-хэш соответственно. Функции с постфиксом \_file возвращают хэши файлов:

```
string md5 ( string $str [, bool $raw_output = false ] )
string md5_file ( string $filename [, bool $raw_output = false ] )
string sha1 ( string $str [, bool $raw_output = false ] )
string sha1_file ( string $filename [, bool $raw_output = false ] )

<?php
echo md5('Test'); // 0cbc6611f5540bd0809a388dc95a615b
echo sha1_file('c:/boot.ini'); // 944710dbcd4a072f95a32ea9a23bb749e22d53b4
?>
```

**Хэш-функция** – преобразование по некоторому алгоритму входного массива данных произвольной длины в выходную битовую строку фиксированной длины.

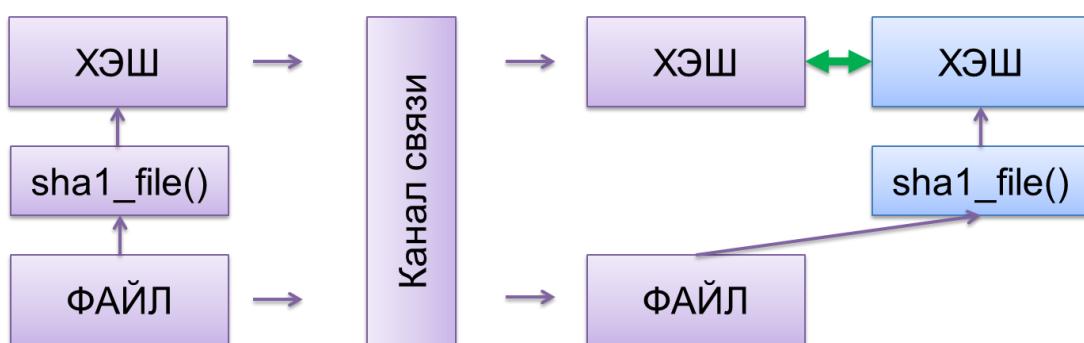
У хэш-функций есть широкий спектр областей применения, см.

<http://ru.wikipedia.org/wiki/Хэширование>

У только что рассмотренных функций есть три основных применения:

- вычисление контрольных сумм файлов;
- генерация псевдослучайных имён файлов;
- хранение паролей.

Вычисление контрольных сумм файлов – простейший способ удостовериться, что файл передан без ошибок. Для этого достаточно на стороне отправителя применить, например, функцию `sha1_file()`, затем передать файл и полученный хэш, а на стороне получателя снова применить эту функцию и сравнить результаты.



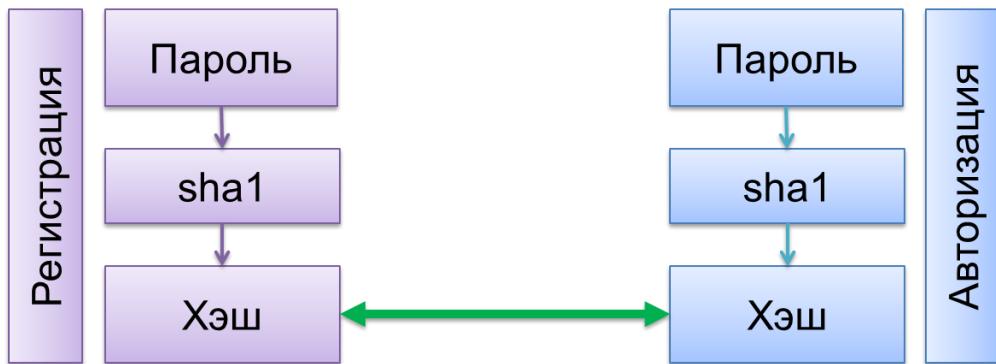
Генерация псевдослучайных имён файлов повсеместно применяется для сохранения на диске полученных от пользователей файлов (в БД хранится исходное имя и «имя-хэш»).

```

<?php
$n = sha1($fn.microtime(TRUE) .mt_rand(1000000, 9999999));
// 9b1a987dcf1ab95f5b6a0721b30d4615c3ea171a
?>
  
```

| <b>id</b> | <b>user_filename</b>        | <b>stored_filename</b>                   |
|-----------|-----------------------------|--|
| 987       | Очень странная картинка.jpg | 9b1a987dcf1ab95f5b6a0721b30d4615c3ea171a |

Пароли, как правило, в целях безопасности хранятся в виде хэшей. Это позволяет снизить риск определения пароля злоумышленником при краже БД паролей. При регистрации пользователя определяется и сохраняется только хэш пароля, а при авторизации снова вычисляется хэш и сравнивается с сохранённым.



## Форматирование

Несмотря на то, что чаще всего форматирование строк для web основывается на CSS, в PHP есть несколько полезных функций, которые могут пригодиться при решении некоторых задач:

- number\_format
- str\_pad
- str\_repeat
- str\_shuffle
- strrev
- wordwrap (рассмотрена ранее)

Функция

```
string number_format ( float $number [, int $decimals = 0 ] )
```

(и она же, в другом синтаксисе)

```
string number_format ( float $number , int $decimals = 0 , string $dec_point  
= '.' , string $thousands_sep = ',' )
```

используется для «человекоудобного» форматирования чисел, например:

```
<?php  
echo number_format(123456.78, 3); // 123,456.780  
echo number_format(123456.78, 3, '.', ','); // 123'456.780  
?>
```

### Функция

```
string str_pad ( string $input , int $pad_length [, string $pad_string = " "
[, int $pad_type = STR_PAD_RIGHT ]] )
```

позволяет дополнять строку указанными символами с указанной стороны до указанной длины, например:

```
<?php
echo str_pad('Test', 10, '*', STR_PAD_RIGHT); // Test*****
echo str_pad('Test', 10, '*', STR_PAD_LEFT); // *****Test
echo str_pad('Test', 10, '*', STR_PAD_BOTH); // ***Test***?
?>
```

### Функция

```
string str_repeat ( string $input , int $multiplier )
```

позволяет сгенерировать строку, состоящую из указанного количества повторений указанной подстроки, например:

```
<?php
echo str_repeat('+-', 5); // +-+-+-+-+
?>
```

### Функция

```
string str_shuffle ( string $str )
```

позволяет перемешать символы в строке в случайном порядке.

```
<?php
echo str_shuffle('ABCDEF'); // ABEFCD
?>
```

### Функция

```
string strrev ( string $string )
```

позволяет разместить символы строки в обратном порядке (сделать реверс строки).

```
<?php
echo strrev('ABCDEF'); // FEDCBA
?>
```

## Задачи для закрепления материала

Самостоятельно реализовать функции, выполняющие те же действия, что и функции:

- str\_pad
- str\_repeat
- str\_shuffle
- strrev
- number\_format [задание повышенной сложности]

## Поиск и замена

Для поиска и замены подстрок в строках в PHP существуют следующие функции:

- str\_replace / str\_ireplace
- strpos / stripos
- strrpos / strripos
- substr\_count
- substr
- substr\_replace (редко используется; рекомендуется изучить самостоятельно)

Функции

```
mixed str_replace ( mixed $search , mixed $replace , mixed $subject [, int &$count ] )
```

и

```
mixed str_ireplace ( mixed $search , mixed $replace , mixed $subject [, int &$count ] )
```

позволяют производить поиск и замену в строке одной подстроки другой подстрокой с учётом и без учёта регистра соответственно.

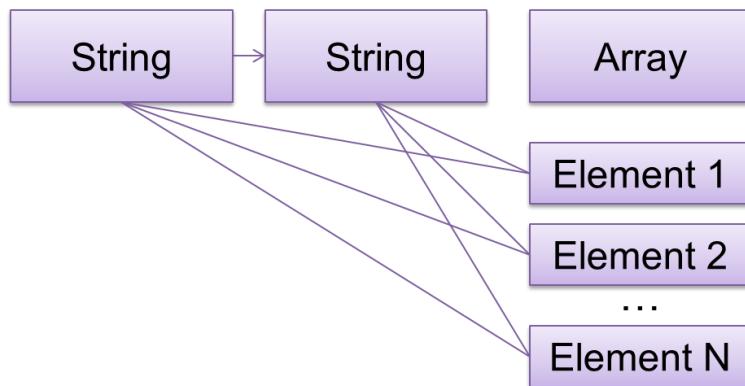
Это одни из наиболее часто используемых функций, потому рассмотрим их работу подробнее.

В общем случае эти функции получают в качестве параметров строки, и тогда их работа выглядит так:

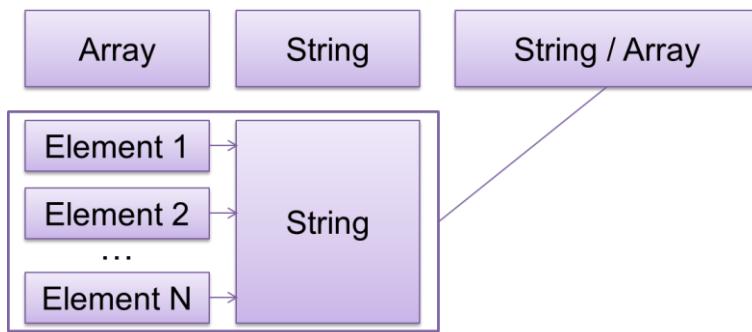
```
<?php
echo str_replace('BC', '*****', 'ABCDEF'); // *****DEF
echo str_ireplace('bc', '*****', 'ABCDEF'); // *****DEF
?>
```

Однако, эти функции могут получать в качестве любого из параметров массив строк. Рассмотрим это.

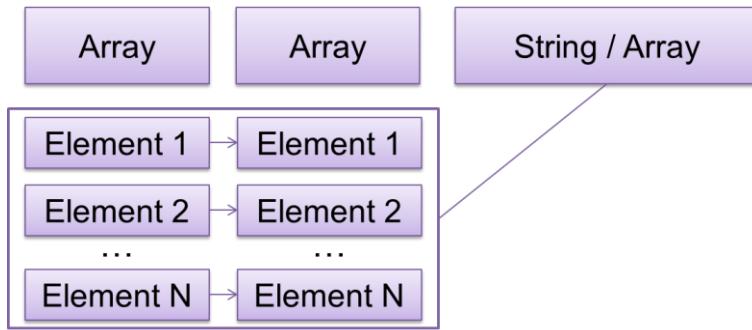
Если массив строк передан третьим параметром, замена будет происходить в каждой из его строк.



Если массив строк передан первым параметром, каждая из его строк будет заменена.

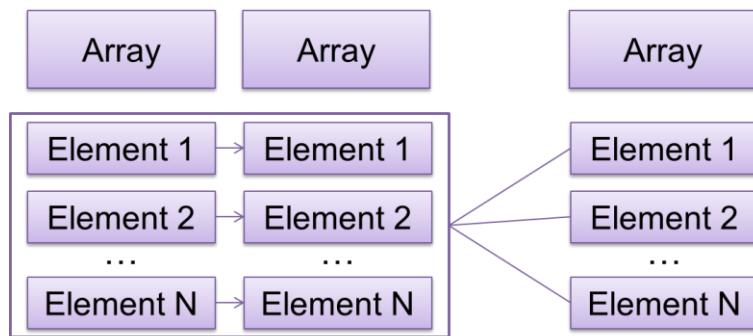


Если массивы строк переданы в качестве двух первых параметров, замена будет происходить поэлементно.



При этом «лишние» элементы первого массива будут заменены на пустые строки, а «лишние» элементы второго – проигнорированы.

В предельном случае массивы могут быть переданы в качестве всех трёх параметров:



### Задача на закрепление материала

Продемонстрировать на примерах следующие варианты работы функции str\_replace:

- (строка, строка, строка)
- (большой\_массив, малый\_массив, строка)
- (большой\_массив, малый\_массив, массив)
- (малый\_массив, большой\_массив, строка)
- (малый\_массив, большой\_массив, массив)
- (массив, строка, массив)
- (массив, строка, строка)

### Поиск и замена (продолжение)

Функции

```
int strpos ( string $haystack , mixed $needle [, int $offset = 0 ] )
```

И

```
int stripos ( string $haystack , string $needle [, int $offset = 0 ] )
```

с учётом и без учёта регистра соответственно ищут первое вхождение подстроки в строку. В случае отсутствия подстроки в строке они возвращают FALSE.

```
<?php
echo strpos('ABCDEF', 'BC'); // 1
echo stripos('ABCDEF', 'bc'); // 1
?>
```

## Функции

```
int strpos ( string $haystack , string $needle [, int $offset = 0 ] )
```

И

```
int strripos ( string $haystack , string $needle [, int $offset = 0 ] )
```

с учётом и без учёта регистра соответственно ищут последнее вхождение подстроки в строку. В случае отсутствия подстроки в строке они возвращают FALSE.

```
<?php
    echo strpos('ABCDEF', 'E'); // 4
    echo strpos('ABCDEF', 'e'); // 4
?>
```

## Функция

```
int substr_count ( string $haystack , string $needle [, int $offset = 0 [, int $length ] ] )
```

производит подсчёт количества вхождений подстроки в строку.

```
<?php
    echo substr_count('this is just a test', 'is'); // 2
?>
```

## Функция

```
string substr ( string $string , int $start [, int $length ] )
```

извлекает из строки \$length символов, начиная со \$start позиции.

```
<?php
    echo substr('this is just a test', 0, 4); // this
?>
```

## Нетривиальное сравнение строк

Несмотря на то, что в PHP операторы сравнения (`==`, `==`, `!=`, `!==`, `<`, `>`, `<=`, `>=`) перегружены для строк, существует обширное количество редкоиспользуемых функций, позволяющих сравнивать строки «нетривиальным образом». Поскольку эти функции действительно используются крайне редко, рекомендуется изучить их самостоятельно.

Вот эти функции: `strcmp()`, `strnatcasecmp()`, `strnatcmp()`, `strncasecmp()`, `strncmp()`, `substr_compare()`, `levenshtein()`, `similar_text()`, `soundex()`.

Какие вопросы у вас возникли после изучения этих функций?

### 3.12. Функции PHP по работе с датой и временем

## Общие сведения

В любом более-менее сложном приложении приходится работать с датой и временем.

Важно понимать, что соответствующие функции PHP оперируют датой и временем в формате UNIXTIME – дате и времени, выраженных в количестве секунд, прошедших с 1 января 1970 года.

Полную информацию по функциям PHP по работе с датой и временем см. здесь: <http://www.php.net/manual/en/ref.datetime.php>

## Получение текущих даты и времени

Для получения информации о текущей дате и текущем времени используется функция

```
int time ( void )  
  
<?php  
echo time(); //  
??
```

Для получения информации о текущей дате и текущем времени солями секунды используется функция

```
mixed microtime ([ bool $get as float = false ] )
```

второй параметр которой позволяет получать результат в виде дроби.

```
<?php
    echo microtime(TRUE); // 1366039679.3925
?>
```

## Работа с UNIXTIME-форматом

Для перевода даты-времени из формата UNIXTIME в «человекопонятный» формат используется функция

```
string date ( string $format [, int $timestamp = time() ] )
```

См. описание спецификаторов формата здесь:  
<http://www.php.net/manual/en/function.date.php>

```
<?php
    echo date('Y.m.d H:i:s'); // 2013.04.15 18:30:30
?>
```

Для перевода даты-времени в формат UNIXTIME из «человекопонятного» формата используется функция

```
int mktime ([ int $hour = date("H") [, int $minute = date("i") [, int $second = date("s") [, int $month = date("n") [, int $day = date("j") [, int $year = date("Y") [, int $is_dst = -1 ]]]]]] )
```

которая каждый не указанный параметр берёт равным текущему значению даты-времени.

```
<?php  
echo mktime(12, 30, 45, 4, 15, 2013); // 1366018245  
echo mktime(12+500, 30, 45, 4, 15, 2013); // 1367818245  
?>
```

## Проверка существования даты

Чтобы выяснить, является ли некая дата корректной с точки зрения календаря, используется функция

```
bool checkdate ( int $month , int $day , int $year )
```

```
<?php  
var_dump(checkdate(2, 29, 2010)); // bool(false)  
var_dump(checkdate(2, 29, 2000)); // bool(true)  
?>
```

## Приостановка выполнения скрипта

Если по какой-то причине выполнение скрипта нужно приостановить на заданное время, используются функции

```
int sleep ( int $seconds )
```

и

```
void usleep ( int $micro_seconds )
```

принимающие свои параметры в секундах и микросекундах соответственно.

```
<?php  
sleep(2);  
usleep(2000000);  
?>
```

## Задача для закрепления материала

Написать скрипт, который строит календарь за указанный год с указанием дней недели.

- [Упрощённый вариант] Календарь может представлять собой просто «ленту дат».
- [Усложнённый вариант] Календарь должен представлять собой таблицу 3x4, где каждая ячейка представляет собой один месяц – т.е. «обычный календарь», который каждый из вас видел сотни раз в жизни.

### 3.13. Функции PHP по работе с файловой системой

#### Общая информация

Следуя общей логике данного курса, мы рассмотрим лишь основные функции PHP по работе с файловой системой. Следует помнить, что поведений той или иной функции может отличаться в зависимости от операционной системы и файловой системы.

Полную информацию о функциях PHP по работе с файловой системой см. здесь: <http://www.php.net/manual/en/ref.filesystem.php>

#### Проверка объекта на существование

Для проверки того факта, что объект файловой системы (в данном случае – файл или каталог) существует, используется функция

```
bool file_exists ( string $filename )  
  
<?php  
var_dump(file_exists('c:/boot.ini')); // bool(true)  
?>
```

#### Распознавание типа объекта

Следующие функции позволяют не только проверить факт существования объекта ФС, но и его тип. Для файлов, каталогов и символьических ссылок используются соответственно функции

```
bool is_file ( string $filename )  
bool is_dir ( string $filename )  
bool is_link ( string $filename )  
  
<?php  
var_dump(is_file('c:/boot.ini')); // bool(true)  
var_dump(is_dir('c:/')); // bool(true)  
var_dump(is_link('c:/boot.ini')); // bool(false)  
?>
```

## Определение времени создания, модификации и последнего доступа к объекту

Следующие функции позволяют определить время создания, модификации объекта ФС и последнего доступа к объекту ФС соответственно.

```
int filectime ( string $filename )
int filemtime ( string $filename )
int fileatime ( string $filename )

<?php
echo filectime('c:/boot.ini'); // 1345029296
echo filemtime('c:/boot.ini'); // 1345018672
echo fileatime('c:/boot.ini'); // 1345018672
?>
```

## Определение размера файла

Следующая функция возвращает размер файла в байтах (Внимание! «Размер» каталога всегда равен нулю. Чтобы выяснить его реальный размер, необходимо рекурсивно сложить размеры всех файлов в нём и его подкаталогах).

```
int filesize ( string $filename )

<?php
echo filesize('c:/boot.ini'); // 211
?>
```

## Чтение из файла и запись в файл

Для чтения текстового файла в массив строк предназначена функция

```
array file ( string $filename [, int $flags = 0 [, resource $context ]] )

<?php
print_r(file('c:/boot.ini'));
/* Array
(
    [0] => [boot loader]
    [1] => timeout=30
    [2] => default=multi(0)disk(0)rdisk(0)partition(1)\WINDOWS
    [3] => [operating systems]
...
) */
?>
```

Для чтения и записи из/в файл предназначены функции

```
string file_get_contents ( string $filename [, bool $use_include_path = false  
[, resource $context [, int $offset = -1 [, int $ maxlen ]]]] )
```

И

```
int file_put_contents ( string $filename , mixed $data [, int $flags = 0 [,  
resource $context ]] )
```

Несмотря на их кажущуюся громоздкость, использовать их очень просто. Рассмотрим пример.

```
<?php  
// Чтение файла целиком  
$x1 = file_get_contents('c:/boot.ini');  
  
// Чтение фрагмента файла  
$x2 = file_get_contents('c:/boot.ini', FALSE, NULL, 100, 50);  
  
// Перезапись файла  
file_put_contents('c:/boot.sav', $x1);  
  
// Добавление информации в конец файла  
file_put_contents('c:/log.txt', "\nNew line...", FILE_APPEND);  
?>
```

Для тех, кто знаком с С-образной работой с файлами, будут полезны функции fopen(), fread(), fwrite(), fclose(), feof(), fseek() и т.д. Мы здесь ограничимся примером того, как они работают:

```
<?php  
$f1=fopen('file1.dat','rb');  
$f2=fopen('file2.dat','wb');  
while (!feof($f1))  
{  
    $x=fread($f1,2048);  
    fwrite($f2,$x);  
}  
fclose($f1);  
fclose($f2);  
?>
```

## Копирование, переименование, удаление

Для копирования, переименования (перемещения) и удаления файлов существуют следующие функции:

```
bool copy ( string $source , string $dest [, resource $context ] )
bool rename ( string $oldname , string $newname [, resource $context ] )
bool unlink ( string $filename [, resource $context ] )
```

Несмотря на то, что rename() в некоторых случаях может работать с каталогами, для простоты будем считать, что эти функции работают только с файлами.

```
<?php
copy('c:/1.txt', 'd:/2.txt');
rename('c:/11.txt', 'd:/22.txt');
unlink('/home/dir/file.ext');
?>
```

## Чтение каталога

Простым (и потому часто неэффективным) способом чтения содержимого каталога является использование функции

```
array scandir ( string $directory [, int $sorting_order = SCAN-
DIR_SORT_ASCENDING [, resource $context ]] )
```

которая возвращает массив имён объектов, содержащихся в каталоге.

```
<?php
$arr = scandir('c:/www_pub/1');
print_r($arr); // Array ( [0] => . [1] => .. [2] => 0.html [3] => env.php )
?>
```

Для полноценной работы с каталогом существуют следующие функции, позволяющие «открыть каталог» (получить его дескриптор), «закрыть», вернуть указатель на текущий объект к началу и прочитать следующий объект соответственно:

```
resource opendir ( string $path [, resource $context ] )
void closedir ([ resource $dir_handle ] )
void rewinddir ([ resource $dir_handle ] )
string readdir ([ resource $dir_handle ] )
```

Обратите внимание, что функция readdir() может вернуть значение, при переключении типов совпадающее с FALSE, потому используется строгое сравнение.

```
<?php
if ($handle = opendir('/path/to/files'))
{
    while (FALSE !== ($file = readdir($handle)))
    {
        echo $file."<br />";
    }
}
closedir($handle);
?>
```

## Определение текущего каталога и смена каталога

Определить текущий каталог и сменить каталог можно следующими функциями:

```
string getcwd ( void )
bool chdir ( string $directory )

<?php
echo $ cwd = getcwd(); // C:\www_pub
chdir('c:/');
echo getcwd(); // C:\
chdir($ cwd);
?>
```

## Создание и удаление каталога

Создать каталог и удалить ПУСТОЙ каталог можно следующими функциями:

```
bool mkdir ( string $pathname [, int $mode = 0777 [, bool $recursive = false
[, resource $context ]]] )
bool rmdir ( string $dirname [, resource $context ] )

<?php
mkdir('c:/1');
rmdir('c:/1');
?>
```

## Определение фрагментов имени файла

Для определения фрагментов имени файла используется функция:

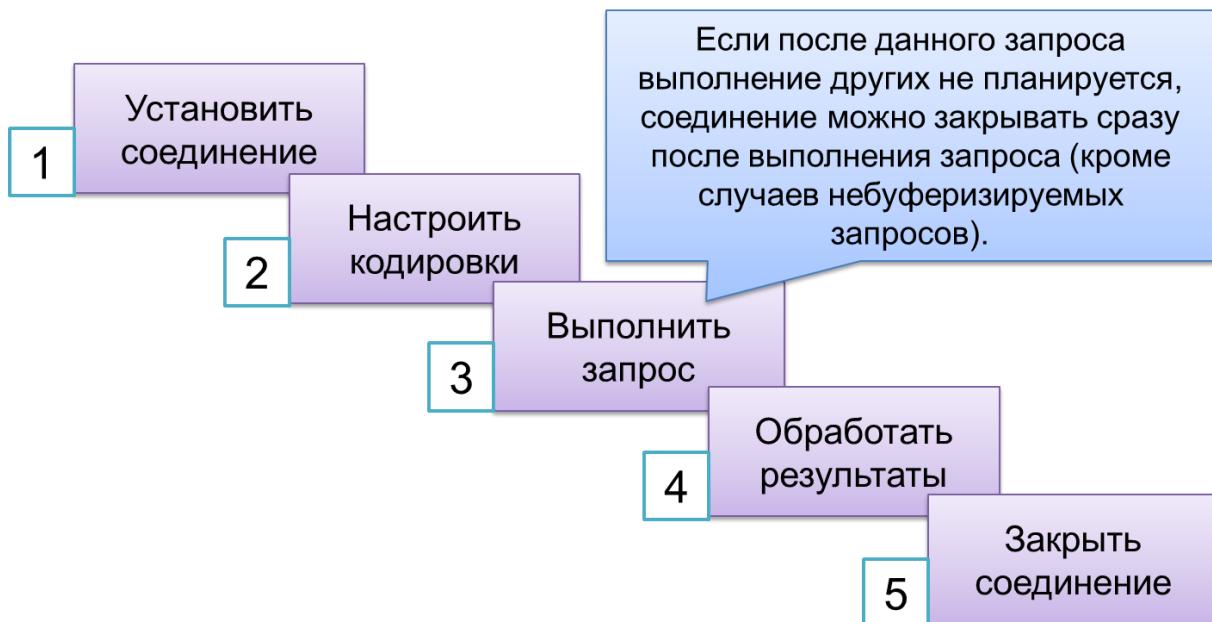
```
mixed pathinfo ( string $path [, int $options = PATHINFO_DIRNAME | PATH-
INFO_BASENAME | PATHINFO_EXTENSION | PATHINFO_FILENAME ] )
```

которая может возвращать массив фрагментов имени файла или отдельный фрагмент имени файла.

```
<?php
print_r(pathinfo('c:/dir1/dir2/file.ext'));
// Array ( [dirname] => c:/dir1/dir2 [basename] => file.ext
// [extension] => ext [filename] => file )
?>
```

### 3.14. Элементарные функции PHP по работе с базами данных

#### Общий алгоритм работы с БД



#### Способы работы с БД из PHP

Долгие годы основным способом работы с БД с помощью PHP было использование расширения `mysql`, однако с недавнего времени оно объявлено устаревшим, а на смену ему пришли две альтернативы:

- MySQL Improved (`mysqli`): <http://www.php.net/manual/en/book mysqli.php>
- PHP Data Objects (PDO): <http://www.php.net/manual/en/book pdo.php>

Оба этих решения претендуют при полном рассмотрении на отдельный курс, потому сейчас мы рассмотрим лишь необходимый минимум.

Несмотря на то, что мы ещё не рассматривали ООП в PHP, мы будем ориентироваться на объектную реализацию работы с MySQL, чтобы вырабатывать правильное понимание.

Если по какой-то причине вы принципиально не хотите использовать объектную реализацию, вы всегда можете обратиться к официальной документации и использовать процедурную.

Поскольку работа с PDO явно выходит за рамки «вводного курса», сейчас мы будем рассматривать работу с `mysqli`.

## Соединение с СУБД

Для установки соединения с СУБД достаточно создать экземпляр класса mysqli, передав в конструктор некоторые параметры:

```
mysqli::__construct() ([ string $host = ini_get("mysqli.default_host") [, string $username = ini_get("mysqli.default_user") [, string $passwd = ini_get("mysqli.default_pw") [, string $dbname = "" [, int $port = ini_get("mysqli.default_port") [, string $socket = ini_get("mysqli.default_socket") ]]]]]])  
<?php  
$mysqli = new mysqli('localhost', 'root', '123456', 'site');  
?>
```

В общем случае этого достаточно.

## Настройка кодировок

Чтобы быть уверенным в том, что «общение» с СУБД будет происходить в нужной кодировке (считаем, что для нас это – UTF8) нужно выполнить такой код:

```
<?php  
$mysqli->query("SET CHARACTER SET 'UTF8'");  
$mysqli->query("SET CHARSET 'UTF8'");  
$mysqli->query("SET NAMES 'UTF8'");  
?>
```

Подобного эффекта можно добиться применением метода

```
bool mysqli::set_charset ( string $charset )  
<?php  
$mysqli->set_charset('UTF8');  
?>
```

## Выполнение запросов

Мы только что уже выполняли запросы, когда настраивали кодировки. Запрос выполняется с помощью метода:

```
mixed mysqli::query(string $query [, int $resultmode = MYSQLI_STORE_RESULT ])  
<?php  
// Добавление данных  
$result = $mysqli->query("INSERT INTO `site_users` (`su_fio`, `su_email`,  
`su_login`, `su_password`) VALUES ('Smith J.', 'smith@gmail.com', 'smith',  
'.sha1('smith')).'))";  
// Чтение данных  
$result = $mysqli->query("SELECT * FROM `site_users`");  
?>
```

## Обработка результатов запроса

Из всего многообразия способов получения результатов выполнения запроса самым простым и универсальным является вызов метода

```
array mysqli_result::fetch_assoc ( void )  
  
<?php  
$result = $mysqli->query("SELECT * FROM `site_users`");  
if ($result !== FALSE)  
{  
    while ($row = $result->fetch_assoc())  
    {  
        echo $row['su_fio'].' ('.$row['su_email'].')';  
    }  
    $result->free();  
}  
?>
```

## Закрытие соединения

PHP автоматически закроет соединение с СУБД по завершении работы скрипта, но во избежание сложнодиагностируемых проблем соединение рекомендуется закрывать явно вызовом метода

```
bool mysqli::close ( void )  
  
<?php  
// Закрытие соединения  
$mysqli->close();  
?>
```

## Весь алгоритм в одном исходнике

```
<?php  
  
// Установка соединения  
$mysqli = new mysqli('localhost', 'root', '123456', 'site');  
  
// Настройка кодировок  
echo $mysqli->query("SET CHARACTER SET 'UTF8'");  
echo $mysqli->query("SET CHARSET 'UTF8'");  
echo $mysqli->query("SET NAMES 'UTF8'");  
  
// Выполнение запроса  
$result = $mysqli->query("SELECT * FROM `site_users`");  
  
// Обработка результата  
if ($result !== FALSE)  
{  
    while ($row = $result->fetch_assoc())  
    {  
        echo $row['su_fio'].' ('.$row['su_email'].')';  
    }  
    $result->free();  
}  
  
// Закрытие соединения  
$mysqli->close();  
?>
```

## 3.15. Работа с XML на PHP

### Общие (предельно краткие) сведения об XML

XML – eXtensible Markup Language (расширяемый язык разметки). Он базируется на языке Standard Generalized Markup Language (SGML, стандартный обобщённый язык разметки) и служит для описания данных.

XML теги не определены изначально, их надо определять отдельно. Для описания правил XML данных используются DTD (Document Type Definition, определение типа документа) или XML Schema.

XML-данные могут храниться как в отдельном файле так и внутри HTML который будет отвечать только за формат отображения, но не за данные. XML можно использовать для обмена информацией между различными (в т.ч. несовместимыми) системами. XML используется как для хранения данных в файловой системе, так и для хранения и выборки информации в базе данных. XML упрощает доступ к данным для пользователей интернет (RSS, Web Services).

XML даёт жизнь производным от XML языкам (например, WAP и WML). XML активно используется для обмена информацией в AJAX-приложениях между клиентом и сервером.

### Общее XML и HTML

- Использование ключевых символов < > & для обозначения элементов языка.
- Однаковое оформление комментариев <!-- вот в таком стиле -->.

### Отличия XML и HTML

- HTML-браузеры «прощают» ошибки в синтаксисе языка и трактуют их по-своему; XML parser'ы выдают сообщение об ошибке и прекращают работу.
- XML значительно более строгий язык, построенный на основе чётких правил, соблюдение которых должно быть неукоснительным.
- В XML не удаляются лишние пробельные символы в начале/середине/конце строки.
- В XML новые строки обозначаются как \n (LF).

### Пример XML-документа

```
<?xml version="1.0"?>
<note>
  <to>Jack</to>
  <from>John</from>
  <heading>Reminder</heading>
  <body>Don't forget to call Boss!</body>
</note>
```

## Преимущества XML

- XML позволяет создавать собственные именованные структуры для хранения информации.
- Задача разбора (parsing) XML хорошо сформулирована и имеет много реализаций.
- XML использует Unicode, что упрощает разработку интернациональных документов.
- Анализаторы XML позволяют проводить проверку структуры документа и типов данных.
- XML – текстовый формат, что делает его более удобным для чтения, документирования и отладки.
- Инструменты для работы с XML доступны на всех платформах.
- XML позволяет использовать инфраструктуру, созданную для HTML (включая HTTP и некоторые браузеры).

## Недостатки XML

- XML документы менее лаконичны чем аналогичные бинарные форматы .
- Передача XML создаёт больший трафик либо больше загружает процессор (если используется сжатие).
- Разбор XML может быть более медленным и требовательным к памяти, чем разбор оптимизированных бинарных документов.

## Хорошо оформленный (well-formed) XML

Хорошо оформленный XML (well-formed XML) – XML-документ, удовлетворяющий следующему набору правил:

- Один корневой элемент.
- Все теги закрыты.
- Учтён регистр тегов.
- Учтены правила вложенности тегов.
- Атрибуты тегов взяты в кавычки.
- Символы <, >, & не могут использоваться в текстовых блоках.

Один корневой элемент:

```
<?xml version="1.0" encoding="UTF-8"?>
<person>
    <givenName>Peter</givenName>
    <familyName>Kress</familyName>
</person>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<person>
    <givenName>Peter</givenName>
    <familyName>Kress</familyName>
</person>
<person>
    <givenName>John</givenName>
    <familyName>Smith</familyName>
</person>
```

Ошибка!

Все теги закрыты:

```
<?xml version="1.0" encoding="UTF-8"?>
<person>
    <givenName>Peter</givenName>
    <familyName>Kress</familyName>
</person>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<person>
    <givenName>Peter</givenName>
    <familyName>Kress </familyName>
</person>
```

Ошибка!

Учтён регистр тегов:

```
<?xml version="1.0" encoding="UTF-8"?>
<person>
    <givenName>Peter</givenName>
    <familyName>Kress</familyName>
</person>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<person>
    <givenName>Peter</givenName>
    <familyName>Kress</FAMILYNAME>
</person>
```

Ошибка!

Учтены правила вложенности тегов:

```
<?xml version="1.0" encoding="UTF-8"?>
<person>
    <givenName>Peter</givenName>
    <familyName>Kress</familyName>
</person>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<person>
    <givenName>Peter
    <familyName>
        </givenName>Kress
        </familyName>
</person>
```

Ошибка!

Атрибуты тегов взяты в кавычки:

```
<?xml version="1.0" encoding="UTF-8"?>
<person>
    <givenName type="official">Peter</givenName>
    <familyName>Kress</familyName>
</person>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<person>
    <givenName type=official>Peter</givenName>
    <familyName>Kress</familyName>
</person>
```

Ошибка!

Символов <, >, & нет в тексте

```
<?xml version="1.0" encoding="UTF-8"?>
<descr>
    <Name>Peter & Jane</Name>
</descr>

<?xml version="1.0" encoding="UTF-8"?>
<descr>
    <Name>Peter & Jane</Name>
</descr>
```

Ошибка!



**Почитать подробнее...**

[http://en.wikipedia.org/wiki/List\\_of\\_XML\\_and\\_HTML\\_character\\_entity\\_references](http://en.wikipedia.org/wiki/List_of_XML_and_HTML_character_entity_references)

Пара примеров того, как специальные символы должны быть представлены в XML:

|        |   |
|--------|---|
| &lt;   | < |
| &gt;   | > |
| &amp;  | & |
| &apos; | ' |
| &quot; | " |

### Основные подходы к работе с XML на PHP

В общем случае для работы с XML в PHP можно использовать:

- DOM (<http://www.php.net/manual/en/book.dom.php>)
- SAX (<http://php.net/manual/en/book.xml.php>)
- SimpleXML (<http://www.php.net/manual/en/book.simplexml.php>) как самый простой.

**Q:** В чём разница работы DOM-парсеров и SAX-парсеров?

**A:** DOM-парсеры (как следует из их названия) сначала получают полноценную DOM-структуру XML-документа, а потом представляют нам возможность работы с ней. SAX-парсеры читают XML «понемножку» и реагируют на события вида «обнаружено открытие элемента», «обнаружено закрытие элемента» и т.д.

В чём преимущества и недостатки DOM- и SAX-парсеров?

Запишите здесь свой ответ:

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

Потренируемся на таком XML-файле (данный файл взят отсюда: <http://www.php.net/manual/en/dom.examples.php>).

**ВАЖНО!** примеры, которые мы рассмотрим, УЧЕБНЫЕ. Это НЕ «финальное идеальное решение некоей задачи».



## См. примеры в папке...

### 03\_XML

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML V4.1.2//EN"
 "http://www.oasis-open.org/docbook/xml/4.1.2/docbookx.dtd" [
]>
<book id="listing">
  <title>My lists</title>
  <chapter id="books">
    <title>My books</title>
    <para>
      <informaltable>
        <tgroup cols="4">
          <thead>
            <row>
              <entry>Title</entry>
              <entry>Author</entry>
              <entry>Language</entry>
              <entry>ISBN</entry>
            </row>
          </thead>
          <tbody>
            <row>
              <entry>The Grapes of Wrath</entry>
              <entry>John Steinbeck</entry>
              <entry>en</entry>
              <entry>0140186409</entry>
            </row>
            <row>
              <entry>The Pearl</entry>
              <entry>John Steinbeck</entry>
              <entry>en</entry>
              <entry>014017737X</entry>
            </row>
            <row>
              <entry>Samarcande</entry>
              <entry>Amine Maalouf</entry>
              <entry>fr</entry>
              <entry>2253051209</entry>
            </row>
            <!-- TODO: I have a lot of remaining books to add.. -->
          </tbody>
        </tgroup>
      </informaltable>
    </para>
  </chapter>
</book>
```

Пример чтения XML-файла с использованием DOM-парсера:

```
<?php

function dom_to_array($element)
{
    $result = array();

    // Извлекаем атрибуты элемента
    if ($element->hasAttributes())
    {
        foreach ($element->attributes as $attribute)
        {
            $result[$attribute->name] = $attribute->value;
        }
    }

    // Определяем тип элемента (нас интересуют XML_ELEMENT_NODE,
    // XML_TEXT_NODE, XML_DOCUMENT_NODE, XML_DOCUMENT_TYPE_NODE)
    switch($element->nodeType)
    {
        // Сам документ
        case XML_DOCUMENT_NODE:
            foreach ($element->childNodes as $subelement)
            {
                $result[$subelement->nodeName] = dom_to_array($subelement);
            }
            break;

        // Элемент XML-документа
        case XML_ELEMENT_NODE:
            // Это -- небольшое упрощение: предполагаем, что если у
            // элемента есть только один "дочерний элемент" -- то это его значение.
            if ($element->childNodes->length==1)
            {
                $result['value'] = $element->nodeValue;
            }
            else
            {
                $element_count = 2; // Счётчик на случай нескольких элементов
                                    // с одинаковыми именами
                $comment_count = 2; // Счётчик на случай нескольких комментариев
                foreach ($element->childNodes as $subelement)
                {
                    // Если в нашем XML будут текстовые блоки
                    // внутри элементов, этот if надо убрать.
                    if ($subelement->nodeName=='#text')
                    {
                        continue;
                    }

                    // Для комментариев сразу просто извлекаем их значения
                    if ($subelement->nodeName=='#comment')
                    {
                        if (!isset($result['comment']))
                        {
                            $result['comment'] = $subelement->nodeValue;
                        }
                        else
                        {
                            $result['comment' . ($comment_count++)] = $subelement->nodeValue;
                        }
                        continue;
                    }
                }
            }
    }
}
```

```
// Обработка полезных элементов
if (!isset($result[$subelement->nodeName]))
{
    $result[$subelement->nodeName] = dom_to_array($subelement);
}
else
{
    $result[$subelement->nodeName . ($element_count++)]
        = dom_to_array($subelement);
}
}
break;

// Текстовая часть элемента или (в данном случае) значение элемента.
case XML_TEXT_NODE:
    $result['value'] = $element->nodeValue;
break;

// <!DOCTYPE ...>
case XML_DOCUMENT_TYPE_NODE:
    // В итоговом массиве не нужен
break;

// Комментарий
case XML_COMMENT_NODE:
    $result['comment'] = $element->nodeValue;
break;

// Тут можно было бы дописать реакцию на остальные
// виды того, что встречается внутри XML :)
default:
    die('Unsupported element type: '.$element->nodeType);
}
return $result;
}

$data = file_get_contents('sample.xml');
$xml = new DOMDocument();
$xml->loadXML($data);
$xml_array = dom_to_array($xml);
print_r($xml_array);

?>
```

Пример чтения XML-файла с использованием SAX-парсера:

```
<?php

// Массив для хранения разобранного XML
$xml_array = array();

// "Стэк" для временного хранения дочерних элементов
$xml_stack = array();

// Ссылка на родительский элемент (удобно для устранения
// "ложных дочерних элементов")
$xml_parent = NULL;

// Обработка начала элемента
function element_start($parser, $name, $attributes)
{
    // ВАЖНО! Если вместо $GLOBALS[] использовать global -- НЕ РАБОТАЕТ!
    // (Проблемы с областью видимости.)

    // Кладём в массив i-й элемент и его "параметры"
    $index = count($GLOBALS['xml_array']);
    $GLOBALS['xml_array'][$index] = array();
    $GLOBALS['xml_array'][$index]['name'] = $name;
    if (count($attributes)>0)
    {
        $GLOBALS['xml_array'][$index]['attributes'] = $attributes;
    }

    // Сохраняем текущее значение в "стэк" и "переключаем"
    // основной массив на только что созданный массив его
    // дочерних элементов. Этакая "эмуляция рекурсии".
    $GLOBALS['xml_array'][$index]['children'] = array();
    $GLOBALS['xml_stack'][count($GLOBALS['xml_stack'])] =
        &$GLOBALS['xml_array'];
    $GLOBALS['xml_parent'] = &$GLOBALS['xml_array'][$index];
    $GLOBALS['xml_array'] = &$GLOBALS['xml_array'][$index]['children'];
}

// Обработка конца элемента
function element_end($parser, $name)
{
    // Восстанавливаем значение основного массива из "стэка".
    // Убираем из "стэка" ненужное.
    $GLOBALS['xml_array'] =
        &$GLOBALS['xml_stack'][count($GLOBALS['xml_stack'])-1];
    unset($GLOBALS['xml_stack'][count($GLOBALS['xml_stack'])-1]);

    // Обработка ситуации, когда единственным
    // "дочерним элементом" оказалось значение элемента.
    // Перекидываем его в ['value'], а ветку ['children'] удаляем.
    if ((isset($GLOBALS['xml_parent']['children']))&&
        (count($GLOBALS['xml_parent']['children'])==1)&&
        (isset($GLOBALS['xml_parent']['children']['value'])))
    {
        $GLOBALS['xml_parent']['value'] =
            $GLOBALS['xml_parent']['children']['value'];
        unset($GLOBALS['xml_parent']['children']);
    }
}
```

```
// Обработка последовательности символов.  
// В нашем случае -- значений элементов.  
function character_data($parser, $data)  
{  
    $data = trim($data);  
    if (strlen($data)>0)  
    {  
        $GLOBALS['xml_array']['value'] = $data;  
    }  
}  
  
// Обработчик "всего, что не попало под основные обработчики".  
// В нашем случае -- комментариев.  
function default_data($parser, $data)  
{  
    $data = trim($data);  
    if (substr($data, 0, 4)=='<!--')  
    {  
        $data = trim(substr($data, 4, strlen($data)-7));  
        $index = count($GLOBALS['xml_array']);  
        $GLOBALS['xml_array'][$index]['name'] = 'comment';  
        $GLOBALS['xml_array'][$index]['value'] = $data;  
    }  
}  
  
// Создаём парсер и навешиваем обработчики.  
$xml_parser = xml_parser_create();  
xml_set_element_handler($xml_parser, "element_start", "element_end");  
xml_set_character_data_handler ($xml_parser, 'character_data');  
xml_set_default_handler ($xml_parser, 'default_data');  
  
// Отключаем преобразование регистра и сохранение пробельных символов.  
xml_parser_set_option($xml_parser, XML_OPTION_CASE_FOLDING, FALSE);  
xml_parser_set_option($xml_parser, XML_OPTION_SKIP_WHITE, TRUE);  
  
// Открываем файл и читаем его кусочками по 4Kb (НЕ грузим весь в память!)  
$fd = fopen('sample.xml', "r");  
while ($data = fread($fd, 4096))  
{  
    if (!xml_parse($xml_parser, $data, feof($fd)))  
    {  
        die('XML error ['.xml_error_string(xml_get_error_code($xml_parser)).'].  
        at line ['.xml_get_current_line_number($xml_parser).'].');  
    }  
}  
fclose($fd);  
  
xml_parser_free($xml_parser);  
print_r($GLOBALS['xml_array']);  
  
?>
```

Пример чтения XML-файла с использованием SimpleXML:

```
<?php  
  
// На то он и SIMPLE-xml ;)  
  
$xml_data = simplexml_load_file('sample.xml');  
$json_data = json_encode($xml_data);  
$xml_array = json_decode($json_data, TRUE);  
print_r($xml_array);  
  
?>
```

---

### Задание для самоподготовки

Для дополнительной самоподготовки вам рекомендуется с помощью трёх только что рассмотренных подходов к обработке XML создать свой собственный XML-файл.

- В качестве данных для записи можно выбрать такие предметные области как:
  - Фильмотека.
  - Отдел кадров.
  - Каталог товаров.
  - Агрегатор RSS-лент.
  - И т.д. ☺

### 3.16. Обработка ошибочных ситуаций и исключений в PHP

Для обработки ошибочных ситуаций в PHP существует широкий набор функций:

- error\_reporting
- trigger\_error (она же user\_error)
- error\_get\_last
- error\_log
- debug\_backtrace
- debug\_print\_backtrace
- set\_error\_handler
- restore\_error\_handler
- set\_exception\_handler
- restore\_exception\_handler



См. примеры в папке...

#### 04\_error\_handling

Функция

```
int error_reporting ([ int $level ] )
```

позволяет определять и изменять текущий уровень отображения сообщений об ошибках в процессе выполнения скрипта.

В качестве параметра она принимает следующие константы (или их «битовые комбинации»): <http://www.php.net/manual/en/errorfunc.constants.php>



Почитать подробнее...

<http://www.php.net/manual/en/function.error-reporting.php>

Пример использования функции error\_reporting():

```
<?php

// Определяем текущий уровень отображения.
echo error_reporting();

// Отключаем отображение любых сообщений об ошибках.
// Полезно в production environment.
error_reporting(0);

// Включаем показ всех сообщений об ошибках,
// кроме сообщений типа E_NOTICE.
error_reporting(E_ALL ^ E_NOTICE);

?>
```

Функция

```
bool trigger_error ( string $error_msg [, int $error_type = E_USER_NOTICE ] )
```

позволяет генерировать сообщение об ошибке.

В качестве параметров она принимает текст сообщения и тип ошибки:  
<http://www.php.net/manual/en/errorfunc.constants.php>



Почитать подробнее...

<http://www.php.net/manual/en/function.trigger-error.php>

Пример использования функции trigger\_error():

```
<?php

$arr = array(1, 2, 3);
if (!isset($arr[999]))
{
    trigger_error('No 999th element in array!', E_USER_ERROR);
}

?>
```

Функция

array error\_get\_last ( void )

возвращает массив с информацией о последней возникшей ошибке.

**ВНИМАНИЕ!** В случае, если в результате возникновения ошибки скрипт был остановлен, эта функция вам уже ничем не поможет – до её вызова выполнение просто не дойдёт.



**Почитать подробнее...**

<http://www.php.net/manual/en/function.error-get-last.php>

Пример использования функции error\_get\_last():

```
<?php

$arr = array(1, 2, 3);
echo $arr[999];

print_r(error_get_last());

?>

Array
(
    [type] => 8
    [message] => Undefined offset: 999
    [file] => D:\_www\error_handling\error_get_last.php
    [line] => 4
)
```

Функция

bool error\_log ( string \$message [, int \$message\_type = 0 [, string \$destination [, string \$extra\_headers ]]] )

позволяет протоколировать ошибочные ситуации. Она достаточно нетривиальна и чаще всего используется не «сама по себе» а в составе какого-то логгера, потому рассмотрим лишь пару простых примеров.



**Почитать подробнее...**

<http://www.php.net/manual/en/function.error-log.php>

Примеры использования функции error\_log():

```
<?php

$arr = array(1, 2, 3);
if (!isset($arr[999]))
{
    // Запись сообщения в лог, настроенный в опции error_log
    // в конфигурации PHP.
    error_log('No 999th element in array!', 0);

    // Должна быть «настроена почта», иначе не сработает.

    // Отправка сообщения системному администратору.
    error_log('No 999th element in array!', 1, "admin@site.com");

    // Запись сообщения в файл.
    error_log('No 999th element in array!', 3, 'D:/_www/our_error_log.txt');
}
?>
```

По умолчанию просто выводится на экран ☺.

Ни каких «красивостей» не будет, в файл просто запишется эта строка.

#### ФУНКЦИИ

```
array debug_backtrace ([ int $options = DEBUG_BACKTRACE_PROVIDE_OBJECT [, int
$limit = 0 ]] )
И
void debug_print_backtrace ([ int $options = 0 [, int $limit = 0 ]] )
```

возвращают в виде массива (объекта) и выводят подробную отладочную (трассировочную) информацию.



**Почитать подробнее...**

<http://www.php.net/manual/en/function.debug-backtrace.php>

<http://www.php.net/manual/en/function.debug-print-backtrace.php>

Пример использования функций debug\_backtrace() и debug\_print\_backtrace():

```
<?php

function f1()
{
    f2('AGA!');
}

function f2($msg)
{
    print_r(debug_backtrace());
    debug_print_backtrace();
}

f1();

?>
```

```

Array
(
    [0] => Array
        (
            [file] => D:\_www\debug_backtrace__debug_print_backtrace.php
            [line] => 5
            [function] => f2
            [args] => Array
                (
                    [0] => AGA!
                )
        )

    [1] => Array
        (
            [file] => D:\_www\debug_backtrace__debug_print_backtrace.php
            [line] => 14
            [function] => f1
            [args] => Array
                (
                )
        )
)

#0  f2(AGA!) called at
[D:\_www\debug_backtrace__debug_print_backtrace.php:5]
#1  f1() called at [D:\_www\debug_backtrace__debug_print_backtrace.php:14]

```

### ФУНКЦИИ

`mixed set_error_handler ( callable $error_handler [, int $error_types = E_ALL | E_STRICT ] )`  
 И  
`bool restore_error_handler ( void )`

позволяют установить собственный обработчик ошибочных ситуаций и «вернуть предыдущий обработчик».



### Почитать подробнее...

<http://www.php.net/manual/en/function.set-error-handler.php>  
<http://www.php.net/manual/en/function.restore-error-handler.php>

Пример использования функций `set_error_handler()` и `restore_error_handler()`:

```

<?php

function our_error_handler($errno, $errstr, $errfile, $errline)
{
    echo 'Error #      : '.$errno."\n";
    echo 'Error is    : '.$errstr."\n";
    echo 'Error file: '.$errfile."\n";
    echo 'Error line: '.$errline."\n";
}

$old_error_handler = set_error_handler('our_error_handler');
echo 1/0;

restore_error_handler();
echo 1/0;
?>
```

```
Error #  : 2
Error is : Division by zero
Error file: D:\_www\set_error_handler__restore_error_handler.php
Error line: 13

Warning: Division by zero in
D:\_www\set_error_handler__restore_error_handler.php on line 17
```

ФУНКЦИИ  
callable `set_exception_handler` ( callable \$exception\_handler )  
и  
`bool restore_exception_handler` ( void )  
позволяют установить собственный обработчик исключений и «вернуть предыдущий обработчик».



[Почитать подробнее...](http://www.php.net/manual/en/function.set-exception-handler.php)

<http://www.php.net/manual/en/function.set-exception-handler.php>  
<http://www.php.net/manual/en/function.restore-exception-handler.php>

Пример использования функций `set_exception_handler()` и `restore_exception_handler()`:

```
<?php
function our_exception_handler($exception)
{
    echo 'Exception: ' . $exception->getMessage() . "\n";
}

set_exception_handler('our_exception_handler');
throw new Exception('TEST');

// До этого кода выполнение на дойдёт :(
restore_exception_handler();
throw new Exception('TEST');

?>
```

`Exception: TEST`

## Полезные константы

Часто для анализа ошибочных ситуаций (и просто для написания кода) используют следующие предопределённые константы:

- `__LINE__` – номер строки.
- `__FILE__` – полное имя файла.
- `__DIR__` – полное имя каталога.
- `__FUNCTION__` – имя функции.
- `__CLASS__` – имя класса.
- `__TRAIT__` – имя трейта.
- `__METHOD__` – имя метода.
- `__NAMESPACE__` – имя пространства имён.

## Обработка исключений

Обработка исключений – реакция программы на возникновение нештатной ситуации, которая по объективным причинам не может быть обработана кодом, в котором эта ситуация возникла.

**ВАЖНО!** Вся суть исключений состоит в том, чтобы передать обработку нештатной ситуации «вызывающему коду». Если мы можем обработать нештатную ситуацию, «на месте», то механизм исключение использовать НЕ НАДО.

Поясним на примере.

Допустим, мы пишем некоторую библиотеку, и понимаем, что функции, которая будет копировать файл, может быть передано имя несуществующего файла. Тогда такой код логичен:

```
<?php
// Это – функция нашей библиотеки
function copy_file($source, $destination)
{
    if (!is_file($source))
    {
        throw new Exception('Source file ['. $source .'] does not exist!');
    }
    // ... Тут ещё много-много полезного кода :)
}

// А так её кто-то где-то будет использовать
try
{
    copy_file('c:/non_existing_file.ext', 'c:/copy.ext');
}
catch (Exception $e)
{
    echo $e->getMessage();
}
?>
```

А вот такое решение – НЕВЕРНО. Здесь порождение и обработку исключения можно (нужно!) полностью убрать, т.к. мы сами можем решить, что делать в данной ситуации.

```
<?php

try
{
    if (!is_file($filename))
    {
        throw new Exception('File ['. $filename .'] does not exist!');
    }
}
catch (Exception $e)
{
    echo $e->getMessage();
}

?>
```

## Синтаксис обработки исключений

Общий синтаксис обработки исключений в PHP:

```
<?php

try
{
    // Код, в котором может возникнуть исключение.
}
catch (Exception $e)
{
    // Реакция на возникшее исключение.
}
finally
{
    // Код, которые сработает вне зависимости от того,
    // возникло исключение или нет.
}

?>
```

Общий синтаксис порождения исключений в PHP:

```
<?php

if (что-то пошло не так)
{
    throw new Exception(сообщение);
}

?>
```

## Создание собственных исключений

До сих пор мы рассматривали только «встроенный механизм» исключений, но в реальности бывает полезно порождать собственные виды исключений (и реагировать на них).

Очень хороший пример в документации можно найти здесь: <http://www.php.net/manual/en/language.exceptions.extending.php>

А мы рассмотрим чуть более простой, но тоже показательный пример. Так мы можем гибче реагировать на разные исключения:

```
<?php

class TypeMismatchException extends Exception {};
class MathException extends Exception {};

function get_average($numbers)
{
    if (!is_array($numbers))
    {
        throw new TypeMismatchException('Function get_average expects first argument to be an array!');
    }

    if (sizeof($numbers) === 0)
    {
        throw new MathException('Function get_average expects non empty array!');
    }
    // ... Тут должен быть полезный код :).
}
```

```
$test1 = 'Test';
$test2 = array();
$test3 = array(1, 2, 3);

try
{
    $avg = get_average($test1);
}
catch (TypeMismatchException $e)
{
    var_dump($e);
}
catch (MathException $e)
{
    var_dump($e);
}

?>
```



## 3.17. ООП в PHP

### Общие сведения об ООП

ООП – парадигма программирования, в которой основными концепциями являются понятия объектов и классов.

Изначально поддержка ООП в PHP отсутствовала, но в последние годы ситуация резко изменилась, и теперь поддержка ООП в PHP вполне сопоставима с другими языками.



**Почитать подробнее...**

<http://www.php.net/manual/en/language.oop5.php>

Для начала вспомним базовые понятия ООП:

- **Инкапсуляция** – скрытие внутренней реализации поведения класса от внешней среды. Внешней среде доступны только вызовы методов (интерфейс класса).
- **Наследование** – возможность создания нового класса на основе существующего.
- **Полиморфизм** – возможность использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.

Реализация инкапсуляции:

```
class Incapsulation
{
    private $value = 0;

    public function setValue($value)
    {
        if ($this->checkValue($value) == TRUE)
        {
            $this->value = $value;
        }
    }

    public function getValue()
    {
        return $this->value;
    }

    private function checkValue($value)
    {
        return ($value>0) ? TRUE : FALSE;
    }
}
```

Сеттер

Геттер

Внутренняя ре-  
ализация некото-  
рой логики

## Реализация наследования:

```
class Inheritance extends Incapsulation
{
    private function checkValue($value)
    {
        return (($value>0)&&($value<100)) ? TRUE : FALSE;
    }
}
```

Сеттеры и геттеры унаследованы, а проверка значения переопределена (написан одноимённый метод, но выполняющий другие действия)

## Реализация полиморфизма:

```
class Flamable
{
    public function burn()
    {
    }
}

class Candle extends Flamable
{
}

class Torch extends Flamable
{
}

class Bonfire extends Flamable
{
}

$candle = new Candle();
$torch = new Torch();
$bonfire = new Bonfire();

$candle->burn();
$torch->burn();
$bonfire->burn();
```

Будет унаследован свечой, факелом, костром

Все «горючие объекты» можно поджечь, вне зависимости от того, что это за КОНКРЕТНЫЙ объект

## Теперь – немного определений

- Класс (class) – «сборочный чертёж», по которому создаются объекты.
- Объект (object) – экземпляр класса.
- Атрибут (attribute) (data member, instance variable, data field) – элемент данных объекта.
- Метод (method) (class function) – набор действий, доступных объекту.
- Конструктор (constructor) – метод, вызываемый при создании объекта, служит для инициализации атрибутов.
- Деструктор (destructor) – метод, вызываемый при разрушении объекта, служит для deinициализации и завершения работы.

## Области видимости

Методы и свойства, объявленные с соответствующими ключевыми словами, «видны» следующим образом:

- public – «видны» из любой части кода.
- protected – «видны» из самого класса, родительских и дочерних классов.
- private – «видны» только из самого класса.

Как это реализуется в PHP:

```
class Person
{
    public $last_name = '';
    protected $phone_for_family = '';
    private $pin_code = '';

    public function talk()
    {
    }

    protected function familyAffairs()
    {
    }

    private function watchEroticMovies()
    {
    }
}

$someone = new Person();
```

```
class Person
{
    // ...
    private $save_to_file = 'tmp';

    // ...

    public function __construct($last_name='')
    {
        $this->last_name = $last_name;
    }

    public function __destruct()
    {
        if ($this->save_to_file!=='')
        {
            file_put_contents($this->save_to_file, serialize($this));
        }
    }
}
```

## Константы классов

Существует возможность объявлять и использовать внутри классов константы. Это реализуется следующим образом:

```
<?php
class MyClass
{
    const CONSTANT = 'constant value';

    function showConstant()
    {
        echo self::CONSTANT;
    }
}

echo MyClass::CONSTANT;
?>
```

## «Магические методы»

В PHP существует несколько специальных методов класса, используемых в определённых случаях:

- `__construct()` – конструктор.
- `__destruct()` – деструктор.
- `__call()` – срабатывает при вызове недоступного метода.
- `__callStatic()` – срабатывает при вызове недоступного статического метода.
- `__get()` – срабатывает при чтении недоступного свойства.
- `__set()` – срабатывает при установке недоступного свойства.
- `__isset()` – срабатывает при `isset()` и `empty()` на недоступном свойстве.
- `__unset()` – срабатывает при `unset()` на недоступном свойстве.
- `__sleep()` – срабатывает перед сериализацией объекта.
- `__wakeup()` – срабатывает после десериализации объекта.
- `__toString()` – срабатывает при преобразовании объекта в строку.
- `__invoke()` – срабатывает при попытке обращения к объекту как к функции.
- `__set_state()` – срабатывает при вызове `var_export()` на объекте.
- `__clone()` – срабатывает после клонирования объекта на новом объекте.

Рассмотрим все эти методы в одном исходнике:



См. примеры в папке...

## 05\_oop

```
<?php

class MagicClass
{
    public function __construct()
    {
        echo "Constructor\n";
    }

    public function __destruct()
    {
        echo "Destructor\n";
    }

    public function __call($name, $arguments)
    {
        echo "Call of inaccessible method [\".$name.\"] with arguments [\".implode(",",
", $arguments).\"]\n";
    }

    public static function __callStatic($name, $arguments)
    {
        echo "Call of inaccessible static method [\".$name.\"] with arguments
[\".implode(\" , \" , $arguments).\"]\n";
    }

    public function __get($name)
    {
        echo "Attempt to read inaccessible property [\".$name.\"]\n";
    }
    public function __set($name, $value)
    {
        echo "Attempt to set inaccessible property [\".$name.\"] with value
[\".$value.\"]\n";
    }

    public function __isset($name)
    {
        echo "Attempt to call isset() or empty() on inaccessible property
[\".$name.\"]\n";
    }

    public function __unset($name)
    {
        echo "Attempt to call unset() on inaccessible property [\".$name.\"]\n";
    }

    public function __sleep()
    {
        echo "Serialisation detected\n";
        return array();
    }

    public function __wakeup()
    {
        echo "Deserialisation detected\n";
    }
}
```

```
public function __toString()
{
    echo "Conversion to string detected\n";
    return "Conversion to string detected\n";
}

public function __invoke()
{
    echo "Access to object as function detected\n";
}

public function __set_state()
{
    echo "var_export() was applied to the object\n";
}

public function __clone()
{
    echo "Object cloned successfully\n";
}

}

$mc = new MagicClass;
$mc->NonExistingMethod('A', 'B', 'C');
$mc::NonExistingStaticMethod('A', 'B', 'C');
echo $mc->NonExistingProperty;
$mc->NonExistingProperty=999;
if (isset($mc->NonExistingProperty)) {}
unset($mc->NonExistingProperty);
$s = serialize($mc);
$x = unserialize($s);
echo $mc;
$mc();
var_export($mc);
$mc2 = clone $mc;

?>
```

## Сравнение объектов

Для сравнения объектов можно использовать операторы == и ===.

Оператор == считает объекты равными, если они – объекты одного класса, а также содержат одинаковые свойства и методы.

Оператор === считает объекты равными, если «это один и тот же объект», т.е. две переменные ссылаются на один объект.

## Подсказки по типам данных

Начиная с PHP 5, в методах классов можно указывать тип ожидаемого параметра. В качестве указываемого типа допустимо: имя класса, имя интерфейса, array, callable. В случае указания имени класса и интерфейса, допустимыми считаются и «наследники» соответствующего класса или интерфейса.

Скалярные типы (например, int или string), ресурсы и трейты нельзя использовать для указания «подсказок по типам данных».



[Почитать подробнее...](http://www.php.net/manual/en/language.oop5.typehinting.php)

<http://www.php.net/manual/en/language.oop5.typehinting.php>

## Интерфейсы

Интерфейсы позволяют определить список методов, которые должны быть реализованы классом, имплементирующим интерфейс.

```
// Определение интерфейса
interface iStorableInDB
{
    public function store();
    public function retrieve();
}

// Имплементация интерфейса классом
class LongConnection implements iStorableInDB
{
    // ...
}
```

## Трейты

Трейты – это механизм повторного использования кода в условиях отсутствия поддержки множественного наследования.

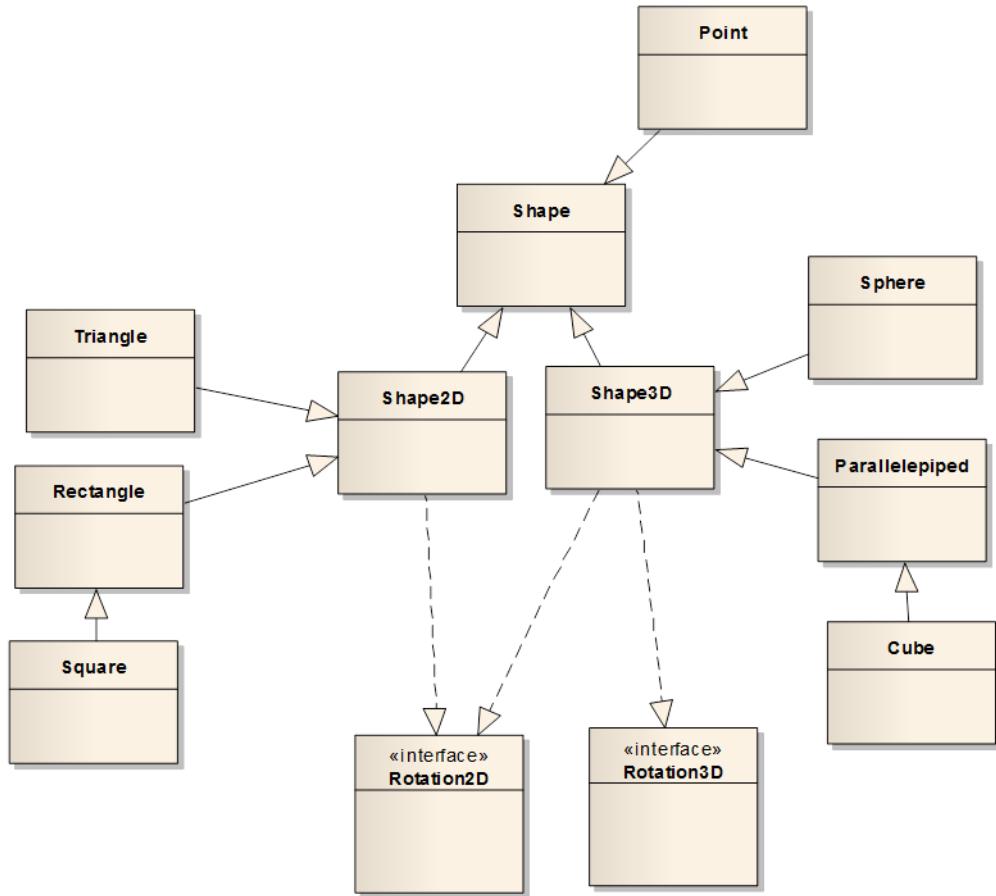
```
// Определение трейта
trait tStorableToDisc
{
    function store() { /* тело метода */ }
    function retrieve() { /* тело метода */ }
}

class cLongConnection extends cNormalConnection
{
    use tStorableToDisc;
    // Полезный код...
}

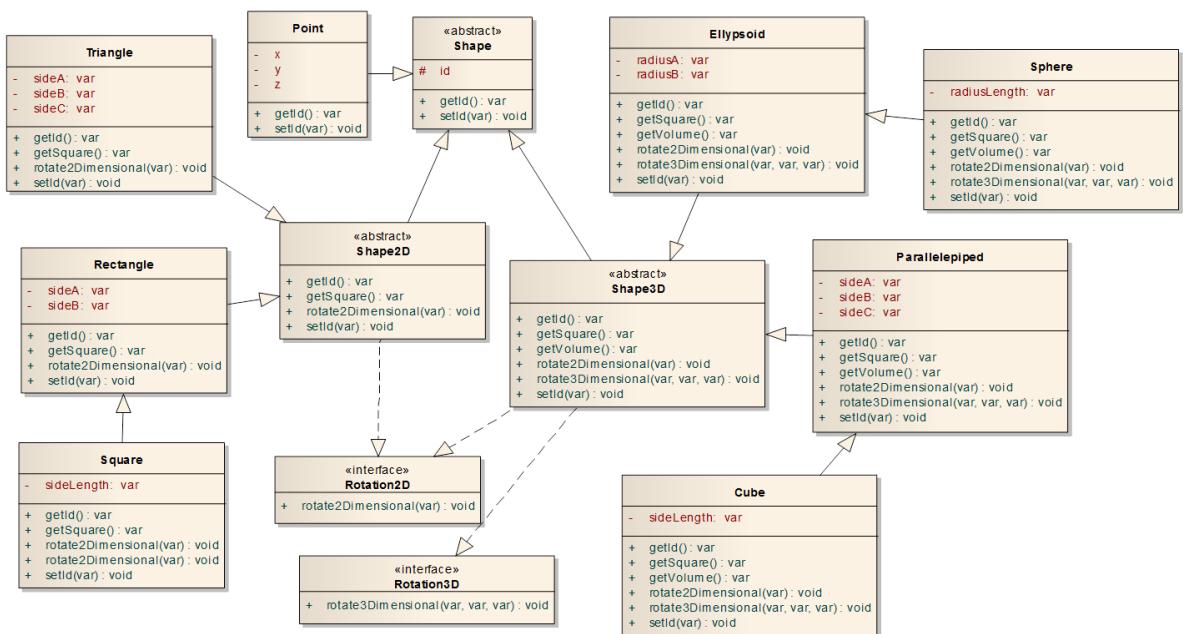
class cShortConnection extends cFastConnection
{
    use tStorableToDisc;
    // Полезный код...
}
```

## Пример использования ООП

Для небольшого закрепления темы про ООП рассмотрим классический пример – геометрические фигуры:



Чуть подробнее:



Теперь остаётся рассмотреть код, реализующий саму иерархию классов и пример его использования.



### См. примеры в папке... 05\_oop/geometry

Если взаимосвязь между представленными в папке исходниками становится неочевидной, обращайтесь к UML-диаграмме, которую мы только что рассмотрели. Здесь (в тексте) приведён лишь исходный код файла, использующего всю подготовленную иерархию классов и интерфейсов.

```
<?php

function __autoload($className)
{
    require './classes/' . $className . '.php';
}

$triangle = new Triangle(3, 4, 5);
$triangle->setId(1);
echo 'Triangle: S = ' . $triangle->getSquare() . "\n";

$rectangle = new Rectangle(3, 4);
$rectangle->setId(2);
echo 'Rectangle: S = ' . $rectangle->getSquare() . "\n";

$square = new Square(5);
$square->setId(3);
echo 'Square: S = ' . $square->getSquare() . "\n";

$ellipsoid = new Ellipsoid(10, 20);
$ellipsoid->setId(4);
echo 'Ellipsoid: S = ' . $ellipsoid->getSquare() . ' ; V = ' . $ellipsoid-
>getVolume() . "\n";

$sphere = new Sphere(15);
$sphere->setId(5);
echo 'Sphere: S = ' . $sphere->getSquare() . ' ; V = ' . $sphere->getVolume()
. "\n";

$parallelepiped = new Parallelepiped(10, 20, 30);
$parallelepiped->setId(6);
echo 'Parallelepiped: S = ' . $parallelepiped->getSquare() . ' ; V = ' . $par-
allelepiped->getVolume() . "\n";

$cube = new Cube(10);
$cube->setId(7);
echo 'Cube: S = ' . $cube->getSquare() . ' ; V = ' . $cube->getVolume() .
"\n";

?>
```

### 3.18. Паттерны проектирования, схема MVC

**Паттерн проектирования (pattern)** – это некое готовое решение, которое можно просто взять и реализовать, концентрируясь только на реализации, но не на изобретении решения (например «приготовление чая» – мы ведь не изобретаем каждый раз чашку, чайник и т.п. – мы знаем, что и как делать).

Следующий материал построен на основе прекрасной статьи:

<http://habrahabr.ru/blogs/programming/136766/>

Также рекомендуется почитать про «антипаттерны» (как делать НЕ надо):

<http://habrahabr.ru/blogs/refactoring/59005/>

Для простоты паттерны делят на виды:

**Порождающие паттерны** (создают новые объекты, или позволяют получить доступ к уже существующим):

- Singleton (одиночка)
- Registry (реестр, журнал записей)
- Multiton (набор «одиночек»)
- Object pool (набор объектов)
- Factory (фабрика)
- Builder (строитель)
- Prototype (прототип)
- Factory method (фабричный метод)
- Lazy initialization (отложенная инициализация)
- Dependency injection (внедрение зависимости)
- Service Locator (локатор служб)

**Структурирующие паттерны** (помогают «навести порядок» и «научить» разные объекты лучше взаимодействовать друг с другом):

- Adapter или wrapper (адаптер, обертка)
- Bridge (мост)
- Composite (компоновщик)
- Decorator (декоратор, оформленитель)
- Facade (фасад)
- Front controller (единая точка входа)
- Flyweight (приспособленец)
- Proxy или surrogate (прокси, заместитель, суррогат)

**Паттерны поведения** (позволяют структурировать подходы к обработке поведения и взаимодействия объектов):

- Chain of responsibility (цепочка обязанностей)
- Command или action (команда, действие)
- Interpreter (интерпретатор)
- Iterator (итератор, указатель)
- Mediator (посредник)
- Memento (хранитель)
- Observer или Listener (наблюдатель, слушатель)
- Blackboard (доска объявлений)
- Servant (слуга)
- State (состояние)
- Strategy (стратегия)
- Specification (спецификация, определение)
- Subsumption (категоризация)

- Visitor (посетитель)
- Single-serving visitor (одноразовый посетитель)
- Hierarchical visitor (иерархический посетитель)

Рассмотрим подробнее...

### **Порождающие паттерны**

Singleton (одиночка)

| Пояснение «на пальцах»   | Пример из программирования  |
|--|---|
| Если нужно подключить телефон нового сотрудника, используется ОДНА уже готовая АТС. Создавать копии этой АТС бессмысленно. | Класс по работе с БД, экземпляр которого активно используется по всей программе. Такой класс может собирать статистику, кэшировать запросы и т.п. |

Registry (реестр, журнал записей)

| Пояснение «на пальцах»                   | Пример из программирования  |
|--|---|
| Телефонная книга. По ФИО получаем номер. | Массив некоторых объектов, элементы которого можно получить по ключу (например, набор «эмуляторов браузеров» для выполнения одного и того же теста под несколькими браузерами). Ещё пример: класс «Сотрудник» может хранить коллекцию «Телефонов» (домашний, рабочий, мобильный). |

Multiton (набор «одиночек»)

| Пояснение «на пальцах»   | Пример из программирования  |
|--|---|
| Аккуратный ящик с инструментами: одни плоскогубцы, один молоток и т.п. | Набор экземпляров классов, каждый из которых поддерживает работу с сетью по своему протоколу: HTTP, SMTP, POP3... |

Object pool (набор объектов)

| Пояснение «на пальцах»                   | Пример из программирования  |
|--|---|
| Поднос с посудой: чашка, пара тарелок... | Многопоточная работа с сетью: SMTP-соединение, несколько HTTP-соединений... |

## Factory (фабрика)

| Пояснение «на пальцах»   | Пример из программирования  |
|--|---|
| Фирма по производству стеклопакетов: вы задаёте параметры и получаете стеклопакет. | Класс, создающий HTTP-соединения с указанным хостом, набором cookies, эмуляцией браузера и т.п.<br><i>Часто реализуется в виде Singleton.</i> |

## Builder (строитель)

| Пояснение «на пальцах»               | Пример из программирования   |
|--------------------------------------|--|
| Художник, рисующий «то, что скажут». | Класс, экспортирующий некоторые данные в указанном формате, в файл с нужным именем и т.п.<br><i>В отличие от Factory содержит внутри себя сложную логику «получения желаемого», а не опирается на внешние готовые решения.</i> |

## Prototype (прототип)

| Пояснение «на пальцах»  | Пример из программирования  |
|---|---|
| Новый сотрудник для низкоквалифицированной работы, быстро обучающийся элементарным операциям. | «Готовый к использованию» эмулятор браузера, который можно «настроить», указав необходимые параметры. |

## Factory method (фабричный метод)

| Пояснение «на пальцах»   | Пример из программирования   |
|--|--|
| Если бы любое животное понимало команду «Голос!», то в ответ на неё от разных животных мы бы слышали «Гав!», «Мяу!», «Ку-ка-ре-ку!», «Чего?» 😊 | Универсальный метод для вывода в виде текста любых данных. Что-то в стиле <code>__toString()</code> .<br><i>Часто является основой для Factory, Builder, Prototype, в которые потом и «перерастает» по мере усложнения логики.</i> |

## Lazy initialization (отложенная инициализация)

| Пояснение «на пальцах»  | Пример из программирования  |
|---|---|
| Бухгалтерия, которая готова выдать справку о доходах таких-то сотрудников. Но пока нет запроса – справки никто заранее готовить не будет. | <p>Есть набор HTTP-соединений, каждое из которых может, если надо, представить свою статистику. Есть некий класс «HTTPConnections», у которого мы можем запросить статистику по всем или отдельным соединениям. Вот когда запросим, тогда он и обратится к конкретным соединениям за статистикой.</p> <p><i>Это позволяет не создавать статистику, пока она не нужна.</i></p> |

## Dependency injection (внедрение зависимости)

| Пояснение «на пальцах»  | Пример из программирования  |
|---|---|
| Служба по доставке в офис питьевой воды. Сами мы воду не возим, мы заказываем эту услугу. | Практически любое обращение к некоторому классу с вызовом его метода, делающим что-то полезное для вызвавшего кода. |

## Service Locator (локатор служб)

| Пояснение «на пальцах»  | Пример из программирования   |
|---|--|
| Операторская служба такси. Мы звоним и заказываем машину, а оператор сам связывается с водителем (в т.ч. из разных «служб такси») и направляет его к нам. | Когда нам нужно выполнить HTTP-запрос, некий объект просматривает набор HTTP-соединений, и передаёт для выполнения запроса свободное и наиболее подходящее (по некоторым параметрам) соединение. Если свободных соединений нет, может создать новое. |

## Структурирующие паттерны

Adapter или wrapper (адаптер, обертка)

| Пояснение «на пальцах»  | Пример из программирования  |
|---|---|
| Палец в розетку просто так не засунешь, а вот если взять гвоздь...<br>☺ | Некий класс возвращает некие данные в 16-тиричной системе счисления. Мы пишем класс, который принимает их и передаёт нам в 10-ричной системе счисления. |

Bridge (мост)

| Пояснение «на пальцах»   | Пример из программирования  |
|--|---|
| Существует множество телефонных аппаратов, но везде можно «сделать вызов», « положить трубку» и т.д. | Есть несколько классов, работающих с сетью (очень по-разному внутри), но все они имплементируют интерфейс с методами connect(), disconnect() и т.п. |

Composite (компоновщик)

| Пояснение «на пальцах»   | Пример из программирования  |
|--|---|
| Люстра с несколькими лампочками, включаемыми одним выключателем. Включили – загорелись все лампочки. | Набор соединений, обслуживающих некую torrent-закачку. Если мы «запускаем закачку», запускаются все соединения. |

Decorator (декоратор, оформленитель)

| Пояснение «на пальцах»   | Пример из программирования   |
|--|--|
| Оформитель подарков в магазине. Вы даёте ему подарок, он возвращает его, завёрнутым в красивую упаковочную бумагу. С бантиком ☺. | Некий класс (или отдельный метод), добавляющий указанные теги к переданному тексту. Например:<br>\$transformation->bold("text")<br>даст на выходе<br>" <b>text</b> " |

Facade (фасад)

| Пояснение «на пальцах»   | Пример из программирования   |
|--|--|
| Кнопка включения питания компьютера. Вы нажимаете всего одну кнопку, а внутри это порождает уйму различных процессов, но результат один – компьютер включился. | Метод, «агрегирующий» в себе множество «сервисных вызовов». Например, достаточно написать get-FileViaHTTP(URL), а внутри: установится соединение, отправится запрос, будут получены данные, данные будут представлены в нужном виде и т.п. |

## Front controller (единая точка входа)

| Пояснение «на пальцах»  | Пример из программирования  |
|---|---|
| Продавец в окошке киоска. Мы называем товар, даём деньги, а он сам там внутри что-то ищет и отдаёт товар (и сдачу, если повезёт ☺). | Унифицированная процедура выполнения некоторого большого количества действий (в т.ч. выполняемых разными объектами). Например: <code>getDataFromStream(Stream)</code> – и нас не волнует, что это за поток, как там всё работает и т.п. Нам просто будут возвращены данные. |

## Flyweight (приспособленец)

| Пояснение «на пальцах»       | Пример из программирования       |
|------------------------------|----------------------------------|
| Актёр, играющий разные роли. | Настраиваемый эмулятор браузера. |

## Proxy или surrogate (прокси, заместитель, суррогат)

| Пояснение «на пальцах»   | Пример из программирования  |
|--|---|
| Офисный работник, идущий в магазин, которого все просят что-то купить – чая, булочек, мороженого, водки и т.д. | Класс, обслуживающий все потребности программы в некотором сетевом соединении (например, класс, работающий с базой данных). |

## Паттерны поведения

Chain of responsibility (цепочка обязанностей)

| Пояснение «на пальцах»  | Пример из программирования  |
|---|---|
| Вы хотите уехать на такси с вокзала и перебираете по очереди все стоящие машины в поисках цены, хотя бы в 2 раза больше реальной, а не в 5, как просит большинство. | Есть набор сетевых соединений. Нам нужно послать запрос. Его для нас пошлёт то соединение, которое сейчас свободно. Для определения свободного соединения нужно по очереди опросить их все, пока не найдём искомое. |

Command или action (команда, действие)

| Пояснение «на пальцах»           | Пример из программирования   |
|----------------------------------|--|
| Записка в офисе «Полейте цветы». | Класс, порождающий в системе некоторое событие (выставляющий некий семафор). Кто и как на это событие среагирует, данный класс не знает. |

Interpreter (интерпретатор)

| Пояснение «на пальцах»   | Пример из программирования  |
|--|---|
| «В пятницу гуляем, как обычно» (как это, где и т.п. знают все, к кому это относится; пояснить подробности им не надо). | <ol style="list-style-type: none"> <li>1) Всевозможные макросы.</li> <li>2) Новый класс, предоставляющий элементарный интерфейс для выполнения часто повторяющихся сложных операций.</li> </ol> |

Iterator (итератор, указатель)

| Пояснение «на пальцах»  | Пример из программирования  |
|---|---|
| Звонок в мебельный: «А какие у вас есть диваны? [перечисление] А кресла? [перечисление] А столы? [перечисление] А... Что вы мне хамите!? <возник экспешн> ☺». | <p>Iterator ☺.</p> <p>См. <a href="http://php.net/manual/en/class.iterator.php">http://php.net/manual/en/class.iterator.php</a></p> |

## Mediator (посредник)

| Пояснение «на пальцах»   | Пример из программирования  |
|--|---|
| Прораб на стройке, получающий задание от начальства, и доносящий его до подчинённых. | Объект, управляющий некоторым набором других объектов. Например «НаборСоединений» может управлять набором «Соединений» и выполнять команды в стиле «закрыть все», «закрыть все неактивные» и т.п. |

## Memento (хранитель)

| Пояснение «на пальцах»  | Пример из программирования  |
|---|---|
| Листик со списком дел, написанный с вечера, чтобы утром ничего не забыть. | Некое HTTP-соединение понимает, что нужно «перезапуститься», т.е. быть уничтоженным и созданным заново. Перед этим оно сохраняет свои настройки в объекте, управляющем такими соединениями. |

## Observer или Listener (наблюдатель, слушатель)

| Пояснение «на пальцах»      | Пример из программирования  |
|-----------------------------|---|
| Оператор, ожидающий звонка. | <ol style="list-style-type: none"> <li>1) Всевозможные Listener'ы, реагирующие на возникновение того или иного события.</li> <li>2) Часть HTTP-сервера, «слушающая» 80-й порт.</li> <li>3) Callback-функции.</li> </ol> |

## Blackboard (доска объявлений)

| Пояснение «на пальцах»   | Пример из программирования   |
|--|--|
| Доска объявлений, как бы банально это ни звучало: кто-то вешает объявления, кто-то читает, кто-то срывает ☺. | Объект, позволяющий другим объектам размещать и читать информацию. Например, при управлении набором HTTP-соединений управляющий класс будет содержать очередь задачий для управляемых им соединений. |

## Servant (слуга)

| Пояснение «на пальцах»                                   | Пример из программирования  |
|--|---|
| Консультант, информирующий посетителей об услугах фирмы. | Объект, информирующий любое HTTP-соединение о загруженности канала связи. |

## State (состояние)

| Пояснение «на пальцах»   | Пример из программирования  |
|--|---|
| В зависимости от того, сыт человек или голоден, он по-разному будет реагировать на предложение «пойти перекусить». | Различная реакция объекта на одну и ту же команду в зависимости от его состояния. Так, например, HTTP-соединение на <code>getData()</code> может отдать данные (если они есть в буфере), отдать «пустоту» (если данных нет), сгенерировать исключение (если соединение не установлено). |

## Strategy (стратегия)

| Пояснение «на пальцах»  | Пример из программирования  |
|---|---|
| Сдача экзамена: можно выучить, можно списать, можно нельзя давать взяток ☺. | В зависимости от нагрузки на систему некоторый запрос обрабатывается сразу или ставится в очередь. Возможно, результат берётся из кэша, если он там есть. |

## Specification (спецификация, определение)

| Пояснение «на пальцах»  | Пример из программирования  |
|---|---|
| Нам передают разные продукты. Если попадается фрукт, мы его чистим. | Если размер файла превышает некоторый заданный, файл архивируется, если нет – отправляется в незаархивированном виде. |

## Subsumption (категоризация)

| Пояснение «на пальцах»                         | Пример из программирования   |
|--|--|
| Чай в кружку, варенье в блюдечку, суп в миску. | 1) Перегрузка методов.<br>2) Передача данных в зависимости от их формата и типа в тот или иной приёмник. |

## Visitor (посетитель)

| Пояснение «на пальцах»   | Пример из программирования  |
|--|---|
| Охранник, обходящий вечером все помещения и проверяющий, закрыты ли двери. | Класс, шифрующий переданные данные (любые). Т.о. можно зашифровать всё содержимое какой-то коллекции. |

## Single-serving visitor (одноразовый посетитель)

| Пояснение «на пальцах»  | Пример из программирования  |
|---|---|
| Одноразовая (не аккумуляторная) батарейка. Купили, использовали в каком-то устройстве, выбросили. | Создали объект, который шифрует данные, зашифровали данные, удалили объект. |

Hierarchical visitor (иерархический посетитель)

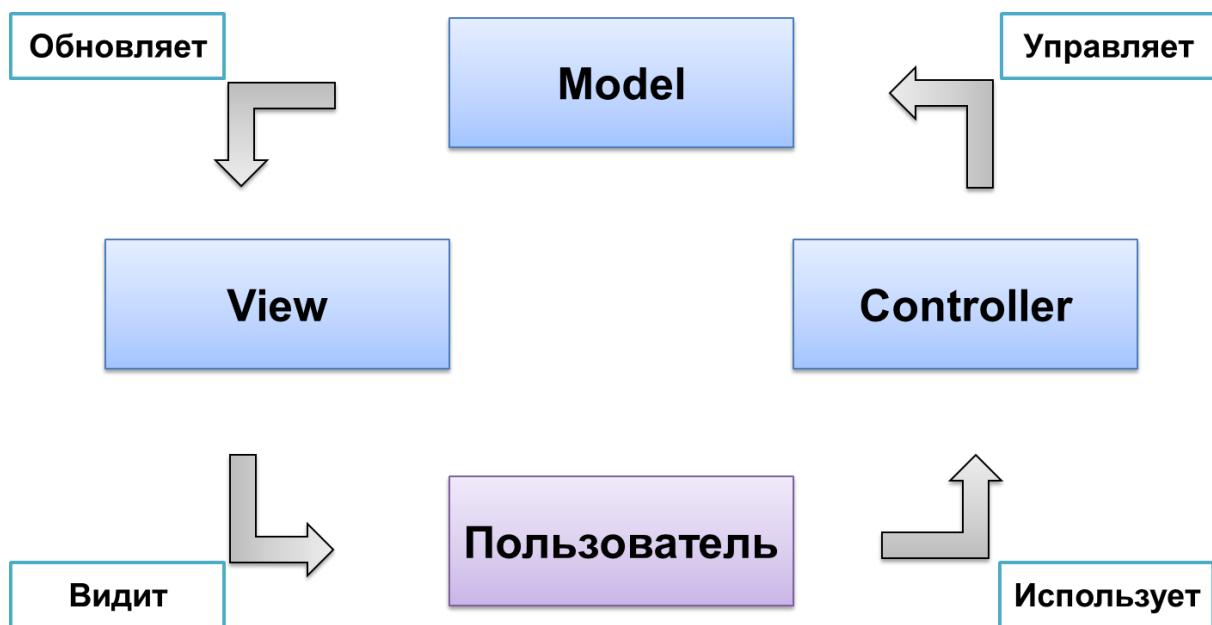
| Пояснение «на пальцах»   | Пример из программирования  |
|--|---|
| Дотошный покупатель, читающий состав каждого из стоящих на полке йогуртов. | Класс, инкрементирующий каждое число в коллекции, на некоторое значение (например, поднимаем цены в интернет-магазине). |

## Схема MVC (Model-View-Controller)

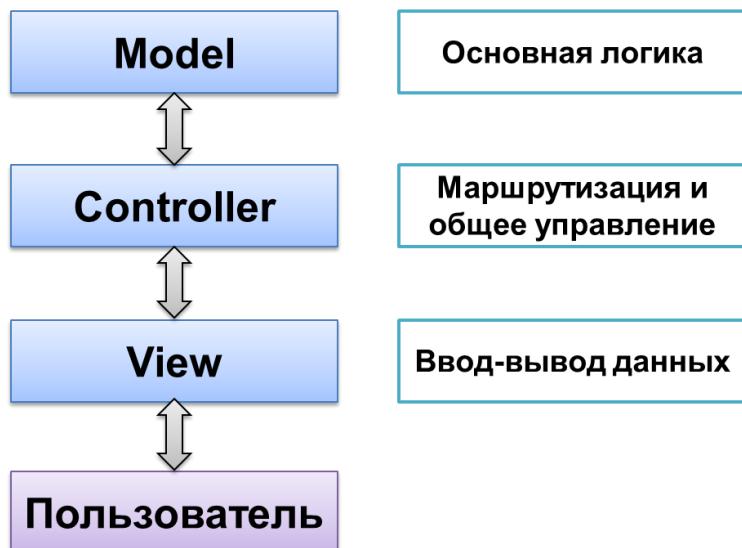
Model View Controller (MVC) – схема построения архитектуры приложения с использованием нескольких шаблонов проектирования, в которой явно выделяются три отдельных компонента:

- Модель (model) реагирует на запросы, изменяя своё состояние и не содержит техник визуализации.
- Представление (view) отвечает за визуализацию (пользовательский интерфейс).
- Контроллер (controller) обрабатывает ввод данных и использует модель и представление для реализации необходимой реакции.

Классическая схема MVC выглядит так:



Более реалистично это выглядит так:



Совершенно тривиальную реализацию MVC сейчас рассмотрим на примере простого кода.

Также рекомендуется ознакомиться:

<http://www.sitepoint.com/the-mvc-pattern-and-php-1/>  
<http://www.sitepoint.com/the-mvc-pattern-and-php-2/>



См. примеры в папке...

05\_oop/mvc

**mvc.php**

```
<?php

require_once('../classes/Model.php');
require_once('../classes/View.php');
require_once('../classes/Controller.php');

$model = new Model();
$view = new View();
$controller = new Controller();

switch ($controller->detectState())
{
    case Controller::STATE_INITIAL:
        echo $view->getForm();
        break;
    case Controller::STATE_POSTED:
        $data = $view->extractData($_POST);
        $operation = $model->detectOperation($data['operation']);
        if ($operation == '?') {
            echo $view->getForm($data['a'], $data['b'], $data['operation'], $data['result']);
        }
        else
        {
            switch ($operation)
            {
                case '+': $result = $model->performSum($data['a'], $data['b']); break;
                case '-': $result = $model->performSub($data['a'], $data['b']); break;
                case '*': $result = $model->performMul($data['a'], $data['b']); break;
                case '/': $result = $model->performDiv($data['a'], $data['b']); break;
            }
            echo $view->getForm($data['a'], $data['b'], $data['operation'], $result);
        }
        break;
}
?>
```

**Model.php**

```
<?php

class Model
{
    public function detectOperation($operation)
    {
        switch ($operation)
        {
            case '+': return '+'; break;
            case '-': return '-'; break;
            case '*': return '*'; break;
            case '/': return '/'; break;
            default: return '?';
        }
    }

    public function performSum($a, $b)
    {
        return $a+$b;
    }

    public function performSub($a, $b)
    {
        return $a-$b;
    }

    public function performMul($a, $b)
    {
        return $a*$b;
    }

    public function performDiv($a, $b)
    {
        return $a/$b;
    }
}

?>
```

**View.php**

```
<?php

// WARNING! HTML is hardcoded here just to make the code shorter and easier!
class View
{
    public function getForm($a = '', $b = '', $operation = '', $result = '')
    {
        $final_text = '<form action="'.$_SERVER['PHP_SELF'].'" method="post">';
        $final_text .= 'A = <input type="text" name="a" value="'.$a.'" /><br />';
        $final_text .= 'B = <input type="text" name="b" value="'.$b.'" /><br />';
        $final_text .= 'Operation <input type="text" name="operation" value="'.$operation.'" /><br />';
        $final_text .= 'Result: '.$result.'<br />';
        $final_text .= '<input type="submit" value="Perform" />';
        $final_text .= '</form>';
        return $final_text;
    }
}
```

```
public function extractData($source)
{
    $data['a'] = '';
    $data['b'] = '';
    $data['operation'] = '?';
    $data['result'] = '';

    if ((isset($source['a']))&&(strlen($source['a'])>0)) {
        $data['a'] = (double) ($source['a']);
    }

    if ((isset($source['b']))&&(strlen($source['b'])>0)) {
        $data['b'] = (double) ($source['b']);
    }

    if (isset($source['operation'])) {
        $data['operation'] = substr(trim((string) ($source['operation'])), 0, 1);
    }

    return $data;
}

?>
```

### Controller.php

```
<?php

class Controller
{
    const STATE_INITIAL = 1;
    const STATE_POSTED = 2;
    const STATE_ERROR = 4;

    public function detectState()
    {
        if (isset($_POST['operation'])) {
            return self::STATE_POSTED;
        }
        else {
            return self::STATE_INITIAL;
        }
    }
}

?>
```

### 3.19. Регулярные выражения в PHP

**Регулярные выражения** (regular expressions) – механизм обработки текста на основе гибких шаблонов поиска и замены.

У регулярных выражений очень долгая и нетривиальная история, но мы сконцентрируемся на основах того, как регулярные выражения реализованы в PHP.



См. примеры в папке...

06\_regregs

В основе регулярных выражений лежат т.н. «метасимволы» (т.е. «символы, описывающие другие символы»).

«Сами по себе» это «бэк-слеш» и «d».

\d

Но в контексте регулярных выражений эта последовательность обозначает любую цифру, т.е. «0123456789».

Хотя «научно» такой вариант называется «экранирующей последовательностью», а «классическими метасимволами» считаются одиночные символы, например «\*», «?» и т.д.

С точки зрения PHP регулярное выражение – это строка, соответствующая определённому набору правил. Первое из таких правил – само регулярное выражение должно быть ограничено специальными ограничителями (НЕ буквенно-цифровыми символами, НЕ бэк-слешем, НЕ пробельным символом). Чаще всего используются:

- / (слеш);
- # (хэш, «решётка»);
- ~ (тильда).

Ограничитель.

Ограничитель.

Регулярное выражение.

Модификатор.

К «классическим» (одиночным) метасимволам в PHP относятся:

- \ – экранирующий символ, допускающий несколько вариантов применения;
  - ^ – начало данных (или строки);
  - \$ – конец данных (или строки);
  - . – любой символ, кроме \n (по умолчанию);
  - [ – начало символьного класса;
  - ] – конец символьного класса;
  - | – указание альтернативного условия;
  - ( – начало подмаски;
  - ) – конец подмаски;
  - ? – квантификатор {0, 1} (в общем случае);
  - \* – квантификатор {0, };
  - + – квантификатор {1, };
  - { – начало квантификатора;
  - } – конец квантификатора.

Внутри символьного класса используются:

- \ – общий экранирующий символ;
  - ^ – отрицание класса (только в начале класса);
  - - – символьный интервал.

К экранирующим последовательностям в PHP относятся:

- `\d` – «цифра» (1234567890);
  - `\D` – «НЕ цифра»;
  - `\h` – «горизонтальный пробельный символ» ([ \t\p{Zs}]);
  - `\H` – «НЕ горизонтальный пробельный символ»; ([\v\h]);
  - `\s` – «любой пробельный символ» ([\v\h]);
  - `\S` – «любой НЕпробельный символ»;
  - `\v` – «вертикальный пробельный символ» ([\n\cK\f\r\x85\x{2028}\x{2029}]);
  - `\V` – «НЕ вертикальный пробельный символ»;
  - `\w` – «буквенно-цифровой символ» (`d`, буквы, `_`);
  - `\W` – «НЕ буквенно-цифровой символ»;
  - `\b` – «граница слова»;
  - `\B` – «НЕ граница слова»;
  - `\A` – «начало данных» (независимо от многострочного режима);
  - `\Z` – «конец данных» либо «позиция перед последним переводом строки» (независимо от многострочного режима);
  - `\z` – «конец данных» (независимо от многострочного режима)
  - `\G` – «первая совпадающая позиция в строке»;
  - `\Q` – «включить игнорирование метасимволов»;
  - `\E` – «выключить игнорирование метасимволов»;
  - `\K` – «сбросить указатель начала совпадений».
  - `\R` – «любой перенос строк» (`\n`, `\r`, `\r\n`);
  - `\a` – «символ оповещения» («сигнал») (hex=07);
  - `\cx` – «Ctrl+x», где x – произвольный символ;
  - `\e` – «escape» (hex=1B);
  - `\f` – «разрыв страницы» (hex=0C);
  - `\n` – «перевод строки» (hex=0A);
  - `\p{xx}` – «символ со свойством xx» (подробнее см. [свойства unicode-символов](#));
  - `\P{xx}` – «символ без свойства xx» (подробнее см. [свойства unicode-символов](#));
  - `\r` – «возврат каретки» (hex=0D);
  - `\t` – «табуляция» (hex=09);
  - `\xhh` – «символ с шестнадцатиричным кодом hh»;
  - `\ddd` – «символ с восьмеричным кодом ddd» либо «ссылка на подмаску».

Самое основное из метасимволов и экранирующих последовательностей, что нужно знать сначала:

- \ – экранирующий символ (превращает «обычные символы» в «экранирующие последовательности» (например, \n), а метасимволы в «просто символы» (например, \\*));
  - ^ и \$ – начало и конец данных (или строки);
  - . – любой символ, кроме \n (по умолчанию);
  - [ и ] – начало конец символьного класса;
  - | – указание альтернативного условия;
  - ( и ) – начало и конец подмаски;
  - { и } – начало и конец квантификатора;
  - ? – сокращённый вариант квантификатора {0,1}
  - \* – сокращённый вариант квантификатора {0, };
  - + – сокращённый вариант квантификатора {1, };
  - \d – «цифра» (1234567890);
  - \D – «НЕ цифра»;
  - \s – «любой пробельный символ» ([\v\h]);
  - \S – «любой НЕпробельный символ»;
  - \w – «буквенно-цифровой символ» (\d, буквы, \_);
  - \W – «НЕ буквенно-цифровой символ».
  - \b – «граница слова»;
  - \B – «НЕ граница слова»;
  - \t – табуляция;
  - \r – возврат каретки;
  - \n – перенос строки.

**Символьный класс** (character class) – это набор, которому соответствует ОДИН символ в анализируемом тексте. Символьный класс заключается в [ ] и допускает «^» в начале для «отрицания набора» и «-» для указания диапазонов:

```
[1234567890abcdefABCDEF] → «шестнадцатиричная цифра»
[\da-fA-F] → «шестнадцатиричная цифра» (сокращённо)
[^\\da-fA-F] → «НЕ шестнадцатиричная цифра»
```

**Модификатор** (modifier) – символ, идущий после закрывающего ограничителя регулярного выражения и влияющий на логику работы выражения. Последовательность указания модификаторов не важна. Основные модификаторы:

- i – выключить регистров чувствительность;
- m – включить мультистроковый режим;
- s – добавить \n в список «любых символов»;
- x – игнорировать пробелы, «разрешить комментарии»;
- u – переключиться в UTF8;
- U – выключить «жадность».

Полный список см. [здесь](#).

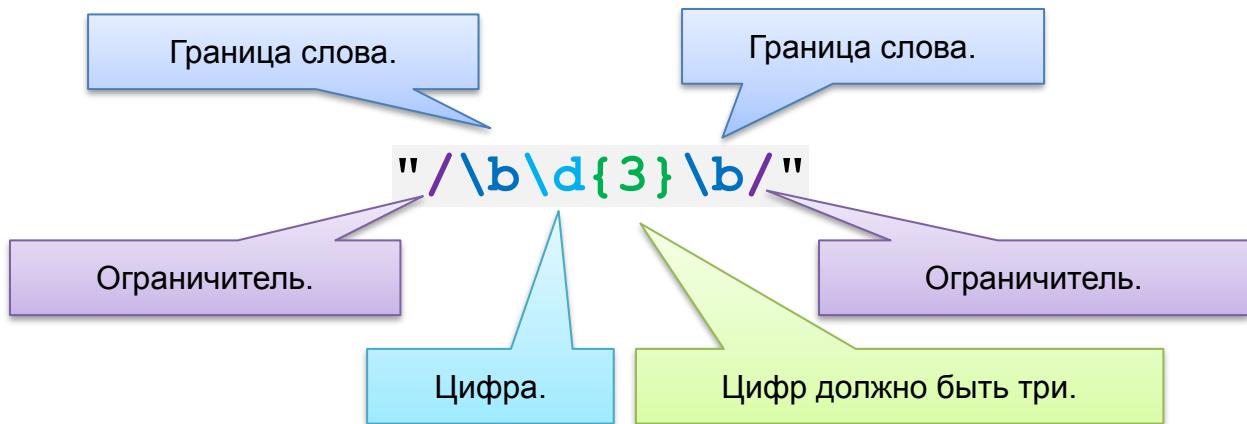
PHP поддерживает несколько вариантов работы с регулярными выражениями, но мы будем ориентироваться на PCRE-совместимый вариант. Основные функции, которые нам понадобятся:

- preg\_match() – определение наличия совпадения;
- preg\_match\_all() – поиск всех совпадений;
- preg\_replace() – замена;
- preg\_replace\_callback() – замена с использованием вызова пользовательской функции.

Полный список см. [здесь](#). Мы рассмотрим использование этих функций сразу на примерах.

Да, это – очень простые примеры, но и наш курс не посвящён регулярным выражениям. Если вам хочется большего -- читайте прекрасную книгу “Mastering Regular Expressions” (Jeffrey Friedl).

**Пример 1:** содержит ли текст хотя бы одно трёхзначное число?

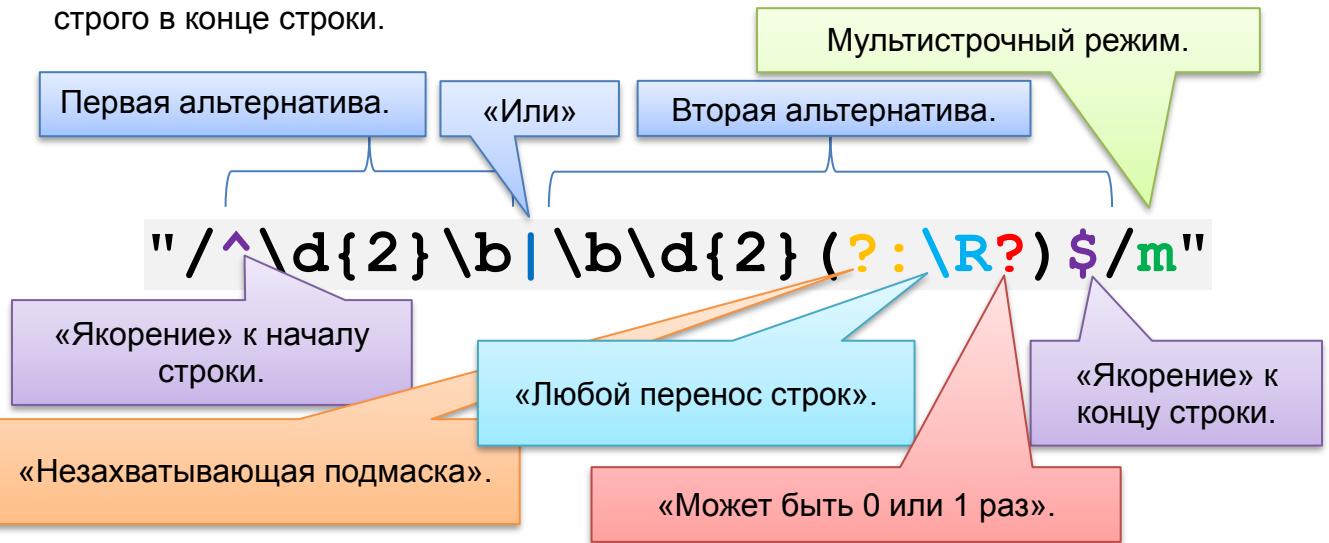


```
$t = 'Это пример текста, а вот и числа: 893, 83, 95624';

// - Неверное решение! В числе 95624 "найдутся" "чила" 956, 562, 624.
if (preg_match("/\d\d\d/", $t))
{
    echo '1.1 = OK';
}

// + Верное решение. Используем экранированные последовательности
// для определения "границ слова" + квантификатор.
if (preg_match("/\b\d{3}\b/", $t))
{
    echo '1.2 = OK';
}
```

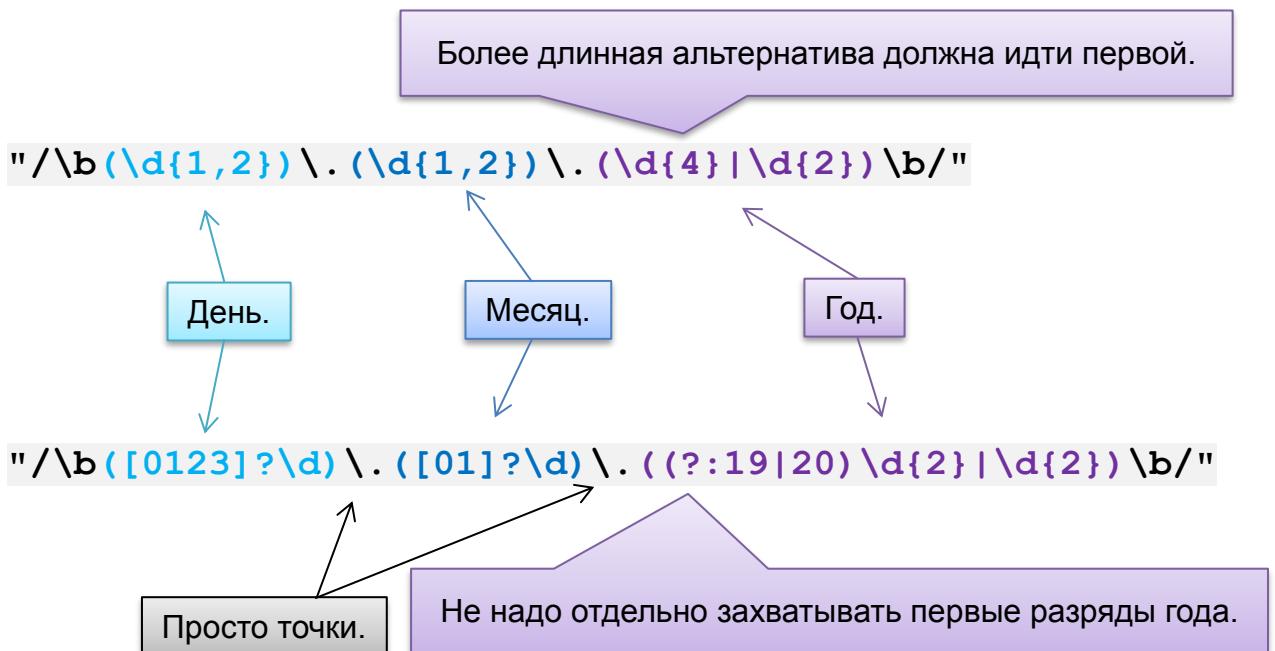
**Пример 2:** показать все двухзначные числа, стоящие строго в начале или строго в конце строки.



```
$t = '99 это будет
многострочный текст 777, в котором 11
22 будут числа 33
444 в самых 44 разных местах.00';
```

```
preg_match_all("/^\d{2}\b|\b\d{2}(?:\R?)$/m", $t, $arr);
print_r($arr);
```

**Пример 3:** для дат в формате [Д]Д.[М].М.[ГГ]ГГ выделить день, месяц, год.



```
$t = 'Это пример текста с датами и чем-то, что только похоже на даты:  
01/01/2000, 1.1.2014, 333.33.44, 11.11.11, 12.2.08, 99.99.9999';  
  
// Первый вариант решения: простой, но реагирует на "даты" вида 99.99.9999  
preg_match_all("/\b(\d{1,2})\.\.(\d{1,2})\.\.(\d{4} | \d{2})\b/", $t, $arr,  
PREG_SET_ORDER);  
print_r($arr);  
  
// Второй вариант решения. Но здесь мы всё равно можем получить  
// "даты" вида 31.02.2005 (т.е. неверные).  
preg_match_all("/\b([0123]?d)\.\.([01]?d)\.\.((?:19|20)\d{2}|\d{2})\b/", $t,  
$arr, PREG_SET_ORDER);  
print_r($arr);  
  
// Третий (оптимальный) вариант решения.  
// Не всегда надо пытаться ВСЁ переложить на регулярные выражения.  
// Некоторые задачи гораздо проще решаются поэтапно.  
preg_match_all("/\b(\d{1,2})\.\.(\d{1,2})\.\.(\d{4} | \d{2})\b/", $t, $arr,  
PREG_SET_ORDER);  
foreach ($arr as $k => $date)  
{  
    if (!checkdate($date[2], $date[1], $date[3]))  
    {  
        unset($arr[$k]);  
    }  
}  
print_r($arr);
```

**Пример 4:** выделить в тексте дублирующиеся слова ([источник](#)).

Поиск:

```
"/(\b[\w']+)\b (\w+) (\b\\1\b)/i"
```

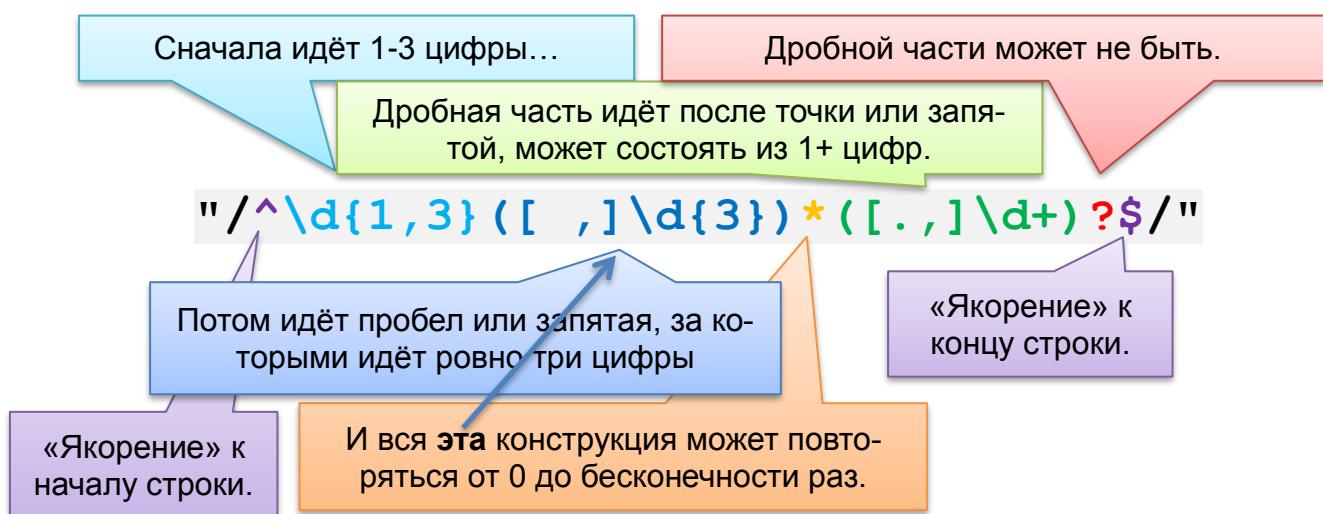
Замена:

```
"\\1\\2<strong>\\3</strong>"
```

Строка замены – это НЕ регулярное выражение. В ней распознаются только «обратные ссылки». В остальном – это **ОБЫЧНЫЙ ТЕКСТ**.

```
$tasks = array
(
    array('given' => 'This is a test', 'expected' => 'This is a test'),
    array('given' => 'This is is a test', 'expected' => 'This is <strong>is</strong> a test'),
    array('given' => 'This test test is is', 'expected' => 'This test <strong>test</strong> is <strong>is</strong>'),
    array('given' => 'This test is a test', 'expected' => 'This test is a test'),
    array('given' => 'This this test is a test', 'expected' => 'This <strong>this</strong> test is a test'),
    array('given' => 'cat dog dog cat dog', 'expected' => 'cat dog <strong>dog</strong> cat dog'),
    array('given' => 'This test is a test tester', 'expected' => 'This test is a test tester'),
    array('given' => 'hello world hello world', 'expected' => 'hello world hello world'),
    array('given' => 'This nottest test is something', 'expected' => 'This nottest test is something'),
    array('given' => 'This is IS a test', 'expected' => 'This is <strong>IS</strong> a test'),
    array('given' => '<Mack> I\'ll I\'ll be back back in in a a bit bit.', 'expected' => '<Mack> I\'ll <strong>I\'ll</strong> be <strong>be</strong> back <strong>back</strong> in <strong>in</strong> a <strong>a</strong> bit <strong>bit</strong>.')
);
foreach ($tasks as $task)
{
    $result = preg_replace("/(\b[\w']+)\b (\w+) (\b\\1\b)/i",
"\\1\\2<strong>\\3</strong>", $task['given']);
    if ($result == $task['expected'])
    {
        echo '+ OK: ['. $task['expected']. "]\n";
    }
    else
    {
        echo '- FAIL: ['. $result. "] instead of [". $task['expected']. "]\n";
    }
}
```

**Пример 5:** определить числа с запятой или пробелом в качестве разделителя разрядов ([источник](#)).



```
$tasks = array
(
    array('given' => '24', 'match' => TRUE),
    array('given' => '1,024', 'match' => TRUE),
    array('given' => '2,000,204', 'match' => TRUE),
    array('given' => '3,000.6', 'match' => TRUE),
    array('given' => '8,205,500.4672', 'match' => TRUE),
    array('given' => '0.5', 'match' => TRUE),
    array('given' => '36,000.57', 'match' => TRUE),
    array('given' => '100,000', 'match' => TRUE),
    array('given' => '5',
        array('given' => '46',
            array('given' => '10.444444444444',
                array('given' => '1 024',
                    array('given' => '9 999 352',
                        array('given' => '10,19836',
                            array('given' => '30 000,7302',
                                array('given' => '0,5',
                                    array('given' => '47 372',
                                        array('given' => '.5',
                                            array('given' => '1025',
                                                array('given' => '1,5826,000',
                                                    array('given' => ',046',
                                                        array('given' => '100.',
                                                            array('given' => '2.2.2',
                                                                array('given' => '10.',
                                                                array('given' => '#123456',
                                                                array('given' => '10,.5',
                                                                array('given' => '507',
                                                                array('given' => '507',
                                                                array('given' => '34 34',
                                                                array('given' => '3692 38',
                                                                array('given' => '36 047.',
                                                                array('given' => '47 .7',
                                                                ),
                );
            foreach ($tasks as $task)
            {
                $result = preg_match("/^\\d{1,3}([ ,]\\d{3})*([,]\\d+)?$/",
                    $task['given'],
                    $arr);
                if ($result == $task['match'])
                {
                    echo '+ OK: [' . $task['given'] . "]\n";
                }
                else
                {
                    echo '- FAIL: [' . $task['given'] . ']' -> [' . @$arr[0] . ']
                    var_dump($result)."\n";
                }
            }
        )
    )
)
```

**Пример 6:** вывести все целые числа жирным, а все дроби – курсивом, предварительно округлив до десятых. Текст может быть ЛЮБЫМ.

```
$t = '7777.88888 И снова какой-то текст с числами 12, 96, 2333, дробями  
34.232, 45.34234, 8921124.900 и "разным": 2012-2014, 11.11.2011. А вот и  
дробь в конце предложения: 333.333. 9999';
```

```
function numbers($x)
{
    $test = trim($x[0], '.');
    echo $test."\n";
    if (preg_match("/^\d+$/", $test))
    {
        return '<strong>' . $test . '</strong>';
    }
    elseif(substr_count($test, '.') == 1)
    {
        return '<i>' . round($test, 1) . '</i>';
    }
    else
    {
        return $x[0];
    }
}

echo preg_replace_callback("/[\d.-]+/", 'numbers', $t);
```

Вся логика различия целых, дробей и «не чисел» находится здесь.

«Поисковый регэксп» очень простой.

**Пример 7:** заменить плейсхолдеры с именами файлов на содержимое файлов. Синтаксис плейсхолдера: {FILE="path"}.

```
$t = 'Это текст, в котором есть плейсхолдеры: {FILE="file1.ext"}  
{FILE="file2.ext"}';

function files($x)
{
    $filename = $x[1];
    if (is_file($filename))
    {
        return file_get_contents($filename);
    }
    else
    {
        return 'File ['. $filename .'] not found!';
    }
}

echo preg_replace_callback("/{FILE=\"(.+)\\"}/imsU", 'files', $t);
```

## 3.20. Обработка форм на PHP

К настоящему моменту мы рассмотрели всё необходимое для понимания принципов обработки форм на PHP. Остаётся лишь свести это в единый алгоритм.

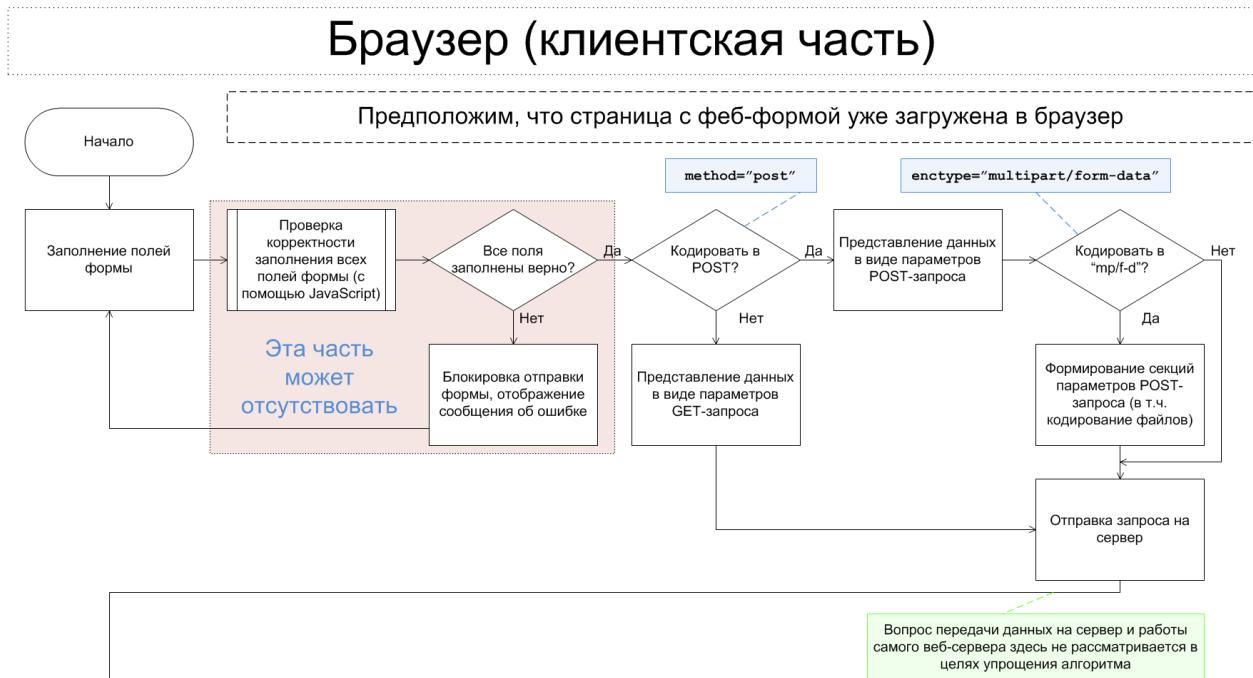


**См. алгоритмы и примеры в папке...**

### 07\_forms

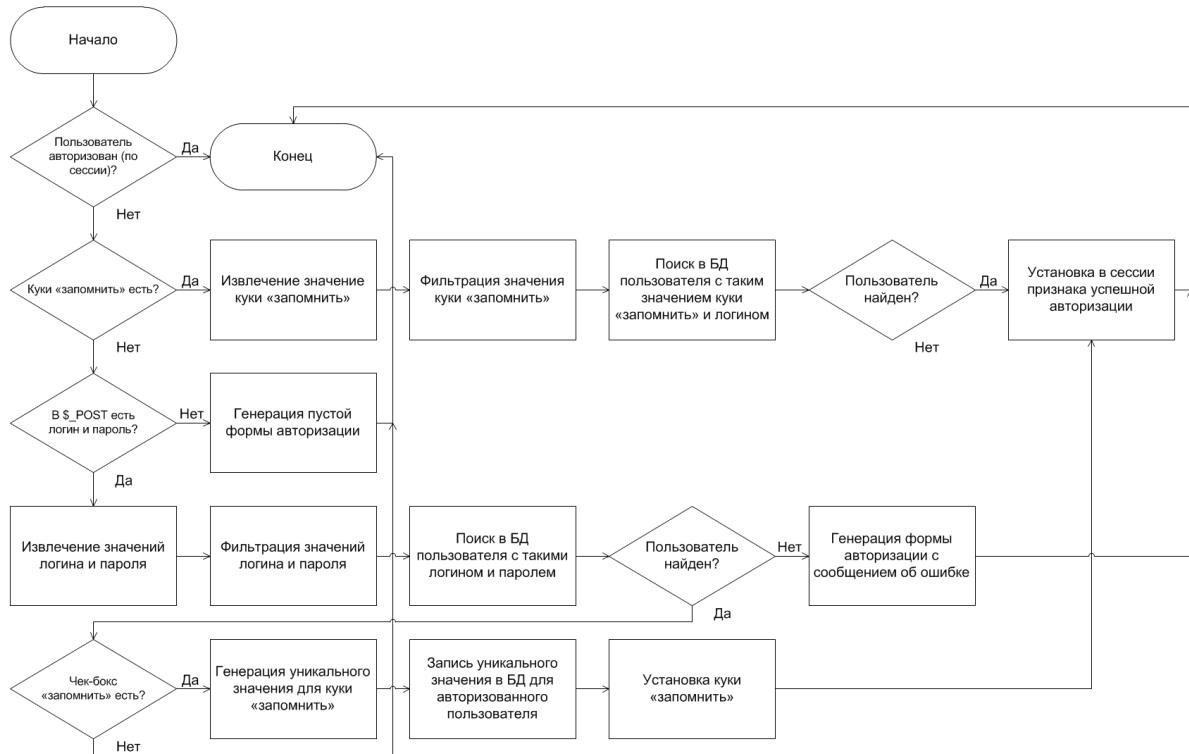
(Полноразмерная весия картинок с алгоритмами:  
Работа веб-форм.png и Работа веб-приложения.png)

Пример обработки веб-формы:



## Пример авторизации:

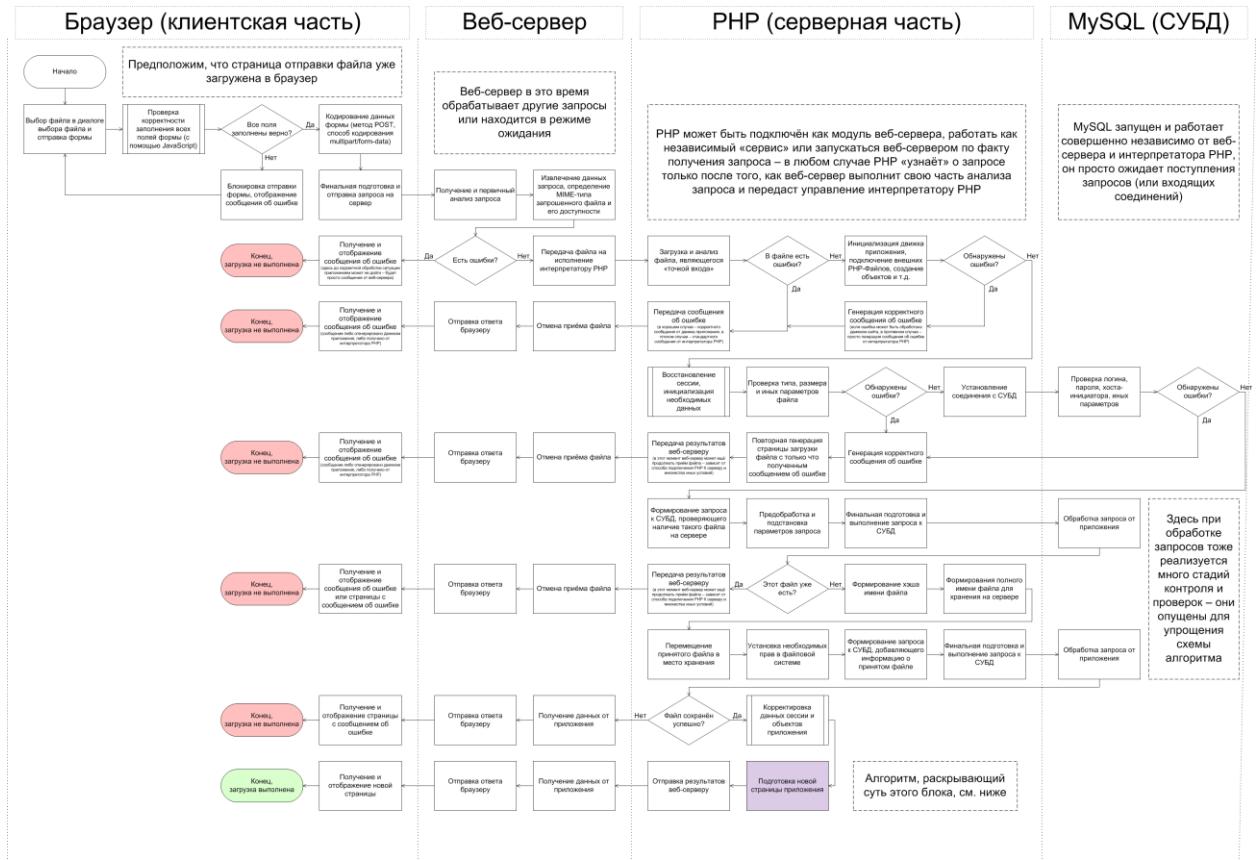
## PHP (серверная часть)



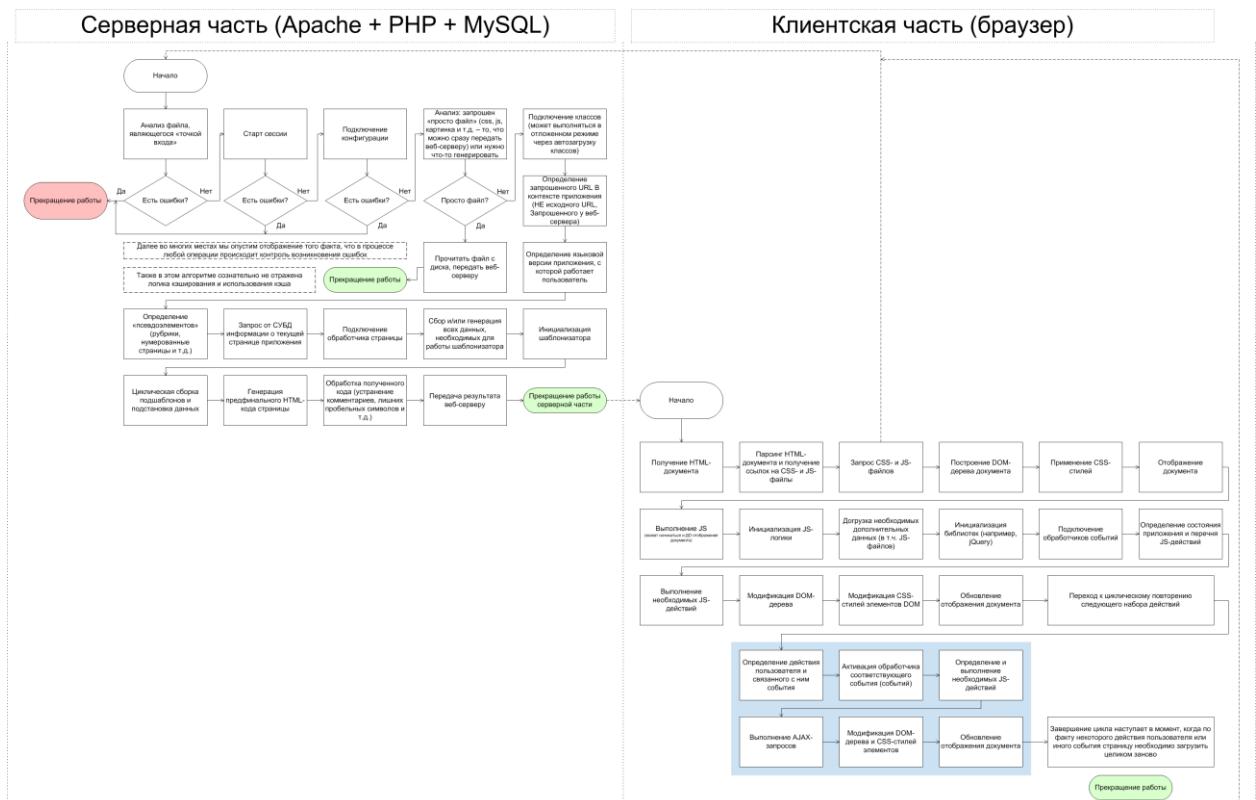
Самой распространённой (и очень опасной) ошибкой является отсутствие фильтрации данных, пришедших со стороны пользователя (это относится не только к формам, но к любым данным из массивов `$_POST`, `$_GET`, `$_COOKIE`, `$_FILES`).

Также для дополнительного закрепления понимания принципов работы форм пересмотрите обобщённые алгоритмы сохранения файла и генерации страницы.

## Работа веб-приложения на примере сохранения файла на сервере:



## Генерация страницы приложения (логика сервера и клиента):



### 3.21. Стандарты PSR

**PSR (PHP Specification Request)** – набор стандартов, созданных с целью выработки единого стиля программирования на PHP и повышения совместимости PHP-кода, созданного разными авторами.

Подробности: <http://petermoulding.com/php/psr>

С оригиналом PSR можно ознакомиться здесь: <http://www.php-fig.org/psr/>

Несмотря на то, что изучение стандартов – дело неблагодарное, PHP-сообщество так долго ждало этих спецификаций, а сами они весьма невелики, мы рассмотрим их целиком.



**См. PSR в виде отдельного документа в папке...**

08\_psr

Если же совсем кратко, то суть PSR в следующем:

- **PSR-0** (Autoloading Standard, стандарт автозагрузки) – определяет правила именования классов и реализации поведения автозагрузки классов.
  - **PSR-1** (Basic Coding Standard, базовый стандарт оформления кода) – описывает основные правила оформления кода.
  - **PSR-2** (Coding Style Guide, рекомендации по оформлению кода) – даёт расширенные пояснения по оформлению кода.
  - **PSR-3** (Logger Interface, интерфейс протоколирования) – описывает интерфейс библиотек протоколирования.

## 3.22. Библиотека PEAR, формат phar

### Общие сведения

К настоящему моменту мы рассмотрели множество способов решения различных задач на PHP. Однако многие из них уже успешно решены, и эти решения представлены в библиотеке...

**PEAR (PHP Extension and Application Repository)** – библиотека классов PHP, в которой представлены решения многих типичных повседневных задач.

Также стоит упомянуть PECL (PHP Extension Community Library) – репозиторий модулей для PHP, написанных на С и доступных через систему пакетов PEAR. PECL был создан как ответ на проблему удаления отдельных модулей из стандартной поставки PHP. В последнее время много нареканий вызывает то, что этот репозиторий слабо обновляется.

PEAR включает порядка 600 пакетов (<http://pear.php.net/packages.php>) «на все случаи жизни».

В качестве примера рассмотрим классическую задачу «сумма прописью».

### Установка PEAR

Подробности по установке PEAR можно найти здесь:

<http://pear.php.net/manual/en/installation.getting.php>

Скачаем: <http://pear.php.net/go-pear.phar> и поместим его в подкаталог pear в каталоге, в котором установлен PHP (например: c:/php/pear).

Выполним: `php go-pear.phar` от имени администратора.

На вопрос о том, `system` ИЛИ `local` версию вы ставите, выбирайте `system`. После завершения установки выполните файл `PEAR_ENV.reg`.

Всё, PEAR готова к работе.

Теперь выполним `pear install Numbers_Words` для установки готового решения по получению суммы прописью.

Если вы получили сообщение об ошибке вида «*Failed to download pear/Numbers\_Words within preferred state "stable", latest release is version 0.16.4, stability "beta", use "channel://pear.php.net/Numbers\_Words-0.16.4" to install. install failed*», выполните команду (с учётом своего номера версии)

```
pear install channel://pear.php.net/Numbers_Words-0.16.4
```

Как правило, установка проходит успешно.

Но если мы внимательно почитаем секцию dependencies в документации, мы увидим, что ещё нужно установить `Math_Integer`. Сделаем это. Хотя может так оказаться, что у вас этот модуль уже установлен или инсталлятор сам его установил. Но проверить не помешает.

## Пример использования PEAR



## См. примеры в папке... 09\_pear\_phar

Теперь осталось написать совсем чуть-чуть кода и выполнить его:

<?php

```
require_once('Numbers/Words/lang.ru.php');
$source_number = 784513245178;
$transformer = new Numbers_Words_ru();
echo $final_string = $transformer->toWords($source_number);

?>
```

Результат: «семьсот восемьдесят четыре миллиарда пятьсот тринадцать миллионов двести сорок пять тысяч сто семьдесят восемь»

## Формат PHAR

В процессе установки PEAR мы использовали команду `php go-pear.phar`, пришло время посмотреть, что такое «.phar».

**PHAR (PHP Archive)** – специальный формат, предназначенный для распространения и выполнения PHP-файлов в виде единого архива.

Про PHAR написано много (<http://php.net/manual/en/intro.phar.php>; <http://www.sitepoint.com/packaging-your-apps-with-phar/>), но всю суть выражает простой пример.

Этот код создаёт исполняемый phar-архив с только что рассмотренным приложением «сумма прописью»:



См. примеры в папке...

### 09\_pear\_phar

```
<?php

// Если не работает, проверьте php.ini на предмет верной настройки:
// phar.readonly = Off
if (Phar::canWrite())
{
    $p = new Phar('phar_creator_result.phar', 0, 'phar_creator_result.phar');
    $p = $p->convertToExecutable(Phar::PHAR);

    // Запуск подготовки архива
    $p->startBuffering();

    // Так можно сразу упаковать целую папку
    // $p->buildFromIterator(new RecursiveIteratorIterator(new DirectoryIterator('./phar_source_files')), './');

    // Так можно добавлять файлы по одному
    $p['pear_example.php'] = file_get_contents('pear_example.php');

    // Указываем точку входа
    $p->setStub($p->createDefaultStub('pear_example.php'));

    // Завершаем подготовку архива и записываем его на диск
    $p->stopBuffering();
}
?>
```

Теперь приложение можно выполнять прямо из phar-файла:

```
php phar_creator_result.phar
```

И ещё один пример: как «положить в phar» целое приложение. Сначала – структура приложения:

```
/app/
-> index.php
-> images/
    -> pic1.png
    -> pic2.png
-> html/
    -> html1.html
-> css/
    -> css1.css
```

Само «приложение» примитивно до невозможности:

### index.php

```
<?php
    echo file_get_contents('./html/html1.html');
?>
```

### html1.html

```
<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" href=".css/css1.css">
</head>
<body>
    Hello!
    <br />
    
</body>
</html>
```

### css1.css

```
body {background-color:lightgrey; font-size:15pt; color:red}
```

И две картинки:

Just an image... :)

Another image... 

Упакуем приложение в phar таким кодом:

```
<?php

// Если не работает, проверьте php.ini на предмет верной настройки:
// phar.readonly = Off
if (Phar::canWrite())
{
    $p = new Phar('app.phar');
    $p->convertToExecutable(Phar::PHAR);

    // Запуск подготовки архива
    $p->startBuffering();

    // Упаковываем всё приложение
    // Создаём итератор по файловой системе
    // (второй параметр ОЧЕНЬ ВАЖЕН, без него может вообще не работать!)
    $recDirIt = new RecursiveDirectoryIterator('./app/', FilesystemIterator::SKIP_DOTS);

    // Создаём итератор по итератору :) (да, так надо)
    $recItIt = new RecursiveIteratorIterator($recDirIt);

    // Наполняем наш phar файлами на основе содержимого папки проекта
    $p->buildFromIterator($recItIt, './app/');

    // Так можно добавлять файлы по одному
    //$p['file.php'] = file_get_contents('file.php');

    // Указываем точку входа
    $p->setStub($p->createDefaultStub('index.php'));

    // Завершаем подготовку архива и записываем его на диск
    $p->stopBuffering();
}
?>
```

Остаётся подправить httpd.conf (конфигурацию Apache):

```
AddType application/x-httpd-php .php .phar
... положить phar-файл в DocumentRoot и набрать в браузере
http://127.0.0.1/app.phar
```

Да-да, все картинки, CSS-ки и т.д. «подхватываются» прямо из phar-файла сами и без нашего участия.



### 3.23. PHPDoc

**PHPDoc** – адаптация системы генерации документации Javadoc для использования с PHP.

PHPDoc поддерживает как объектно-ориентированный, так и процедурный код в документах и имеет простой и достаточно хорошо структурированный внутренний синтаксис.

Одним из широко используемых инструментальных средств для генерации документации на основе PHPDoc является `phpDocumentor` (<http://www.phpdoc.org/>).

**ВАЖНО!** С русским языком у `phpDocumentor` есть проблемы, т.е. или гуглите решение (оно есть), или пишите PHPDoc на английском.

Подробное описание PHPDoc является достаточно объёмным (пусть и очень простым): <http://www.phpdoc.org/docs/latest/references/phpdoc/index.html>

Мы же рассмотрим общую концепцию на примере.

**ВАЖНО!** Обязательно ознакомьтесь с полным описанием хотя бы для того, чтобы знать, какие теги поддерживает PHPDoc.

#### Пример использования PHPDoc



См. примеры в папке...

10\_phpdoc

Краткое описание метода

```
/**
 * Top-level constructor to initialize DBMS-connection parameters
 *
 * Important! Do not forget to call from descendant classes!
 *
 * @param string $server      host name
 * @param string $login       DBMS user login
 * @param string $password    DBMS user password
 * @param string $database   database name
 */
protected function __construct($server, $login, $password, $database)
{
    $this->server = $server;
    $this->login = $login;
    $this->password = $password;
    $this->database = $database;
}
```

Подробное описание метода

Описание параметров

И после обработки получается...

\_\_construct()

```
__construct(string $server, string $login, string $password, string $database)
```

Top-level constructor to initialize DBMS-connection parameters  
Important! Do not forget to call from descendant classes!

Parameters

|                   |  |
|-------------------|--|
| string \$server   | host name or ip-address of DBMS server |
| string \$login    | DBMS user login                        |
| string \$password | DBMS user password                     |
| string \$database | database name                          |

## Вместо постскриптума к теме

Иногда можно услышать вопросы в стиле: «А можно ли на PHP сделать... нуу... что-нибудь ЭТАКОЕ?» Можно. Не пытайтесь повторить это на работе:



См. примеры в папке...

### 11\_madness

```
// Безумие. Часть 1.
// 1) У нас есть функция.
function fnc($x)
{
    return $x*$x;
}

// 2) Её имя мы кладём в переменную.
$a = 'fnc';

// 3) Имя этой переменной мы кладём в другую переменную.
$b = 'a';

// 4) А имя этой другой переменной -- в третью.
$c = 'b';

// 5) А теперь мы можем вызвать нашу функцию так. Да, это работает.
echo $$$c(4);

// Безумие. Часть 2.
// 1) Объявим класс. БЕЗ методов.
class Madness
{
    // 2) На самом деле методы будут, но они будут храниться в массиве.
    protected $methods = array();

    // 3) Это т.н. magic method, который срабатывает,
    // когда на объекте вызвали
    // несуществующий метод (помните, изначально у нас не было
    // никаких методов, т.к. они будут храниться в массиве).
    function __call($method, $args)
    {
        // 4) Какой бы метод класса ни был вызван, он
        // всегда не существует, т.к. хранится в массиве,
        // а не объявлен явно. Проверяем, есть ли в массиве искомое.
        if (isset($this->methods[$method]))
        {
            // 5) Если вызванный несуществующий метод есть в массиве,
            // вызываем его и возвращаем результат его работы.
            // Исходные аргументы передаём при вызове.
            return $this->methods[$method]($args);
        }
        else
        {
            // 6) Если вызванный несуществующий метод на самом
            // деле не существует, т.е. его нет в массиве, мы
            // создаём его "на лету", используя
            // механизм замыканий.
            echo "Just for debug. New method [\".$method.\"] created.\n";
            $this->methods[$method] = function($args) use ($method)
            {
                return 'Call of [\".$method.\"] with arguments: '.print_r($args, TRUE);
            };
        }
    }
}
```

```
// 7) И сразу же вызываем новосозданный метод,  
// передавая ему параметры, с которыми он был  
// вызван, и возвращая результаты его работы.  
return $this->methods[$method]($args);  
}  
}  
  
// 8) Да начнётся безумие!  
$madness = new Madness();  
  
// 9) Заведём несколько строковых переменных, в которых будем хранить  
// имена методов, которые в будущем мы захотим вызвать.  
$realname1 = 'fnc1';  
$realname2 = 'fnc2';  
$realname3 = 'fnc3';  
  
// 10) Чтобы было веселее, имена этих переменных сохраним  
// в других переменных.  
$name1 = 'realname1';  
$name2 = 'realname2';  
$name3 = 'realname3';  
  
// 11) А теперь из переменных, хранящих имена переменных, создадим массив.  
$functions = array($name1, $name2, $name3);  
  
// 12) Пройдём по этому массиву...  
foreach ($functions as $function)  
{  
    // 13) Здесь, используя двойное разыменование,  
    // будем вызывать у нашего объекта несуществующие методы, которые  
    // будут на лету создаваться и выполняться.  
    echo $madness->$$function($function);  
  
    // 14) Второй вызов будет приводить к срабатыванию  
    // только что созданных (в шаге 13) методов.  
    echo $madness->$$function($function, 'AGAIN!');  
}
```