МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

**«Дальневосточный федеральный университет»**

(ДВФУ)

**ШКОЛА ЕСТЕСТВЕННЫХ НАУК**

«СОГЛАСОВАНО»
Руководитель ОП Разработка программно-информационных систем по направлению 09.04.04 Программная инженерия

_____ Артемьева И.Л.
(подпись) (Ф.И.О. рук. ОП)
«21» 07 2018 г.

«УТВЕРЖДАЮ»
Заведующая кафедрой прикладной математики, механики, управления и программного обеспечения

_____ Артемьева И.Л.
(подпись) (Ф.И.О. зав. каф.)
«21» 07 2018 г.

**РАБОЧАЯ ПРОГРАММА УЧЕБНОЙ ДИСЦИПЛИНЫ**

Инженерия интернет систем

**Направление подготовки – 09.04.04 Программная инженерия**

Магистерская программа «Разработка программно-информационных систем»

**Форма подготовки (очная)**

курс 1 семестр 2
лекции 0 час.
практические занятия 0 час.
лабораторные работы 36 час.
в том числе с использованием МАО лек. 0 / пр. 0/ лаб. 18 час.
в том числе в электронной форме лек._____/пр._____/лаб._____ час.
всего часов аудиторной нагрузки – 72 час.
в том числе с использованием МАО – 18 час.
в том числе контролируемая самостоятельная работа 0 час.
в том числе в электронной форме _____ час.
самостоятельная работа 72 час.
в том числе на подготовку к экзамену _____ час.
курсовая работа / курсовой проект не предусмотрено
зачет 2 семестр
экзамен не предусмотрено

Рабочая программа составлена в соответствии с требованиями образовательного стандарта, самостоятельно устанавливаемого ДВФУ, утвержденного приказом ректора от 07.07.2015 № 12-13-1282

Рабочая программа обсуждена на заседании кафедры прикладной математики, механики, управления и программного обеспечения, протокол № 7.2 от 21.07.2018 г.

Заведующая кафедрой прикладной математики, механики, управления и программного обеспечения Артемьева И.Л., д.т.н., профессор

Составитель: ассистент кафедры прикладной математики, механики, управления и программного обеспечения Лось Р.П., веб-разработчик ООО Амаяма Авто

**Оборотная сторона титульного листа РПУД**

**I. Рабочая программа пересмотрена на заседании кафедры**:

Протокол от «_____» _____ 20__ г. № _____

Заведующий кафедрой _____ _____

(подпись) (И.О. Фамилия)

**II. Рабочая программа пересмотрена на заседании кафедры**:

Протокол от «_____» _____ 20__ г. № _____

Заведующий кафедрой _____ _____

(подпись) (И.О. Фамилия)

# ABSTRACT

**Master's degree in** 09.04.04 – Software engineering

**Master's Program "**Development of software and information systems**"**

**Course title:** Internet systems engineering

**Basic (variable) part of Block, _3_credits**

**Instructor:** Los R.

**At the beginning of the course a student should be able to:** analyze the problems and the development trends of the technology of programming; apply main methods and tools of the automation of the design and the evaluation tests of software in a professional activity; know about the main stages of the design of program and mathematical software, and information technologies; know about the modern technologies of programming; understand computer equipment development trends and assess the relevance of software

**Learning outcomes:** an ability to independently get knowledge and new abilities with the help of information technologies and use acquired skills in practice including new fields of knowledge which are not related to the sphere of activity; an ability to create network protocol services; possession of methods and tools of receiving, storage, processing and broadcasting of information by means of modern computer technologies including global computer networks; possession of the methods of optimization and an ability to use it at solving problems of professional activity; an ability to design distributed information systems, its components and communication protocols

**Course description:** professional application of modern internet technologies in order to create internet applications for solving various professional problems from various subject domains; safety and reliability of internet applications

**Main course literature:**

1. Ajvaliotis, D. Administrirovanie servera NGINX [Management of Nginx server] / D. Ajvaliotis. — Moskva : DMK Press, 2015. — 288 s. (rus) — Access: http://lib.dvfu.ru:8080/lib/item?id=Lan:Lan-63190&theme=FEFU

2. Praktikum po administrirovaniyu programmnogo obespecheniya: laboratornyj praktikum [Management of Software / Practicing] / — Stavropol': Severo-Kavkazskij federal'nyj universitet, 2017.— 85 c. (rus) — Access: http://lib.dvfu.ru:8080/lib/item?id=IPRbooks:IPRbooks-75589&theme=FEFU

3. Dzhosh, L. Sovremennyj PHP. Novye vozmozhnosti i peredovoj opyt [Modern PHP. New possibilities and advanced experience] / L. Dzhosh ; per. s angl.

R.N. Ragimov. — Moskva : DMK Press, 2016. — 304 s. (rus) — Access: http://lib.dvfu.ru:8080/lib/item?id=Lan:Lan-93269&theme=FEFU

4. N. Prohorenok. HTML, JavaScript, PHP i MySQL. Dzhentl'menskij nabor Web-Mastera [HTML, JavaScript, PHP and MySQL. Web-master gentleman set] / N. Prohorenok. - SPb. : BHV-Peterburg, 2010.— 912 s. (rus) — Access: http://lib.dvfu.ru:8080/lib/item?id=chamo:380700&theme=FEFU

5. D. N. Kolisnichenko. Linux. Ot novichka k professionalu [Linux. From beginner to Professional] / D. N. Kolisnichenko. . - Sankt-Peterburg : BHV-Peterburg, 2010. - 784 s. (rus) — Access: http://lib.dvfu.ru:8080/lib/item?id=chamo:828765&theme=FEFU

**Form of final knowledge control:** pass-fail exam.

**Аннотация рабочей программы учебной дисциплины**
**«Инженерия интернет систем»**

Рабочая программа дисциплины «Инженерия интернет систем» разработана для студентов 1 курса, обучающихся по направлению 09.04.04 Программная инженерия, профиль «Разработка программно-информационных систем». Дисциплина является дисциплиной выбора вариативной части учебного плана Б1.В.ДВ.02.01.

Трудоемкость дисциплины 3 зачетных единицы (108 часов). Дисциплина реализуется во 2 семестре. Учебным планом предусмотрено: 36 часов лабораторных работ (из них 18 в интерактивной форме), 72 часа самостоятельной работы.

Дисциплина «Инженерия интернет систем» базируется на дисциплинах бакалавриата, связанных с изучением методов создания программных систем. Знания, полученные при ее изучении, будут применяться на производственных практиках, для выполнения научно-исследовательской работы и магистерской диссертации.

**Цель** дисциплины – обучение студентов профессионально применять имеющиеся современные Интернет-технологии с целью создания интернет приложений для решения различных профессиональных задач для различных предметных областей, а также приобретение навыков обеспечения безопасности и надежности работы Интернет-приложений.

**Задачи дисциплины:**

1. Изучить основные Интернет-технологии, тенденции их развития и применение в различных предметных областях;
1. Сформировать навыки эффективного использования Интернет-ресурсов в профессиональной деятельности;
2. Научить проектировать информационные Интернет системы, их компоненты и протоколы их взаимодействия.

Для успешного изучения дисциплины «Инженерия интернет систем» у обучающихся должны быть сформированы следующие предварительные компетенции: готовность анализировать проблемы и направления развития технологий программирования, способность применять в профессиональной деятельности основные методы и средства автоматизации проектирования, испытаний и оценки качества программного обеспечения, знать содержание основных этапов разработки программного, математического обеспечения и информационных технологий; знать современные технологии программирования; знать направление развития компьютерной техники; знать тенденции развития и актуальность программного обеспечения.

Планируемые результаты обучения по данной дисциплине (знания, умения, владения), соотнесенные с планируемыми результатами освоения образовательной программы, характеризуют этапы формирования следующих компетенций (общекультурные/ общепрофессиональные/ профессиональные компетенции (элементы компетенций)):

| **Код и формулировка компетенции** | **Этапы формирования компетенции** | |
|---|---|---|
| ПК-3 знание методов оптимизации и умением применять их при решении задач профессиональной деятельности | Знает | Современные методы оптимизации процесса разработки программного обеспечения |
| | Умеет | Применять методы оптимизации при решении задач профессиональной деятельности |
| | Владеет | Приёмами анализа и разработки Интернет-приложений для использования их в различных предметных областях |
| ПК-8 способность проектировать распределенные информационные системы, их компоненты и протоколы их взаимодействия | Знает | Основные компоненты и протоколы, используемые в Интернет-технологиях |
| | Умеет | проектировать Интернет-приложения |
| | Владеет | Инструментами и способами использования современных Интернет-технологий при создании распределенных приложений |
| ПК15 способность проектировать программное обеспечение, имеющее встроенные средства адаптации к изменяемым условиям эксплуатации | Знает | Основные компоненты и протоколы, используемые в Интернет-технологиях |
| | Умеет | Программировать интернет приложения |
| | Владеет | Инструментами и способами использования современных Интернет-технологий при создании распределенных приложений |

Для формирования вышеуказанных компетенций в рамках дисциплины «Инженерия интернет систем» применяются следующие методы активного/ интерактивного обучения: семинары, проектный метод и деловая игра.

## I.СТРУКТУРА И СОДЕРЖАНИЕ ТЕОРЕТИЧЕСКОЙ ЧАСТИ КУРСА

Не предусмотрено.

## II. СТРУКТУРА И СОДЕРЖАНИЕ ПРАКТИЧЕСКОЙ ЧАСТИ КУРСА

### Лабораторные работы (36 час.)

**Лабораторная работа №1. Основные команды для работы в Bash (4 часа).**

Работа на удалённом компьютере по протоколу SSH. Утилиты для работы с файлами и каталогами. Утилиты для работы с текстом.

**Лабораторная работа №2. Установка и настройка веб-сервера (4 часа).**

Основы компьютерных сетей. Модели работы веб-серверов. Установка дополнительных утилит/библиотек с помощью менеджера пакетов apt. Установка и настройка веб-сервера Apache2. Установка и настройка веб-сервера Nginx.

**Лабораторная работа №3. Установка настройка PHP-Fpm в связке с Nginx. (2 часа)**

Основные способы межпроцессорного взаимодействия (unix-сокеты, net-сокеты). Утилиты для работы с процессами, сокетами. Сервисы в Linux. Создание простейшего скрипта на языке PHP.

**Лабораторная работа №4. Технология программирования PHP (4 часа).**

Использование PHP для создания простых скриптов. Работа с файлами с помощью PHP.

**Лабораторная работа №5. Система управления базами данных MySql (4 часа).**

Реляционные базы данных. Установка и настройка MySql-сервера. Работа с MySql с помощью консольного клиента. Работа с MySql базой данных из PHP-скрипта.

**Лабораторная работа №6. Создание статичной Web-страницы (4 часа).**

Язык гипертекстовой разметки HTML. Основные тэги и атрибуты HTML. Работа с гиперссылками.

**Лабораторная работа №7. Каскадные таблицы стилей (2 часа).**

Использование CSS в оформлении web-страниц.

**Лабораторная работа №8. Создание динамически формируемой HTML-страницы с использованием языка PHP (6 часов).**

Протокол HTTP. Передача параметров между клиентом и сервером. Формы в HTML.

**Лабораторная работа №9. Скрипты в HTML-документах (6 часов).**

Использование Java Script при создании web-сайта.

## Ш. УЧЕБНО-МЕТОДИЧЕСКОЕ ОБЕСПЕЧЕНИЕ САМОСТОЯТЕЛЬНОЙ РАБОТЫ ОБУЧАЮЩИХСЯ

Трудоемкость самостоятельной работы 36 часов и 36 часов контролируемой самостоятельной работы. Учебно-методическое обеспечение самостоятельной работы обучающихся по дисциплине «Инженерия интернет систем» представлено в Приложении 1 и включает в себя:

план-график выполнения самостоятельной работы по дисциплине, в том числе примерные нормы времени на выполнение по каждому заданию;

характеристика заданий для самостоятельной работы обучающихся и методические рекомендации по их выполнению;

требования к представлению и оформлению результатов самостоятельной работы;

критерии оценки выполнения самостоятельной работы.

## IV. КОНТРОЛЬ ДОСТИЖЕНИЯ ЦЕЛЕЙ КУРСА

| № п/п | Контролируемые разделы/темы дисциплины | Коды и этапы формирования компетенций | | Оценочные средства – наименование | |
|---|---|---|---|---|---|
| | | | | текущий контроль | промежуточная аттестация |
| 1 | Лабораторная работа №1. Основные команды для работы в Bash | ПК-3<br>ПК-8<br>ПК-15 | Знания | Опрос УО-1 | Вопросы к зачету № 2, 4, 9 |
| | | | Умения | Л/работа ПР-6 | |
| | | | Владения | С/работа ПР-11 | |
| 2 | Лабораторная работа №2. Установка и настройка веб-сервера | ПК-3<br>ПК-8<br>ПК-15 | Знания | Опрос УО-1 | Вопросы к зачету № 1, 3, 5, 6, 22 |
| | | | Умения | Л/работа ПР-6 | |
| | | | Владения | С/работа | |

| | | | | | |
|---|---|---|---|---|---|
| | | | | ПР-11 | |
| 3 | Лабораторная работа №3. Установка настройка PHP-Fpm в связке с Nginx. | ПК-3<br>ПК-8<br>ПК-15 | Знания | Опрос УО-1 | Вопросы к зачету № 6-12 |
| | | | Умения | Л/работа ПР-6 | |
| | | | Владения | С/работа ПР-11 | |
| 4 | Лабораторная работа №4. Технология программирования PHP | ПК-3<br>ПК-8<br>ПК-15 | Знания | Опрос УО-1 | Вопросы к зачету № 11-13 |
| | | | Умения | Л/работа ПР-6 | |
| | | | Владения | С/работа ПР-11 | |
| 5 | Лабораторная работа №5. Система управления базами данных MySql | ПК-3<br>ПК-8<br>ПК-15 | Знания | Опрос УО-1 | Вопросы к зачету № 14-17 |
| | | | Умения | Л/работа ПР-6 | |
| | | | Владения | С/работа ПР-11 | |
| 6 | Лабораторная работа №6. Создание статичной Web-страницы | ПК-3<br>ПК-8<br>ПК-15 | Знания | Опрос УО-1 | Вопросы к зачету № 18-20 |
| | | | Умения | Л/работа ПР-6 | |
| | | | Владения | С/работа ПР-11 | |
| 7 | Лабораторная работа №7. Каскадные таблицы стилей | ПК-3<br>ПК-8<br>ПК-15 | Знания | Опрос УО-1 | Вопросы к зачету № 21 |
| | | | Умения | Л/работа ПР-6 | |
| | | | Владения | С/работа ПР-11 | |
| 8 | Лабораторная работа №8. Создание динамически формируемой HTML-страницы с использованием языка PHP | ПК-3<br>ПК-8<br>ПК-15 | Знания | Опрос УО-1 | Вопросы к зачету № 22-24 |
| | | | Умения | Л/работа ПР-6 | |
| | | | Владения | С/работа ПР-11 | |
| 9 | Лабораторная работа №9. Скрипты в HTML-документах | ПК-3<br>ПК-8<br>ПК-15 | Знания | Опрос УО-1 | Вопросы к зачету № 25, 26 |
| | | | Умения | Л/работа ПР-6 | |

Типовые контрольные задания, методические материалы, определяющие процедуры оценивания знаний, умений и навыков и (или) опыта деятельности, а также критерии и показатели, необходимые для оценки знаний, умений, навыков и характеризующие этапы формирования компетенций в процессе освоения образовательной программы, представлены в Приложении 2.

## V. СПИСОК УЧЕБНОЙ ЛИТЕРАТУРЫ И ИНФОРМАЦИОННО-МЕТОДИЧЕСКОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ

### Основная литература

*(электронные и печатные издания)*


1. Айвалиотис, Д. Администрирование сервера NGINX [Электронный ресурс] / Д. Айвалиотис. — Электрон. дан. — Москва: ДМК Пресс, 2015. — 288 с. — Режим доступа: http://lib.dvfu.ru:8080/lib/item?id=Lan:Lan-63190&theme=FEFU
2. Практикум по администрированию программного обеспечения [Электронный ресурс]: лабораторный практикум — Электрон. текстовые данные.— Ставрополь: Северо-Кавказский федеральный университет, 2017.— 85 с. — Режим доступа: http://lib.dvfu.ru:8080/lib/item?id=IPRbooks:IPRbooks-75589&theme=FEFU
3. Джош, Л. Современный PHP. Новые возможности и передовой опыт [Электронный ресурс] / Л. Джош; пер. с англ. Р.Н. Рагимов. — Электрон. дан. — Москва: ДМК Пресс, 2016. — 304 с. — Режим доступа: http://lib.dvfu.ru:8080/lib/item?id=Lan:Lan-93269&theme=FEFU
4. Н. Прохоренок. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-Мастера [Электронный ресурс]: научная / Н. Прохоренок. - Электрон. текстовые данные.— СПб.: БХВ-Петербург, 2010.— 912 с. — Режим доступа: http://lib.dvfu.ru:8080/lib/item?id=chamo:380700&theme=FEFU
5. Д. Н. Колисниченко. Linux. От новичка к профессионалу [Электронный ресурс]: научная / Д. Н. Колисниченко. - Электрон. текстовые данные. - Санкт-Петербург: БХВ-Петербург, 2010. - 784 с. — Режим доступа: http://lib.dvfu.ru:8080/lib/item?id=chamo:828765&theme=FEFU


### Дополнительная литература

*(печатные и электронные издания)*


1. Молчанова Л.А., Прудникова Л.И. Java в примерах и задачах: учеб.-метод. пособие [для вузов]. Владивосток: Изд-во Тихоокеанского экономического университета. – 2011. Режим доступа: http://lib.dvfu.ru:8080/lib/item?id=chamo:359168&theme=FEFU
2. Баженова И.Ю. Языки программирования: учебник для высшего профессионального образования. Под редакцией В.А. Сухомлина. М.: Академия. – 2012. 358 С. Режим доступа: http://lib.dvfu.ru:8080/lib/item?id=chamo:668317&theme=FEFU

3. Кристофер Шмитт. CSS. Рецепты программирования (3-е издание), 2011

4. Мациевский Н. Разгони свой сайт. Методы клиентской оптимизации веб-страниц: Учебное пособие.- Интернет университет Информационных технологий БИНОМ. Лаборатория знаний. 2009. – 264 с. http://lib.dvfu.ru:8080/lib/item?id=chamo:277610&theme=FEFU

5. Тахагхогхи С., Вильямс Хью Е. Руководство по MySQL. - Издательство «Русская редакция», 2007. — 544 стр. http://lib.dvfu.ru:8080/lib/item?id=chamo:665602&theme=FEFU

**Перечень ресурсов информационно-телекоммуникационной сети «Интернет»**

1. https://pythonworld.ru/web Python для Web
2. http://window.edu.ru/resource/792/23792 Java-программирование интернет-приложений. Программа дисциплины. - М.: МГУ, 2004.
3. http://window.edu.ru/resource/031/76031 Тузовский А.Ф. Проектирование Интернет приложений: учебно-методическое пособие / А.Ф. Тузовский; Томский политехнический университет. - Томск: Изд-во Томского политехнического университета, 2010. - 200 с
4. http://linux.vt.tpu.ru/lab1.pdf Практическое знакомство с операционной системой UNIX
5. https://www.digitalocean.com/community/tutorials/ Учебники
6. http://www.kolpinkurs.ru/saiti/lbcss.htm Лабораторные работы по CSS

**Перечень информационных технологий и программного обеспечения**

Среда программирования web-приложений на языках PHP, HTML и JavaScript.

SSH-клиент. Файловый менеджер с поддержкой SFTP.

## VI. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ОСВОЕНИЮ ДИСЦИПЛИНЫ

Студенты выполняют лабораторные работы, оформляют отчет в электронной форме и представляют получившийся у них результат. Проводится защита работы каждого студента с ответами на вопросы преподавателя.

Текущий контроль предусматривает посещение студентом занятий, подготовку к самостоятельным и лабораторным работам, участие в семинарских занятиях (подготовка презентаций, ответы). Перед студентами ставится задача подготовки информационно-содержательного наполнения сайта, разработки его структуры и программирование кода Интернет-ресурса с помощью

специальных программных средств либо с помощью текстового редактора Блокнот.

В процессе подготовки к занятиям студенту следует обобщить литературные данные, сделать анализ источников информации. Для более успешного выполнения заданий студенту необходимо ознакомиться с содержанием рабочей программы, изучить соответствующие разделы программы курса и литературные источники, рекомендуемые к данному курсу, а также использовать ресурсы сети Интернет.

Самостоятельная работа студентов по дисциплине проводится в форме деловой игры с коллективным обсуждением результатов.

Итогом изучения дисциплины является защита готового проекта Интернет-ресурса. Оценка производится на основе оригинальности содержательной части ресурса, дизайна и умении оптимального использования конструкций языка HTML, PHP при написании кода сайта.

## VII. МАТЕРИАЛЬНО-ТЕХНИЧЕСКОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ

Лабораторные занятия проходят в аудиториях, оборудованных компьютерами типа Lenovo C360G-i34164G500UDK с лицензионными программами Microsoft Office 2013 и аудиовизуальными средствами проектор Panasonic DLPProjectorPT-D2110XE, плазма LG FLATRON M4716CCBAM4716CJ. Для выполнения самостоятельной работы студенты в жилых корпусах ДВФУ обеспечены Wi-Fi.

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

**«Дальневосточный федеральный университет»**

(ДВФУ)

**ШКОЛА ЕСТЕСТВЕННЫХ НАУК**

# УЧЕБНО-МЕТОДИЧЕСКОЕ ОБЕСПЕЧЕНИЕ САМОСТОЯТЕЛЬНОЙ РАБОТЫ ОБУЧАЮЩИХСЯ

по дисциплине «Инженерия интернет систем»

**Направление подготовки – 09.04.04 Программная инженерия**

Магистерская программа «Разработка программно-информационных систем»

**Форма подготовки (очная)**

**Владивосток**
**2018**

План-график выполнения самостоятельной работы по дисциплине

| № п/п | Дата/сроки выполнения | Вид самостоятельной работы | Примерные нормы времени на выполнение | Форма контроля |
|---|---|---|---|---|
| 1 | 1-4 неделя | Ознакомиться с моделью OSI. Изучить основы протокола HTTP | 16 часов | собеседование |
| 2 | 5-8 неделя | Сообщение на тему: «Популярные PHP-фреймворки: Symfony, Laravel, Yii» | 16 часов | собеседование |
| 3 | 9-12 неделя | Деловая игра по теме: «Лучшая система защиты информации в сети Интернет» | 16 часов | собеседование |
| 4 | 13-15неделя | Доклад на тему: Уровни кеширования в клиент-серверных приложениях | 12 часов | собеседование |
| 5 | 16-18 неделя | Веб-сайт с RESTful API | 12 часов | Проект |
| | | ИТОГО | 72 часа | |

## Рекомендации по самостоятельной работе студентов

Учебным планом предусмотрено 72 часа самостоятельной работы, из них 36 часов контролируемая самостоятельная работа.

Самостоятельная работа обучающихся подразумевает обязательную подготовку к лабораторным занятиям (оформление отчетов), изучение основной и дополнительно литературы по дисциплине, подготовку к текущему контролю и промежуточной аттестации в конце семестра, консультации преподавателей

## Рекомендации по работе с литературой

Для более эффективного освоения и усвоения материала рекомендуется ознакомиться с теоретическим материалом по той или иной теме до проведения лабораторного занятия. Всю учебную литературу желательно изучать «под конспект».

Цель написания конспекта по дисциплине – сформировать навыки по поиску, отбору, анализу и формулированию учебного материала.

Работу с теоретическим материалом по теме можно проводить по следующей схеме:

- название темы;
- цели и задачи изучения темы;
- основные вопросы темы;

- характеристика основных понятий и определений, необходимых для усвоения данной темы;

- краткие выводы, ориентирующие на определенную совокупность сведений, основных идей, ключевых положений, систему доказательств, которые необходимо усвоить.

При работе над конспектом обязательно выявляются и отмечаются трудные для самостоятельного изучения вопросы, с которыми уместно обратиться к преподавателю при посещении консультаций, либо в индивидуальном порядке.

## Подготовка презентации и доклада

Для подготовки презентации рекомендуется использовать: PowerPoint, MS Word, Acrobat Reader, LaTeX-овский пакет bпорядeamer. Последовательность подготовки презентации:

1. Четко сформулировать цель презентации: вы хотите свою аудиторию мотивировать, убедить, заразить какой-то идеей или просто формально отчитаться.

2. Определить каков будет формат презентации: живое выступление (тогда, сколько будет его продолжительность) или электронная рассылка (каков будет контекст презентации).

3. Отобрать всю содержательную часть для презентации и выстроить логическую цепочку представления.

4. Определить ключевые моменты в содержании текста и выделить их.

5. Определить виды визуализации (иллюстрации, образы, диаграммы, таблицы) для отображения их на слайдах в соответствии с логикой, целью и спецификой материала.

6. Подобрать дизайн и форматировать слайды (количество картинок и текста, их расположение, цвет и размер).

7. Проверить визуальное восприятие презентации.

***Практические советы по подготовке презентации*** - готовьте отдельно:

- печатный текст + слайды + раздаточный материал;
- *слайды* – визуальная подача информации, которая должна содержать минимум текста, максимум изображений, несущих смысловую нагрузку, выглядеть наглядно и просто;

- *текстовое содержание презентации* – устная речь или чтение, которая должна включать аргументы, факты, доказательства и эмоции;
- *рекомендуемое число слайдов* 17-22;
- *обязательная информация для презентации*: тема, фамилия и инициалы выступающего; план сообщения; краткие выводы из всего сказанного; список использованных источников;
- *раздаточный мате*риал – должен обеспечивать ту же глубину и охват, что и живое выступление: люди больше доверяют тому, что они могут унести с собой, чем исчезающим изображениям, слова и слайды забываются, а раздаточный материал остается постоянным осязаемым напоминанием; раздаточный материал важно раздавать в конце презентации; раздаточный материалы должны отличаться от слайдов, должны быть более информативными.

## Методические указания к подготовке к практическим и лабораторным работам

Подготовку к лабораторной работе студент должен начать с изучения теоретического материала и ознакомления с планом, который отражает содержание предложенной темы. Все новые понятия по изучаемой теме необходимо выучить наизусть и внести в глоссарий, который целесообразно вести с самого начала изучения курса. Результат такой работы должен проявиться в способности студента свободно ответить на теоретические вопросы по теме задания, и правильном его выполнении.

В процессе выполнения лабораторной работы или практического задания студент должен создать требуемый документ с помощью предлагаемого программного средства и выполнить требуемые в задании операции. Задание по лабораторной или практической работе содержит методические указания по подготовке документа, который должен быть получен в результате выполнения работы. При подготовке следует их внимательно прочесть.

## Критерии оценки лабораторных (практических) работ

– 100-86 - выполнены все задания практической (лабораторной) работы, студент четко и без ошибок ответил на все контрольные вопросы.

– 85-76 - выполнены все задания практической (лабораторной) работы; студент ответил на все контрольные вопросы с замечаниями.

– 75-61 выполнены все задания практической (лабораторной) работы с замечаниями; студент ответил на все контрольные вопросы с замечаниями.

- 60-50 баллов - студент не выполнил или выполнил неправильно задания практической (лабораторной) работы; студент ответил на контрольные

**Лабораторная работа №1. Основные команды для работы в Bash**

Чтобы начать работу с Unix, нужно получить доступ к терминалу и зарегистрироваться в системе. В случае удаленной работы подключение терминала можно осуществить запуском программы поддерживающей протокол ssh и соединением с тем компьютером, на котором предполагается вести работу. Когда компьютер готов зарегистрировать пользователя, на экране отображается приглашение к вводу его имени:

```
login: _
```

В ответ на это приглашение нужно ввести регистрационное имя, согласованное с администратором или владельцем компьютера (понятно, что на собственном компьютере пользователь сам вправе выбирать имена пользователей). Имя пользователя рекомендуется составлять из строчных латинских букв и цифр. Некоторые имена (например, root, ftp и т. п.) могут быть зарезервированы для системных целей и не могут быть отданы обычному пользователю. Особый частный случай в большинстве систем – имя root, которое принадлежит администратору или владельцу компьютера. Это имя дает практически неограниченные права по управлению системой. Пользователя с правами root часто называют привилегированным. После ввода имени и нажатия клавиши Enter («Return», «Ret», «CR») система выведет на экран запрос на ввод пароля. Например:

```
login: alex
password: _
```

Ввод пароля завершается нажатием клавиши Enter. Вводимый пароль не отображается на экране. Если пользователь с указанным именем существует и его пароль введен правильно, система сделает домашний каталог пользователя текущим и запустит командную оболочку, связанную с данным пользователем. Оболочка обычно выполняет некоторый начальный набор команд, который может вывести приветственное сообщение, указать на наличие или отсутствие новой почты, выполнить начальный набор команд (все эти действия зависят от особенностей настройки конкретной ОС). И, наконец, на экране появится приглашение к вводу команды:

```
$ _
```

Приглашение не обязательно имеет такой вид, как показано выше, и зависит от конкретной командной оболочки и ее конфигурации.

В системах Unix используются различные командные оболочки (command shells), называемые также командными процессорами или интерпретаторами команд. Среди них наиболее известны и распространены:

- sh (Bourne shell) – оболочка Борна (испытана временем, но не слишком удобна в работе);
- csh (C-shell) – оболочка С (несколько более удобна по сравнению с sh, но несовместима с ней по командному языку);
- ksh (Korn shell) – оболочка Корна (включает мощный командный язык, основанный на языке sh, и развитые средства интерактивной работы);
- bash (Bourne-Again Shell) – «снова» оболочка «Борна» (удобна для интерактивной работы, создана на основе sh и во многом с ней совместима).

Тип оболочки, как правило, можно определить по последнему символу приглашения: знак доллара («$») указывает на sh-совместимую оболочку (sh, bash, ksh), а знак амперсанда («&») соответствует оболочке csh. Однако у привилегированного пользователя независимо от используемого командного процессора последним символом приглашения обычно бывает знак решетки («#»).

Основными функциями командных оболочек являются:

- организация диалога с пользователем (ввод команд);
- выполнение внутренних команд;
- запуск внешних программ;
- исполнение командных файлов.

Возможности командных языков в системе Unix являются гораздо более полными, чем в системе MS-DOS, и вполне могут быть названы полноценными языками программирования. Командные языки в разных оболочках различаются, а стандартным принято считать командный язык оболочки bash.

Общий синтаксис команд в Unix-подобных ОС выглядит следующим образом:

```
$ имя_команды [ключи ...] [параметры ...]
```

Первый элемент обозначает конкретную команду, аргументы (ключи и параметры) могут сообщать дополнительную информацию. Ключи обычно начинаются со знака «минус». Например, команда

```
$ ls -l -a /home
```

состоит из:

- имени команды «ls», выводящей список файлов в заданном каталоге;

- ключа (модификатора) «l», указывающего, что нужно вывести подробный листинг;
- ключа «a», указывающего, что нужно выводить все файлы, включая служебные («дот файлы»);
- параметра «/home», задающего путь к каталогу.

В командах ОС Unix, их ключах и параметрах регистр букв (строчные или заглавные) различается. Для большей части команд характерна запись строчными буквами. Ключи во многих случаях могут объединяться в одну группу. Например, команда «ls -la /home» полностью эквивалентна рассмотренной выше.

Далее рассмотрим простейшие команды для работы с файловой системой.

Команда изменения текущего каталога:

```
$ cd [имя каталога]
```

Если команда cd вызвана без аргументов, текущим каталогом станет домашний каталог пользователя. Чтобы вывести на экран полное имя текущего каталога, нужно использовать команду pwd без аргументов. Команда

```
$ ls [имя_каталога]
```

позволяет получить листинг указанного каталога. Если имя каталога не указано, то будет выведен листинг текущего каталога. У команды ls есть несколько полезных ключей

l – вывести полную информацию о каждом файле;

a – вывести листинг всех файлов, включая такие, имена которых начинаются с символа точки.

Команды mkdir и rmdir позволяют, соответственно, создать или удалить указанный каталог

```
$ mkdir [имя каталога]
$ rmdir [имя каталога]
```

Команда просмотра файлов less позволяет просматривать файлы произвольного размера и перемещаться по их содержимому с помощью клавиш управления курсором (для выхода используется клавиша «q»)

```
$ less [имя файла]
```

Команда копирования файлов

```
$ cp [источник] [приемник]
```

Команда перемещения или переименования файлов

```
$ mv [источник] [приемник]
```

Команда удаления файлов

```
$ rm [имя файла]
```

С командами cp и rm может использоваться ключ «r», позволяющий копировать, перемещать или удалять каталоги со всем их содержимым рекурсивно.

Для полной информации о перечисленных командах, их аргументах и вариантах их использования можно обратиться к страницам руководства пользователя (команда man).

С каждой программой, запускаемой из командной строки Unix, связаны три стандартных потока данных:

- стандартный поток ввода (stdin);
- стандартный поток вывода (stdout);
- стандартный поток ошибок (stderr).

Программы, требующие входных данных, обычно читают информацию из стандартного потока ввода. Например, команда wc подсчитывает количество строк, слов и символов во входных данных. Если запустить эту команду без аргументов, то wc будет ожидать входных данных с терминала (чтобы закончить ввод данных, нужно нажать комбинацию клавиш Ctrl-D):

```
$ wc
two words
<Ctrl-D>
```

В данном примере программа wc прочитала введенный пользователем текст из стандартного потока ввода (куда пользователь ввел текст «two words»). По умолчанию этот поток соединен с терминалом (с клавиатурой) пользователя, но допускается его перенаправление. Чтобы связать данные стандартного входного потока с произвольным файлом, можно использовать операцию перенаправления «<», например:

```
$ wc < /etc/passwd
```

В данном случае команда wc уже не требует ввода с клавиатуры, т. к. она уже получила входные данные из файла /etc/passwd. Заметим, что данная команда может иметь практическое применение – первая цифра означает количество строк в файле /etc/passwd, что соответствует количеству пользователей, зарегистрированных в системе. Стандартный поток вывода – это поток, куда программы записывают выходные данные. В предыдущем примере команда wc выводила результат (три числа) именно в этот поток. Так же работают и большинство других неинтерактивных команд (включая echo, pwd и ls, рассмотренные выше). Подобно стандартному потоку ввода выходной поток изначально связан с терминалом и также допускает перенаправление. Для связывания стандартного потока вывода с файлом используется операция «>», например:

```
$ ls > filelist.txt
```

В этом примере команда ls, вместо того, чтобы вывести список файлов на экран, записала его в файл с именем «filelist.txt». При этом, если файл с таким именем не существовал, он будет создан, в противном случае его старое содержимое будет потеряно. Существует и другая возможность перенаправления вывода, когда новые выходные данные будут дописаны в конец существующего файла. Для этого используется операция «>>». В следующем примере текущие дата и время будут дописаны в конец файла с именем «dates.txt»:

```
$ date >> dates.txt
```

Сообщения об ошибках выводятся в стандартный поток ошибок. Например, пусть выполняется попытка получить список файлов в каталоге без соответствующих прав доступа:

```
$ ls -l /home/ftp/bin/
ls: /home/ftp/bin/: Access denied
```

В данном случае команда ls вывела сообщение в поток стандартной ошибки. Чтобы перенаправить его в указанный файл, можно использовать операции «2>» и «2>>» (по аналогии с «>» и «>>», только цифра 2 говорит о том, что нужно перенаправить поток ошибок), например:

```
$ ls -l /home/ftp/bin/ 2> last-error.txt
```

Операции перенаправления ввода-вывода можно комбинировать, например:

```
$ wc < /etc/passwd 2>> errors.txt > result.txt
```

Существует другой полезный способ перенаправления вводавывода – конвейеры команд. Операция «|» (знак вертикальной черты) позволяет перенаправить стандартный поток вывода одной команды на стандартный входной поток другой команды:

```
$ ls -l /etc | less
```

В этом примере команда ls выводит длинный список файлов в каталоге /etc, эти данные попадают на вход программы less, которая позволяет пролистывать текст с помощью клавиш управления курсором. Так осуществляется «объединение» двух независимых команд в один «конвейер».

Рассмотрим более сложный пример формирования конвейера команд. Пусть нам требуется получить в файле «bash-users.txt» отсортированный список пользователей в системе, пользующихся командной оболочкой bash. Этого можно было бы добиться использованием нескольких команд, сохраняя

промежуточные данные во временных файлах (комментарии к командам оболочки приведенные после знака #)

```
$ grep 'bash' /etc/passwd > list1.tmp
```

# Поиск по заданному шаблону «bash» в файле /etc/passwd

```
$ sort < list1.tmp > list2.tmp
```

# Сортировка по алфавиту данных из файла list1.tmp и запись в list2.tmp

```
$ cut -f1 -d: < list2.tmp > bash-users.txt
```

#Выделение первых полей строк по разделителю :

# и запись в файл bash-users.txt

```
$ rm list1.tmp list2.tmp
```

# Удаление временных файлов

Конвейеризация команд позволяет обойтись одной составной командой без использования промежуточных файлов

```
$ grep 'bash' /etc/passwd | sort | cut -f1 -d: > bash-users.txt
```

Заметим, что команды типа sort или cut часто называют фильтрами. Фильтры получают данные из стандартного входного потока, преобразовывают их и выводят в стандартный поток вывода.

**Лабораторная работа №2. Установка и настройка веб-сервера**

Apache доступен из дефолтных репозиториев Ubuntu, что позволяет устанавливать его с помощью средств управления пакетами.

Начнём с обновления локального индекса пакетов:

```
$ sudo apt update
```

Далее установим пакет apache2:

```
$ sudo apt install apache2
```

После подтверждения установки apt установит Apache и все необходимые зависимости.

Перед тестированием установки Apache необходимо изменить настройки файрвола для разрешения доступа извне к дефолным веб-портам. Если вы следовали инструкциям по настройке файрвола из руководства по первичной настройке сервера, ваш файрвол UFW уже должен быть настроен таким образом, чтобы ограничивать доступ к вашему серверу.

В процессе установки Apache регистрирует себя в конфигурации UFW, создавая несколько профилей приложения, которые могут быть использованы для включения и отключения доступа к Apache через файрвол.

Выведем профили приложений ufw следующей командой:

```
$ sudo ufw app list
```

Вы увидите список приложений пользователей:

```
Available applications:
  Apache
  Apache Full
  Apache Secure
  OpenSSH
```

Как видно из этого вывода, для Apache доступно три профиля:

Apache: этот профиль открывает порт 80 (обычный, не шифрованный веб-трафик).

Apache Full: этот профиль открывает порты 80 (обычный, не шифрованный веб-трафик) и 443 (трафик шифруется с помощью TLS/SSL).

Apache Secure: этот профиль открывает только порт 443 (трафик шифруется с помощью TLS/SSL).

Рекомендуется включать самый ограниченный профиль, который будет позволять входящий трафик. Поскольку мы не настраивали SSL для нашего сервера в этом руководстве, нам потребуется включить только порт 80:

```
$ sudo ufw allow 'Apache'
```

Вы можете проверить внесённые изменения командой:

```
$ sudo ufw status
```

В выводе вы должны видеть, что HTTP трафик разрешён:

```
Status: active
To                         Action      From
--                         ------      ----
OpenSSH                    ALLOW       Anywhere
Apache                     ALLOW       Anywhere
```

Как видно из этого вывода профиль был включен для разрешения доступа к веб-серверу.

После завершения процесса установки операционная система запустит Apache. Веб-сервер уже должен быть запущен.

Проверим в системе инициализации systemd, что сервис работает, следующей командой:

```
$ sudo systemctl status apache2
● apache2.service - The Apache HTTP Server
   Loaded: loaded (/lib/systemd/system/apache2.service; enabled; vendor preset: enabled)
  Drop-In: /lib/systemd/system/apache2.service.d
           └─apache2-systemd.conf
   Active: active (running) since Tue 2018-04-24 20:14:39 UTC; 9min ago
 Main PID: 2583 (apache2)
    Tasks: 55 (limit: 1153)
   CGroup: /system.slice/apache2.service
           ├─2583 /usr/sbin/apache2 -k start
           ├─2585 /usr/sbin/apache2 -k start
           └─2586 /usr/sbin/apache2 -k start
```

Как видно из представленного вывода, сервис выглядит работающим корректно. Тем не менее, самый надёжный способ проверить работу Apache - это запросить веб-страницу.

Вы можете запросить дефолтную веб-страницу Apache с помощью IP адреса вашего сервера. Если вы не знаете IP адрес вашего сервера, вы можете найти его несколькими способами с помощью командной строки.

Введите следующую команду:

```
$ hostname -I
```

Она вернёт несколько адресов, разделённых пробелами. Вы можете попробовать каждый из них в вашем веб-браузере.

Другой способ заключается в использовании команды, которая позволяет увидеть ваш IP адрес из другого места в сети Интернет:

```
$ curl -4 icanhazip.com
```

После того, как вы найдёте IP адрес вашего сервера, введите его в свой веб-браузер: “http://IP_адрес_вашего_сервера”

Вы должны увидеть дефолтную страницу Apache.

Эта страница свидетельствует о том, что Apache работает корректно. На этой странице также представлена базовая информация о важных файлах и директориях Apache.

Теперь, когда у вас есть работающий веб-сервер, рассмотрим некоторые базовые команды для управления им.

Для остановки веб-сервера наберите:

```
$ sudo systemctl stop apache2
```

Для запуска остановленного сервера наберите:

```
$ sudo systemctl start apache2
```

Для перезапуска сервиса наберите:

```
$ sudo systemctl restart apache2
```

Если вы вносите какие-то изменения в конфигурацию, Apache зачастую может перезагружаться без потери открытых соединений. Для этого наберите команду:

```
$ sudo systemctl reload apache2
```

По умолчанию Apache сконфигурирован на запуск при загрузке сервера. Вы можете отключить такое поведение следующей командой:

```
$ sudo systemctl disable apache2
```

Для повторного включения сервиса при загрузке сервера наберите:

```
$ sudo systemctl enable apache2
```

Теперь Apache должен опять запускаться автоматически при загрузке сервера.

При использовании веб-сервера Apache вы можете использовать виртуальные хосты (аналог серверных блоков в Nginx) для хранения конфигурационных настроек разных сайтов. Это позволяет иметь более одного сайта на одном сервере. В этом руководстве мы будем для примера использовать доменное имя example.com, но вам следует заменить его вашим собственным доменным именем.

Apache уже имеет один виртуальный хост, включенный по умолчанию, который настроен на отдачу документов из директории /var/www/html. Хотя это и удобно для обслуживания одного сайта, это становится неудобным, когда сайтов несколько. Вместо того, чтобы изменять /var/www/html, давайте создадим новую структуру директорий внутри /var/www для нашего сайта example.com, оставив /var/www/html для показа дефолтной страницы пользователям в случаях, когда клиентский запрос не совпадает ни с одним из настроенных доменных имён.

Создайте директорию для example.com используя флаг -p для создания необходимых родительских директорий:

```
$ sudo mkdir -p /var/www/example.com/html
```

Далее настройте владельца директории с помощью переменной окружения $USER:

```
$ sudo chown -R $USER:$USER /var/www/example.com/html
```

Теперь права должны для корневой директории быть настроены правильным образом при условии, что вы не меняли своё значение umask. На всякий случай мы можем удостовериться в этом командой:

```
$ sudo chmod -R 755 /var/www/example.com
```

Далее создадим страницу index.html в nano или любом другом текстовом редакторе:

```
$ nano /var/www/example.com/html/index.html
```

Добавим в файл следующий HTML:

/var/www/example.com/html/index.html

```
<html>
    <head>
        <title>Welcome to Example.com!</title>
    </head>
    <body>
        <h1>Success!  The example.com server block is working!</h1>
    </body>
</html>
```

Сохраните и закройте файл.

Для того, чтобы Apache мог отдавать этот контент, нам необходимо настроить виртуальный хост с корректными настройками. Вместо того, чтобы редактировать существующий файл виртуального хоста /etc/apache2/sites-available/000-default.conf, создадим новый файл для нашего сайта - /etc/apache2/sites-available/example.com.conf:

```
$ sudo nano /etc/apache2/sites-available/example.com.conf
```

Скопируйте следующий текст настроек виртуального хоста в созданный файл:

/etc/apache2/sites-available/example.com.conf

```
<VirtualHost *:80>
    ServerAdmin admin@example.com
    ServerName example.com
    ServerAlias www.example.com
    DocumentRoot /var/www/example.com/html
    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Обратите внимание, что мы обновили DocumentRoot на адрес нашей новой директории, и ServerAdmin на адрес электронной почты, доступный для администратора example.com. Мы также добавили две директивы: ServerName, которая устанавливает базовое доменное имя, которое должно использоваться для хоста, а также ServerAlias, которая определяет другие имена, которые должны использоваться для отображения хоста так же, как и базовое доменное имя.

Сохраните и закройте файл после внесения изменений.

Теперь активируем профиль сайта с помощью утилиты a2ensite:

```
$ sudo a2ensite example.com.conf
```

Деактивируем дефолтный сайт, определённый в 000-default.conf:

```
$ sudo a2dissite 000-default.conf
```

Далее проверим наши настройки на наличие ошибок:

```
$ sudo apache2ctl configtest
```

Вы должны увидеть следующий вывод:

```
Syntax OK
```

Перезапустите Apache для применения внесённых изменений:

```
$ sudo systemctl restart apache2
```

Теперь Apache должен работать с вашим доменным именем. Вы можете проверить это введя http://example.com в вашем браузере, где в результате вы должны увидеть что-то в этом роде: Успешная работа Apache.

После успешной настройки Apache приступим в установке и настройке веб-сервера Nginx, который может являться, в будущем, как промежуточным звеном (Reverse Proxy), так и полноценным веб-сервером.

Для начала приостановим Apache2:

```
$ sudo systemctl stop apache2
```

Nginx доступен в стандартных репозиториях Ubuntu, поэтому мы можем использовать менеджер пакетов apt для его установки.

Установим nginx:

```
$ sudo apt install nginx
```

В результате выполнения этих команд apt установит Nginx и другие необходимые для его работы пакеты на ваш сервер.

Перед тем, как начать проверять работу Nginx, нам необходимо настроить наш файрвол для разрешения доступа к сервису. При установки Nginx регистрируется в сервисе файрвола ufw. Поэтому настройка доступа осуществляется достаточно просто.

Для вывода настроек доступа для приложений, зарегистрированных в ufw, введём команду:

```
$ sudo ufw app list
```

В результате выполнения этой команды будет выведен список профилей приложений:

```
Available applications:
  Nginx Full
  Nginx HTTP
  Nginx HTTPS
  OpenSSH
```

Как видно из этого вывода, для Nginx настроено три профиля:

Nginx Full: этот профиль открывает порты 80 (обычный, не шифрованный веб-трафик) и 443 (трафик шифруется с помощью TLS/SSL).

Nginx HTTP: этот профиль открывает только порт 80 (обычный, не шифрованный веб-трафик).

Nginx HTTPS: этот профиль открывает только порт 443 (трафик шифруется с помощью TLS/SSL).

Рекомендуется настраивать ufw таким образом, чтобы разрешать только тот трафик, который вы хотите разрешить в явном виде. Поскольку мы ещё не настроили SSL для нашего сервера, в этой статье мы разрешим трафик только для порта 80.

Сделать это можно следующей командой:

```
$ sudo ufw allow 'Nginx HTTP'
```

Вы можете проверить изменения введя команду:

```
$ sudo ufw status
```

В результате должен отобразиться вывод следующего вида:

```
Status: active
To                         Action      From
--                         ------      ----
OpenSSH                    ALLOW       Anywhere
Nginx HTTP                 ALLOW       Anywhere
```

После завершения процесса установки операционная система запустит Nginx автоматически. Таким образом веб-сервер уже должен быть запущен.

Мы можем убедиться в этом выполнив следующую команду:

```
$ systemctl status nginx
● nginx.service - A high performance web server and a reverse proxy server
   Loaded: loaded (/lib/systemd/system/nginx.service; enabled; vendor preset: enabled)
   Active: active (running) since Fri 2018-04-20 16:08:19 UTC; 3 days ago
     Docs: man:nginx(8)
 Main PID: 2369 (nginx)
    Tasks: 2 (limit: 1153)
   CGroup: /system.slice/nginx.service
           ├─2369 nginx: master process /usr/sbin/nginx -g daemon on; master_process on;
           └─2380 nginx: worker process
```

Как видно из вывода выше, сервис запущен и работает. Тем не менее, убедимся в его полной работоспособности путём запроса веб-страницы.

Для этого мы можем проверить, отображается ли веб-страница Nginx, доступная по умолчанию при вводе доменного имени или IP адреса сервера. Если вы не знаете публичного IP адреса сервера, вы можете найти этот IP адрес несколькими способами.

Попробуйте набрать эту команду в терминале вашего сервера:

```
$ ip addr show eth0 | grep inet | awk '{ print $2; }' | sed 's/\/.*$//'
```

В результате будет выведено несколько IP адресов. Попробуйте вставить каждый из них в браузер.

Другим способом определить свой IP адрес будет проверка, как ваш сервер виден из Интернета:

```
$ curl -4 icanhazip.com
```

Наберите полученный IP адрес или доменное имя в вашем веб-браузере.

http://IP_адрес_вашего_сервера

Вы должны увидеть страницу Nginx по умолчанию. Если вы видите подобную страницу в своём браузере, вы успешно установили Nginx.

Теперь, когда Nginx установлен и мы убедились в его работоспособности, ознакомимся с некоторыми базовыми командам для управления нашим веб-сервером.

Для остановки веб-сервера используйте команду:

```
$ sudo systemctl stop nginx
```

Для запуска остановленного веб-сервера наберите:

```
$ sudo systemctl start nginx
```

Для перезапуска веб-сервера можно использовать следующую команду:

```
$ sudo systemctl restart nginx
```

Если вы вносите изменения в конфигурацию Nginx, часто можно перезапустить его без закрытия соединений. Для этого можно использовать следующую команду:

```
$ sudo systemctl reload nginx
```

По умолчанию Nginx настроен на автоматический старт при запуске сервера. Если такое поведение веб-сервера вам не нужно, вы можете отключить его следующей командой:

```
$ sudo systemctl disable nginx
```

Для повторного включения запуска Nginx при старте сервера введите:

```
$ sudo systemctl enable nginx
```

При работе с Nginx серверный блоки (аналог виртуальных хостов в Apache) используются для инкапсуляции настроек сайтов и позволяют хостить более одного домена на сервере. Мы рассмотрим настройку серверных блоков на примере example.com, но вам будет необходимо заменить этот домен своим реальным доменным именем.

Nginx уже настроен для поддержки одного серверного блока, который настроен на показ документов из директории /var/www/html. Несмотря на то, что это работает для одного сайта, это не очень удобно для хостинга нескольких сайтов. Вместо того, чтобы менять /var/www/html создадим новую структуру директорий внутри /var/www/ для нашего сайта example.com. Директорию /var/www/html оставим без изменений, её содержимое будет отображаться, если клиентские запросы не подходят для отображения других настроенных на сервере сайтов.

Создадим директорию для example.com следующей командой, используя флаг -p для создания любых необходимых родительских директорий:

```
$ sudo mkdir -p /var/www/example.com/html
```

Далее настроим права доступа для созданной директории для текущего пользователя, используя переменную окружения $USER:

```
$ sudo chown -R $USER:$USER /var/www/example.com/html
```

Теперь права должны для корневой директории быть настроены правильным образом при условии, что вы не меняли своё значение umask. На всяких случай мы можем удостовериться в этом командой:

```
$ sudo chmod -R 755 /var/www/example.com
```

Далее создадим страницу index.html в nano или любом другом текстовом редакторе:

```
$ nano /var/www/example.com/html/index.html
```

Добавим в файл следующий HTML:

/var/www/example.com/html/index.html

```
<html>
    <head>
        <title>Welcome to Example.com!</title>
    </head>
    <body>
        <h1>Success!  The example.com server block is working!</h1>
    </body>
</html>
```

Сохраните и закройте файл.

Для того, чтобы Nginx мог отдавать этот контент, нам необходимо настроить серверный блок. Вместо того, чтобы редактировать существующий файл конфигурации серверного блока, создадим новый файл для нашего сайта - /etc/nginx/sites-available/example.com:

```
$ sudo nano /etc/nginx/sites-available/example.com
```

Скопируйте следующий текст настроек серверного блока в созданный файл:

/etc/nginx/sites-available/example.com

```
server {
        listen 80;
        listen [::]:80;

        root /var/www/example.com/html;
        index index.html index.htm index.nginx-debian.html;

        server_name example.com www.example.com;

        location / {
                try_files $uri $uri/ =404;
        }
}
```

Обратите внимание на то, что мы изменили конфигурацию root на адрес нашей новой директории, а server_name на наше доменное имя.

Теперь активируем файл путём создания ссылки на него в директории sites-enabled, которую Nginx проверяет при старте:

```
$ sudo ln -s /etc/nginx/sites-available/example.com /etc/nginx/sites-enabled/
```

Теперь два серверных блока активированы и настроены для ответа на основании своих директив listen и server_name.

example.com: Будет отвечать на запросы example.com и www.example.com.

default: Будет отвечать на любые запросы на порту 80, которые не соответствуют другим настроенным блокам.

Для того, чтобы избежать потенциальной проблемы hash bucket memory, которая может появиться при добавлении дополнительных имён серверов, нам необходимо изменить одно значение в файле /etc/nginx/nginx.conf. Откройте файл командой:

```
$ sudo nano /etc/nginx/nginx.conf
```

Найдите директиву server_names_hash_bucket_size и удалите символ # для того, чтобы раскомментировать её:

/etc/nginx/nginx.conf

```
...
http {
    ...
    server_names_hash_bucket_size 64;
    ...
}
...
```

Далее проверим файлы Nginx на наличие синтаксических ошибок:

```
$ sudo nginx -t
```

Сохраните и закройте файл.

Если никаких проблем не обнаружилось, перезапустите Nginx для применения внесённых изменений:

```
$ sudo systemctl restart nginx
```

Теперь Nginx должен корректно обрабатывать ваше новое доменное имя. Вы можете убедиться в этом набрав в браузере http://example.com и увидев что-то вроде такого вывода: Успешная настройка example.com

Теперь, когда мы знаем основные команды для управления веб-сервером, ознакомимся с основными директориями и файлами.

Теперь, когда у вас есть установленный и настроенный веб-сервер, вы можете выбирать, какой контент отдавать пользователям, и какие другие технологии вы можете использовать в дополнение к веб-серверу.

**Лабораторная работа №3. Установка настройка PHP-Fpm в связке с Nginx.**

Классической связкой для работы сайтов написанных на php считается apache + mod_php. Так как mod_php по умолчанию требует prefork_mpm, то apache для обработки каждого отдельного соединения создает отдельный процесс, что сильно не экономно с точки зрения расходования оперативной памяти.

Потом появился **nginx** - легковесный проксирующий веб-сервер и его начали ставить перед apache чтобы он занимался выдачей статики и не беспокоил apache по мелочам. Следующим логичным шагом является выключение из цепочки nginx - apache - php посредника в лице apache, чем мы и займемся.

**php-fpm** позволяет демонизировать php дабы избежать затрат на запуск процессов (чем страдает CGI) что умеет и FastCGI. Но php-fpm дает также другие полезные возможности:

- запуск пулов обработчиков от имени заданных пользователей,
- динамическое управление количеством запущенных обработчиков,
- возможность перезапуска обработчиков в случае, если с ними что-то не так,
- ведение лога медленных запросов по аналогии с mysql,
- страницы статуса и пинга,
- прочее.

Установка php-fpm в Debian крайне проста:

```
$ apt install php7.2-fpm
```

Теперь вам необходимо настроить виртуальный хост в nginx так чтобы запросы к статике обрабатывал сам nginx, а запросы к php-скриптам передавал на обработку php-fpm. Пример:

```
server {
    listen 80;
    server_name example.com;
    root /var/www/example.com/html;
    index index.php;
    location ~ \.php$ {
        try_files $uri =404;
        fastcgi_pass unix:/var/run/php7.2-fpm.sock;
        fastcgi_index index.php;
        fastcgi_param SCRIPT_FILENAME
$document_root$fastcgi_script_name;
        include fastcgi_params;
    }
}
```

Здесь указано куда перенаправлять запросы. В нашем случае используется unix-сокет. Для поиска необходимого сокета можно воспользоваться утилитами ps, lsof, ss, netstat. Для чтения документации по утилите:

```
$ man <название утилиты>
```

Также указываем индексный файл и путь до файла. Как видно, все довольно просто. Данный пример можно использовать как формулу.

Создадим простейший php-скрипт

/var/www/example.com/html/index.php

```
<?php echo “hello world!”;
```

При переходе на http://IP_адрес_вашего_сервера увидим приветствие.

**Лабораторная работа №4. Технология программирования PHP.**

Запустите web-сервер и в директории /var/www/example.com/html/ создайте файл index.php.

Откройте данный файл с помощью файлового менеджера WinSCP и занесите следующее содержимое:

```
<html>
  <head>
    <title>Пример</title>
  </head>
  <body>
    <?php echo"Привет, я PHP-программа";?>
  </body>
</html>
```

и сохраните данный файл.

Запустите web-браузер и наберите в адресной строке http://IP_адрес_вашего_сервера и нажмите Enter. Ожидается, что вы увидите результат выполнения данного скрипта.

В окне браузера нажмите правой кнопкой мыши и выберите

"Просмотр HTML-кода". Вы увидите следующий текст:

```
<html>
  <head>
    <title> Пример </title>
  </head>
  <body>
    Привет, я PHP-программа
  </body>
</html>
```

Имена переменных в PHP обозначаются знаком $. То же самое "Привет, я PHP-программа!" можно получить следующим образом:

```
<?php
$message=" Привет, я PHP-программа!";
echo $message;
?>
```

Для вывода информации на экран применяется также функция print.

Измените содержание файла index.php:

```
<html>
<body>
<?php
  print "Здесь используется функция print.";
  print "<p>";
  echo "А здесь использована функция echo.",
  "",
  "P.S. Можно добавить вторую текстовую строку.",
  "",
  "Текстовые строки разделяются запятыми.";
  print "<p>";
  printf("Здесь используется функция printf.");
  print "<p>";
  printf("Функция printf в основном используется для
форматированного вывода чисел и строк.");
  print "<p>";
  printf("Не забывайте о скобках, когда пользуетесь функцией
printf.");
?>
</body>
</html>
```

сохраните документ и нажмите кнопку обновить.

Функция print – самый простейший способ отправки текста в браузер.

Функция echo работает так же, как и print, однако позволяет добавлять к первой текстовой строке, другие строки, разделяя их запятыми.

Функция printf отображает числа в определенном формате, например, выводит дробное число с определенным количеством нулей после запятой, поэтому в функции printf использование скобок обязательно.

При работе со скобками пользуйтесь следующими тремя правилами:

- echo никогда не используется со скобками;
- printf всегда используется со скобками;
- print используется и так и так.

Для реализации циклов в PHP используются операторы while, do…while, for и foreach.

Наберите каждый из следующих примеров и разберитесь как организуются различные виды циклов.

Пример цикла while:

```
<html>
<body>
<?php
$i=0;
while($i<=8)
{ echo "Переменная цикла i=".$i."<br>";
echo "i<sup>2</sup>=".$i*$i."<br>";
$i++; }
?>
</body>
</html>
```

Пример цикла do:

```
<html>
<body>
<?php
$i=0;
do
{ echo "Переменная цикла i=".$i."<br>";
echo "i<sup>2</sup>=".$i*$i."<br>";
$i++; } while($i<=8)
?>
</body>
</html>
```

Пример цикла for:

```
<html>
<body>
<?php
$i=0;
for($i=0;$i<=8;$i++)
{ echo "Переменная цикла i=".$i."<br>";
echo "i<sup>2</sup>=".$i*$i."<br>"; }
?>
</body>
</html>
```

Любой php-скрипт можно также выполнить без использования браузера. Для этого в терминале введите:

```
$ php <путь до php-скрипта>
```

Работа с файлами разделяется на 3 этапа:

- Открытие файла.
- Манипуляции с данными.

- Закрытие файла.

Для того чтобы открыть файл в среде PHP используется функция fopen(). Обязательными параметрами этой функции является имя файла и режим файла.

```
$fp = fopen('counter.txt', 'r');
```

Согласно документации PHP выделяют следующие виды режимов файлов:

r – открытие файла только для чтения.

r+ - открытие файла одновременно на чтение и запись.

w – создание нового пустого файла. Если на момент вызова уже существует такой файл, то он уничтожается.

w+ - аналогичен r+, только если на момент вызова фай такой существует, его содержимое удаляется.

a – открывает существующий файл в режиме записи, при этом указатель сдвигается на последний байт файла (на конец файла).

a+ - открывает файл в режиме чтения и записи при этом указатель сдвигается на последний байт файла (на конец файла). Содержимое файла не удаляется.

Примечание: в конце любой из строк может существовать еще один необязательный параметр: b или t. Если указан b, то файл открывается в режиме бинарного чтения/записи. Если же t, то для файла устанавливается режим трансляции перевода строки, т.е. он воспринимается как текстовый.

Для демонстрации рассмотрим следующий сценарий:

```
<?php
//Открывает файл в разных режимах
$fp = fopen('counter.txt', 'r'); // Бинарный режим
$fp = fopen('counter.txt', 'rt'); // Текстовый режим
$fp = fopen("http://www.dvfu.ru", "r");// Открывает HTTP соединение
на чтение
$fp = fopen("ftp://user:password@example.ru", 'w'); //Открываем FTP
соединение с указанием логина и пароля
?>
```

Записывать данные в файл при помощи PHP можно при помощи функции fwrite(). Это функция принимает 2 обязательных параметра и 1 необязательный. В качестве обязательных параметров выступает дескриптор файла и режим файла:

```php
<?php
$fp = fopen("counter.txt", "a"); // Открываем файл в режиме записи
$mytext = "Это строку необходимо нам записать\r\n"; // Исходная
строка
$test = fwrite($fp, $mytext); // Запись в файл
if ($test) echo 'Данные в файл успешно занесены.';
else echo 'Ошибка при записи в файл.';
fclose($fp); //Закрытие файла
?>
```

Для построчного считывания файла используют функцию fgets(). Функция принимает 2 обязательных параметра:

```php
<?php
$fp = fopen("counter.txt", "r"); // Открываем файл в режиме чтения
if ($fp)
{
while (!feof($fp))
{
$mytext = fgets($fp, 999);
echo $mytext."<br />";
}
}
else echo "Ошибка при открытии файла";
fclose($fp);
?>
```

Примечание: В данном примере значение 999 определяет количество символов, которые будут считываться до тех пор, пока указатель не достигнет конца файла (EOF).

Для того, чтобы считать файл как единое целое, нужно использовать функцию readfile(), принимающая 1 обязательный параметр. Функция открывает файл, отображает его содержимое в окне браузера, а затем закрывает файл:

```php
<?php
echo readfile("counter.txt");
?>
```

Также можно использовать функцию fpassthru() которая принимает 1 обязательный параметр. Перед использованием этой функции необходимо открыть файл в режиме чтения. По окончанию считывания файла функция автоматически закрывает файл(при этом дескриптор файла становиться недействительным).

```php
<?php
$fp = fopen("counter.txt", "r"); // Открываем файл в режиме чтения
if ($fp) echo fpassthru($fp);
else echo "Ошибка при открытии файла";
?>
```

Очень часто встречаются ситуации, когда необходимо содержимое сайта считать в массив. Эту возможность предусматривает использование функции file(). При вызове этой функции, каждая строка файла сохранятся в отдельном элементе указанного массива.

Примечание: Не следует применять функцию file() к двоичным файлам (binary-safe), т.к. она не является безопасной в плане считывания двоичных файлов, если при этом, где-то встретиться символ конца файла (EOF), то она не гарантирует вам чтение всего двоичного файла.

```php
<?php
$file_array = file("counter.txt"); // Считывание файла в массив
$file_array
//
// Работа с данными массива
//
?>
```

Давайте представим ситуацию, когда файл необходимо считать по символам. Для этого мы можем воспользоваться функцией fgetc(). Функция принимает единственный параметр. Функция полезна если нам необходимо найти какой-либо символ или количество одинаковых символов.

```php
<?php
$fp = fopen("counter.txt", "r"); // Открываем файл в режиме чтения
if ($fp)
{
  while(!feof($fp))
  {
    $char = fgetc($fp);
    if ($char == 'c') $i = $i + 1;// Находим символ «c»
  }
  echo 'Количество букв "c" в файле: '. $i;
}
else echo "Ошибка при открытии файла";
?>
```

Закрытие файла происходить с помощью функции fclose(), которая принимает 1 обязательный параметр.

```php
<?php
$fp = fopen("counter.txt", "r");
if ($fp)
{
echo 'Файл открыт';
fclose($fp); // Закрытие файла
}
?>
```

Сборник рецептов

1) Нам необходимо проверить существует ли тот или иной файл. Для этого мы воспользуемся функцией file_exists().

```php
<?php
myfile("counter.txt"); // Используем функцию myfile, передав в качестве аргумента имя файла

function myfile($name) //Создаем функцию для проверки существования файла
{
  if (file_exists($name)) echo 'Файл существует';
  else echo "Файл не существует";
}
?>
```

Примечание: Функция file_exists не производит проверку файлов на удаленном веб-сервере. Для правильной работы функции, файл со скриптом должен находиться на том сервере, где и проверяемый файл.

2) Определяем размер файла с помощью функции filesize()

```php
<?php
myfile("counter.txt");

function myfile($name) //Создаем функцию для проверки существования файла и определения размера файла
{
  if (file_exists($name)) echo "Размер файла: ".filesize($name).' байт';
  else echo "Файл не существует";
}
?>
```

3) Создание временного файла с помощью функции tmpfile()

```php
<?php
$myfile = tmpfile();
fwrite($myfile, "Эта строка записывается во временный файл."); // Записываем во временный файл
fseek($myfile, 0); // Устанавливаем указатель файла
echo fread($myfile, 1024); // выводим содержимое файла
?>
```

4) Вам необходимо определить количество строк в файле. Для этого используем функцию count()

```php
<?php
$fp = file("counter.txt");
echo 'Количество строк в файле: '.count($fp);
?>
```

5) Нам необходимо использовать механизм блокировки файла

```php
<?php
$fp = fopen("counter.txt", 'a');
flock($fp, LOCK_EX); // Блокирование файла для записи
fwrite($fp, "Строка для записи");
flock($fp, LOCK_UN); // Снятие блокировки
fclose($fp);
?>
```

6) Нам необходимо удалить определенную строку из файла

```php
<?php
$num_stroka = 5; //Удалим 5 строку из файла
$file = file("counter.txt"); // Считываем весь файл в массив

for($i = 0; $i < sizeof($file); $i++)
if($i == $num_stroka) unset($file[$i]);

$fp = fopen("counter.txt", "w");
fputs($fp, implode("", $file));
fclose($fp);
?>
```

7) Определение типа файла. Используем функцию filetype(), которая принимает единственный параметр

```php
<?php
$mytype = filetype("counter.txt");
echo "Тип файла: ".$mytype;
?>
```

После вызова строка может содержать одно из следующих значений:

- file – обычный файл
- dir – каталог
- ink – символическая ссылка
- fifo – fifo-канал
- block – блочно - ориентированное устройство
- char – символьно - ориентированное устройство
- unknown – неизвестный тип файла

8) Если вы хотите просмотреть все параметры файла, то следует воспользоваться функцией stat()

```php
<?php
$filename = stat("counter.txt");
echo "<pre>";
print_r($filename);
echo "</pre>";
?>
```

9) Нам необходимо очистить файл, используем функцию ftruncate()

```php
<?php
$fp = fopen("counter.txt", 'a'); //Открываем файл в режиме записи
ftruncate($fp, 0) // очищаем файл
?>
```

10) Нам необходимо узнать дату последнего изменения файла, используем функцию filectime(). Функция возвращает значение времени в форме Unix timestamp.

```php
<?php
echo filectime("counter.txt");
?>
```

## Лабораторная работа №5. Система управления базами данных MySql

MySQL является системой управления базами данных с открытым исходным кодом и обычно используется как часть популярного стека LAMP (Linux, Apache, MySQL, PHP/Python/Perl). MySQL использует реляционную базу данных и SQL (Structured Query Language, язык структурированных запросов) для управления данными.

Короткая версия установки очень проста: достаточно обновить индекс пакетов, установить пакет mysql-server, а затем запустить скрипт настройки безопасности.

```
$ sudo apt update
$ sudo apt install mysql-server
$ sudo mysql_secure_installation
```

Для root пользователя рекомендуется установить пароль. Для этого запускаем клиентскую программу mysql:

```
$ sudo mysql
```

И, оказавшись в консоли MySQL, меняем пароль:

```
> SET PASSWORD FOR 'root'@'localhost' = PASSWORD('123');
```

Не забудьте переподключиться и проверить, что пароль поменялся!

```
> quit
$ mysql –u root -p
```

Создание нового пользователя в MySQL

Ранее мы вносили все изменения в настройки MySQL под root-пользователем, имея полный доступ ко всем базам данных. Однако для случаев, когда могут потребоваться более жесткие ограничения, есть способы создания пользователей с особыми наборами прав доступа.

Давайте начнем с создания нового пользователя из консоли MySQL:

```
> CREATE USER 'newuser'@'localhost' IDENTIFIED BY 'password';
```

К сожалению, на данном этапе пользователь "newuser" не имеет прав делать что-либо с базами данных. На самом деле, даже если если пользователь "newuser" попробует залогиниться (с паролем "password"), он не попадет в консоль MySQL.

Таким образом, первое, что нам необходимо сделать, это предоставить пользователю доступ к информации, которая ему потребуется.

```
> GRANT ALL PRIVILEGES ON * . * TO 'newuser'@'localhost';
```

Звездочки в этой команде задают базу и таблицу, соответственно, к которым у пользователя будет доступ. Конкретно эта команда позволяет пользователю читать, редактировать, выполнять любые действия над всеми базами данных и таблицами.

Поле завершения настройки прав доступа новых пользователей, убедитесь, что вы обновили все права доступа:

```
> FLUSH PRIVILEGES;
```

Теперь ваши изменения вступят в силу.

Теперь мы можем использовать логин и пароль только что созданного пользователя для работы с mysql:

```
$ mysql -u [имя пользователя] –p
```

Для работы с mysql из php требуется установка дополнительного модуля. В качестве примера будем использовать модуль mysqli.

Для того, чтобы его установить воспользуемся командой:

```
$ sudo apt install php-mysql
```

Простой пример показывает, как соединиться с базой данных, выполнить запрос, распечатать результат и отсоединиться.

```php
<?php
// Соединяемся, выбираем базу данных
$mysqli = new mysqli('127.0.0.1', 'your_user', 'your_pass', 'your_database');

// О нет!! переменная connect_errno существует, а это значит, что соединение не было успешным!
if ($mysqli->connect_errno) {
    // Соединение не удалось. Что нужно делать в этом случае?
    // просто попробуем так:
    echo "Извините, возникла проблема на сайте";
    // На реальном сайте этого делать не следует, но в качестве примера мы покажем
    // как распечатывать информацию о подробностях возникшей ошибки MySQL
    echo "Ошибка: Не удалась создать соединение с базой MySQL и вот почему: \n";
    echo "Номер ошибки: " . $mysqli->connect_errno . "\n";
    echo "Ошибка: " . $mysqli->connect_error . "\n";

    // Вы можете захотеть показать что-то еще, но мы просто выйдем
    exit;
}

echo 'Соединение успешно установлено';

// Выполняем SQL-запрос
$sql = 'SELECT * FROM your_table';

if (!$result = $mysqli->query($sql)) {
    // О нет! запрос не удался.
    echo "Извините, возникла проблема в работе сайта.";

    // И снова: не делайте этого на реальном сайте, но в этом примере мы покажем,
    // как получить информацию об ошибке:
    echo "Ошибка: Наш запрос не удался и вот почему: \n";
    echo "Запрос: " . $sql . "\n";
    echo "Номер ошибки: " . $mysqli->errno . "\n";
    echo "Ошибка: " . $mysqli->error . "\n";
    exit;
}
```

```
37: // Уфф, мы это сделали. У нас есть соединение с базой данных и
    успешный запрос.
38: // Но где же его результат?
39: if ($result->num_rows === 0) {
40:     // Упс! в запросе нет ни одной строки! Иногда это ожидаемо и
    нормально, иногда нет.
41:     echo "Мы ничего не смогли найти.";
42:     exit;
43: }
44:
45: // Распечатываем список полученных записей
46: echo "<ul>\n";
47: while ($row = $result->fetch_assoc()) {
48:     echo "<li>" . "\n";
49:     echo $row['column1'] . ' ' . $row['column2'];
50:     echo "</li>\n";
51: }
52: echo "</ul>\n";
53:
54: // Скрипт автоматически закрывает соединение MySQL и освобождает
    память, тем не
55: // менее давайте сделаем это вручную
56: $result->free();
57: $mysqli->close();
58: ?>
```

Аналогичным образом можно формировать запросы на запись, обновление и удаление.

## Лабораторная работа №6. Создание статичной Web-страницы

Для данной практической работы нам потребуется текстовый редактор и браузер.

Замечание: Осваивать HTML надо не в специализированных программных пакетов, а основываясь на ручном создании HTML-кода документа. Научившись вручную писать HTML-код документа, вы без труда освоите любой HTML-редактор.

Теги это начальный или конечный маркеры элемента. Теги определяют границы действия элемента и отделяют элементы друг от друга. В тексте Web - страницы теги заключаются в угловые скобки, а конечный тег всегда снабжается косой чертой.

Теги подразделяются на две группы:

- Теги логического форматирования;
- Теги физического форматирования

**Теги логического форматирования**

Отображают на экране монитора элементы документа, таким образом, как установлено по умолчанию спецификации языка разметки HTML .

Переопределять их параметры или свойства нельзя (кроме использования стилей CSS)

```
<acronym> </acronym>
```

используется для расшифровки аббревиатуры. Реализуется через параметр title и отображается в браузере при наведении курсора на слово-аббревиатуру.

Например:

```
<acronym title="HyperText Markup Language"> HTML</acronym>
```

расшифровка сокращений.

• <cite></cite> Позволяет выделить цитаты и высказывания, названия библиографических источников и пр. Текст помещенный между < cite ></ cite > браузером выделяется курсивом. Аналогичные теги: <EM>,<I>

• <Hx></Hx> x = 1,2,--6 Задаёт определенный размер заголовка, всего текста целиком или его конкретного фрагмента. Этот тег уже подразумевает отступ от текстовой части и делать отступ самостоятельно не надо.

 - Используется для выделения фрагментов текста, которых необходимо акцентировать внимание пользователя. В результате действия тега, фрагмент текста выделяется жирным начертанием.

**Теги физического форматирования**

Теги физического форматирования предназначены для выделения отдельных текстовых фрагментов различными способами, установленными автором документа и позволяют разработчику документа визуально изменять вид текста, варьируя его параметры и значения. К тегам физического форматирования относятся:

• Тег <P></P> - элемент абзаца. Применяется в тексте для обозначения законченности мысли. В Web -странице четкое деление на абзацы обязательное правило, поскольку тексты должны быть лаконичными.

Пример применения:  Здесь располагается текст абзаца  (разрешается писать и заглавные буквы и простые)

Тег  имеет элемент align с атрибутами left, center, right и justify которые выравнивают абзац по левому краю, по центру, либо по правому краю, соответственно.

```
<p align=“left”>Text</p>;
<p align=“center”>Text</p>;
<p align=“right”>Text</p>;
<p align=“justify”>Text</p>;
```

Тег <br>. Элемент, обеспечивающий принудительный переход на новую строку. Имеет только начальный тег. В месте его размещения строка заканчивается, а оставшийся текст печатается с новой строки.

Атрибуты:

left – следующая строка текста отражается на ближайшем свободном пространстве у левого поля;

right - следующая строка текста отражается на ближайшем свободном пространстве у правого поля;

all - следующая строка текста отражается на ближайшем свободном пространстве у любого поля;

• <nobr></nobr> Текст, заключенный между этими тегами, будет выведен в одну строку. Если строка не вмещается в экран, применяют горизонтальную полосу прокрутки.

•  отображает текст жирным начертанием

•  - выделяет нужную часть текста курсивом.

• <u></u> - делает текст подчеркнутым.

• <small></small> - отображает в себе текст шрифтом, меньшим, чем шрифт окружающего текста;

•  - присваивает части текста определенные свойства;

• <sup></sup> - располагает часть текста по нижней линии;

• <sub></sub> располагает часть текста по верхней линии;

• <tt></tt>- элемент, обозначающий текст телетайпа;

• <ins></ins> и <del></del> Эти теги позволяют выделить текст, который надо обозначить как вставленный или удаленный (соответственно). Вставленный текст подчеркивается. Удаленный текст зачеркивается.

• <font></font> Определяет тип, размер и цвет шрифта.

Атрибуты:

face (вид) - позволяет задавать определенный шрифт или несколько шрифтов. Например:

<Font face =“ Arial, Verdana, Tahoma”>;

size – указывает размер шрифта в условных единицах от 1 до 7;

color – задает цвет шрифта (см Приложение 1).

• <bdo></bdo> Позволяет изменять направление текста. Используется с атрибутом dir, которому присваивается либо LTR (слева направо), либо RTL (справа налево).

< BDO dir =“ RTL ”> Направление текста можно изменить</ bdo >

• < blockquote > </ blockquote > Элемент задает отступ слева по тексту.

Очень полезный тег при форматировании текста, так как позволяет выделять отдельные фрагменты текста и сместить их вправо, а также служит для выделения длинных цитат.

Для навигации между различными страницами существует специальный механизм гиперссылок.

Для создания ссылки необходимо сообщить браузеру, что является ссылкой, а также указать адрес документа, на который следует сделать ссылку. Оба действия выполняются с помощью тега A, который имеет единственный параметр href. В качестве значения используется адрес документа (URL).

Адрес ссылки может быть абсолютным и относительным. Абсолютные адреса работают везде и всюду независимо от имени сайта или веб-страницы, где прописана ссылка.

Пример 1. Использование абсолютных ссылок

```
<html>
<body>
<a href=www.dvfu.ru>Перейти на сайт ДВФУ</a>
</body>
</html>
```

1. Относительные ссылки, как следует из их названия, построены относительно текущего документа или адреса. Примеры таких адресов:

/

/demo/

/images/pic.gif

../help/me.html

manual/info.html

2. Первые две ссылки называются неполные и указывают веб-серверу загружать файл index.html (или default.html) находящемуся в корне сайта (пример 1) или папке demo (пример 2). Если файл index.html отсутствует, браузер, как правило, показывает список файлов, находящихся в данном каталоге. Слэш перед адресом говорит о том, что адресация начинается от корня сайта, двоеточие - перейти на уровень выше в списке каталогов сайта.

Пример 2. Использование относительных ссылок

```
<html>
<body>
<a href=images/xxx.jpg>Посмотрите на мою фотографию !</a><br> <a
href=tip.html>Как сделать такое же фото?</a> </body> </html>
```

3. Ссылки внутри страницы

Большие документы читаются лучше, если они имеют оглавление со ссылками на соответствующие разделы. Для создания ссылки следует вначале сделать закладку в соответствующем месте и дать ей имя при помощи параметра name тега A.

Пример 3. Создание внутренней ссылки

```
<html>
<body>
<a name=top></a> Друг уронил утюг в унитаз. И разбил его. Причем так
разбил, что по назначению унитаз и использовать никак нельзя.
Мгновением назад только что вот все было хорошо и вот уже дыра, да
такая, что можно забыть, что есть такой предмет в доме. Махнул рукой
нечаянно, а потом мучайся...
<a href=#top>Наверх</a>
</body>
</html>
```

4. Между тегами <a name=top> и </a> отсутствует текст, так как требуется лишь указать местоположение перехода по ссылке, находящейся внизу страницы. Имя ссылки на закладку начинается символом # , после чего идет название закладки. Название выбирается любое, соответствующее тематике.

5. Можно, также, делать ссылку на закладку, находящуюся в другой вебстранице и даже другом сайте. Для этого в адресе ссылки надлежит указать ее адрес и в конце добавить символ решетки # и имя закладки.

Пример 4. Ссылка на закладку из другой веб-страницы

```
<html>
<body>
<a href=text.html#bottom>Перейти к нижней части текста</a>
</body>
</html>
```

5. Ссылка на новое окно

Если требуется сделать ссылку на документ, который открывается в новом окне браузера, используется параметр target=_blank тега A.

6. Создание нового окна обычно требуется в случаях, когда делается ссылка на другой сайт, в остальном лучше открывать документы в текущем окне, поскольку обилие окон может сбить читателя с толку.

7. Так как ссылки на текущее или новое окно ничем не отличаются друг от друга, на некоторых сайтах рядом со ссылкой ставят специальную иконку, показывающую, что документ открывается в новом окне.

Пример 5. Создание ссылки на новое окно

```
<html>
<body>
<a href=”www.dvfu.ru”>Обычная ссылка на сайт www.dvfu.ru</a><br> <a href=”www.dvfu.ru” target=”_blank”>Ссылка открывает новое окно на сайт www.dvfu.ru</a>
</body>
</html>
```

Обычная ссылка на сайт **www.dvfu.ru** Ссылка открывает новое окно на сайт **www.dvfu.ru**

## Лабораторная работа №7. Каскадные таблицы стилей

Для работы с CSS (Каскадными таблицами стилей) их необходимо подключить к HTML-документу.

Для осуществления этой задачи мы можем воспользоваться одним из 3-х предлагаемых методов:

- внешний файл,
- inline-описание,
- описание в секции заголовка.
- inline-описание

или описания, встроенное в тег:

```
<p style="color:red; text-align:center;">
Этот текст переопределен стилем
</p>
```

При помощи дополнительного атрибута style мы можем определить нужные нам стилевые параметры в любом теге. Это самый легкий способ, и действует он в пределах лишь одного тега. Но представьте, насколько вырастет размер файла, и насколько неудобно будет его исправлять, если мы будем указывать стиль у каждого тега. Этот способ не слишком отличается, к примеру, от прямого описания внешнего вида при помощи тега <font>.

Описание в секции заголовка.

Его действие распространяется на всю страничку. Определение стилей происходит при помощи классов, которые представляют собой списки с определением всех необходимых параметров оформления.

При использовании этого метода описание стилей необходимо разместить в секции заголовка:

```
<head>
<style type="text/css">
.header {
text-align :center;
font-size : 27pt;}
.red {color : red; }
</style>
</head>
```

Теперь эти стили можно применять в любом месте html-кода. Для этого используется следующая конструкция:

```
<p class="header">Этот текст написан стилем header</p>
<p class="red">Этот текст написан красным цветом</p>
```

Как видите, все не так уж сложно. Главное понять основные принципы. Кроме определения новых классов мы также имеем возможность переопределять стандартные теги. Например, тег :

```
<style type="text/css">
p { text-align : center; font-size :12pt;}
</style>
```

Теперь весь текст, заключенный в теги , будет выглядеть так, как определено данным стилем. Это очень удобно и позволяет легко адаптировать уже существующие странички к использованию стилей. Кроме того, это несколько уменьшает объем файла за счет отсутствия лишних атрибутов class.

Вынесение описания стилей во внешний файл.

Диапазон его воздействия простирается на все файлы, в которые включено описание. Это способ наиболее соответствует концепции HTML 4.0. В случае, если нам потребуется изменить внешний вид сайта, то будет достаточно скорректировать лишь один этот файл. Сравните его с предыдущими способами. В одном из них придется менять описание на каждой страничке, а в другом даже более того - около каждого тега, что, разумеется, совершенно не вдохновляет.

Каким же образом производится внедрение внешнего файла? Для начала создается стилевой файл с описанием всех нужных нам классов (mystyle.css):

```
.header { text-align : center; font-size : 27pt;}
.red { color :red; }
p { text-align : center; font-size : 12pt;}
```

А потом ссылка на него внедряется в документ при помощи тега <link>:

```
<head> .... <link rel="stylesheet" type="text/css"
href="css/mystyle.css" title="MyStyleSheet"> .... </head>
```

Это самый удобный способ, и для основной таблицы стилей рекомендуется пользоваться именно им.

Каскадность стилей

Каскадность заключается в том, что стили могут переопределяться. Приведенный выше список способов внедрения стилей соответствует порядку переопределения. Нижерасположенный способ может переопределять вышерасположенный.

Например, мы определили во внешнем стилевом файле, что текст в теге <p> должен быть написан при помощи шрифта высотой 10 пунктов. Но если в заголовке странички мы дополнительно укажем, что тот же текст в теге <p> должен быть написан шрифтом в 12 пунктов, то текст будет выведен именно таким кеглем - т.е. стиль в заголовке странички переопределил стиль во внешнем файле.

Синтаксис CSS

Описание каждого класса делается при помощи конструкции, подобной этой:

```
.small { font-size: 9pt; }
```

Сначала указывается имя класса - оно может быть произвольным, но желательно все-таки давать осмысленное название. Далее, в фигурных скобках перечисляются все необходимые параметры для данного класса. Параметры отделяются друг от друга точкой с запятой. Вот еще один пример, в котором используется более длинное описание:

Заметьте, что имя класса начинается с точки и таким образом определяет универсальный класс, т.е. такой, который может быть применен к любому тегу. И делается это при помощи следующей конструкции:

```
<p class="small">Накладываем стиль на этот текст</p>
```

Существуют универсальные классы и, так называемые, теговые классы:

```
p.small { font-size: 9pt; }
```

Класс, определенный таким образом, сработает только в том теге, для которого он предназначен, а для всех остальных будет проигнорирован.

Мы можем определять параметры не только для одного тега, но и сразу для нескольких. Для этого в определении стиля достаточно перечислить их через запятую:

```
p, td { font-size: 9pt; color:green;}
```

Такой прием называется группировкой, и в данном случае мы определили и для <p>, и для <td> одинаковый размер и цвет текста.

В случае переопределения существующих тегов, в описании стиля можно указывать не все параметры, а лишь те из них, которые мы хотим изменить. Все остальные параметры примут значения по умолчанию, которые для разных тегов различны.

Псевдоклассы

В CSS есть такое понятие как псевдокласс. В отличие от обычного класса, действие псевдокласса распространяется не на весь текст, к которому применен данный стиль, а лишь на его часть и возможно лишь в определенном состоянии. Чтобы было понятнее, давайте разберем эффект, при котором ссылки подчеркиваются лишь при наведении на них курсора. Эффект достаточно распространенный, и его можно наблюдать в том числе и на этом сайте. Вот фрагмент таблицы стилей, который позволяет достигать вышеописанного эффекта:

```
a { text-decoration: none; }
a:hover { text-decoration: underline;}
```

Верхняя строчка - это переопределение стандартного тега <a>, которое запрещает подчеркивать ссылки, а вот нижняя - это определение стиля для

псевдокласса hover, который описывает стиль ссылки в момент, когда курсор находится над ней.

А вот и другой пример псевдокласса - определение буквицы вначале абзаца:

```
1: p:first-letter { font-size: 200\%; font-weight: bold; }
```

Заметьте, что и в том, и в другом случае действие стиля распространяется либо на определенное состояние (пользователь собирается щелкнуть по ссылке), либо на фрагмент текста (изменяется только первая буква абзаца). В этом и заключается смысл псевдоклассов.

Примечания

Как и в любом достаточно сложном языке, при создании таблицы стилей можно пользоваться комментариями. Их формат аналогичен классическому C:

```
1: /* Этот текст является комментарием */
```

Для небольших сайтов эта возможность Вам вряд ли пригодится, а вот при создании сложных, многоуровневых таблиц стилей комментарии могут пригодиться. Кстати, здесь будет уместно привести золотое правило - чем понятнее названа переменная (в данном случае имя класса), тем меньше комментариев необходимо.

Основные параметры CSS

Все параметры, используемые для определении стиля, условно можно разделить на несколько больших групп:

управляющие видом шрифта (гарнитура, кегль, цвет, наклон, жирность,..) управляющие форматированием абзаца (выравнивание, интерлиньяж, расстояние между словами, отступ красной строки,..) управляющие свойствами блока (отступы слева-сверху-справа-снизу, местоположение блока на страничке, видимость блока,..) другие, которые не вписываются ни в одну из перечисленных выше групп (цвет ссылок странички, изменение внешнего вида курсора,..) Рассмотрим подробнее параметры, используемые для управления внешним видом текста и форматирования абзацев - как наиболее часто употребляемые.

Основные параметры шрифта

```
font-weight: [bold|normal|number] - жирность шрифта
font-style: [normal|italic|oblique] - наклон шрифта
font-size: number - размер шрифта
font-family: name - гарнитура шрифта
color:number - цвет шрифта
background-color: number - цвет подложки
background: url - текстурная подложка
```

Псевдоклассы Ссылок

```
a:active{} Таблица стилей для активных ссылок (при нажатии)
a:link{} Таблица стилей для собственно ссылок
a:visited{} Таблица стилей для посещённых ссылок
a:hover {} Таблица стилей для ссылок при наведении указателя мыши
```

Основные параметры абзаца (и Элементов типа "Box")

```
text-align: [left|right|center|justify] - выравнивание
text-indent: number - отступ красной строки
line-height: number -интерлиньяж
letter-spacing: number - трекинг
padding-left: number - отступ от текста слева
padding-right: number - отступ от текста справа
padding-top: number - отступ от текста сверху
padding-bottom: number - отступ от текста снизу
margin-left: number - отступ от границы слева
margin-right: number- отступ от границы справа
margin-top: number - отступ от границы сверху
margin-bottom: number - отступ от границы снизу
```

Единицы измерения в CSS

В свойствах, которым требуется указание размеров, можно использовать несколько способов для их задания:

относительный размер в процентах (%)

относительный размер при помощи словесного описания (larger, smaller, xx-small, x-small, small, medium, large, x-large, xx-large)

абсолютный размер в типографских единицах - размер может задаваться в пунктах (pt), пиках (pc), пикселях (px), средней шириной буквы "m" (em), средней шириной буквы "x" (ex)

абсолютный размер в стандартных единицах длины - размер может задаваться в сантиметрах (cm), миллиметрах (mm), дюймах (in) абсолютный в пикселях (px)

Задание цвета в CSS

Цвет для тех свойств, где это нужно, может быть определен одним из трех способов:

при помощи названия цвета: yellow, red, green, grey,..

шестнадцатеричным заданием цвета в формате #RRGGBB: #ff0000, #883490, #ffffff,..

десятичным заданием составляющих цвета в формате rgb(red, green, blue): rgb(255,0,0), rgb(100,23,78),..

Примеры описания таблицы стилей:

```
.epigraph {
    font-size: 12pt;
    font-style: italic;
    text-align: right;
    color: rgb(127,127,0);
}

p.big {
    font-size: 16px;
    font-weight: bold;
    color: #ff0000;
}

.menu {
    font-weight: bold;
    font-size: 9pt;
    font-family: arial, helvetica, sans-serif;
}

a:hover {
    color: #b63a3a;
    text-decoration: none;
}
```

Несколько простых правил

Фон

Текст должен читаться. Дикий, аляповатый фон сильно мешает восприятию содержания. При выборе фона учитывайте также, что распечатывать удается лишь темный текст на светлом фоне.

Картинки

Текст с иллюстрациями смотрится лучше, чем просто текст. Нужно учитывать, что в России пользователи Интернета страдают от медленной связи, графика загружается долго, поэтому лучше, если вы поместите небольшие по размеру и количеству килобайт картинки. Скачивая понравившуюся картинку с чужого сайта, вы нарушаете закон об авторских правах. Хорошо, когда вы сами рисуете иллюстрации к своей страничке.

Шрифты

Слишком много разных шрифтов на странице - дурной тон. Лучше выделять слова размером, курсивом или жирным стилем. Подчеркивания воспринимаются как ссылки, советую этого избегать. Помните, ничто не раздражает читателя так, как мигающий текст.

Стиль

Не важно, в каком стиле будет оформлен ваш сайт, поместите ли вы сердечки, цветочки и вензелечки, или используете в оформлении строгую линейную графику, главное, чтобы все страницы сайта были оформлены единообразно, подчинены одной идее.

## Лабораторная работа №8. Создание динамически формируемой HTML-страницы с использованием языка PHP

Одно из главнейших достоинств PHP - то, как он работает с формами HTML. Здесь основным является то, что каждый элемент формы автоматически становится доступным вашим программам на PHP.

Пример #1 Простейшая форма HTML

```
<form action="action.php" method="post">
 <p>Ваше имя: <input type="text" name="name" /></p>
 <p>Ваш возраст: <input type="text" name="age" /></p>
 <p><input type="submit" /></p>
</form>
```

В этой форме нет ничего особенного. Это обычная форма HTML без каких-либо специальных тегов. Когда пользователь заполнит форму и нажмет кнопку отправки, будет вызвана страница action.php. В этом файле может быть что-то вроде:

Пример #2 Выводим данные формы

```
Здравствуйте, <?php echo htmlspecialchars($_POST['name']); ?>.
Вам <?php echo (int)$_POST['age']; ?> лет.
```

Если не принимать во внимание куски кода с htmlspecialchars() и (int), принцип работы данного кода должен быть прост и понятен. htmlspecialchars() обеспечивает правильную кодировку "особых" HTML-символов так, чтобы вредоносный HTML или Javascript не был вставлен на вашу страницу. Поле age, о котором нам известно, что оно должно быть число, мы можем просто преобразовать в integer, что автоматически избавит нас от нежелательных символов. PHP также может сделать это автоматически с помощью расширения filter. Переменные $_POST['name'] и $_POST['age'] автоматически установлены для вас средствами PHP. Ранее мы использовали суперглобальную переменную $_SERVER, здесь же мы точно так же используем суперглобальную переменную $_POST, которая содержит все POST-данные. Заметим, что метод отправки (method) нашей формы - POST. Если бы мы использовали метод GET, то информация нашей формы была бы в суперглобальной переменной $_GET. Кроме этого, можно использовать переменную $_REQUEST, если источник данных не имеет значения. Эта переменная содержит смесь данных GET, POST, COOKIE.

**Практическое задание 1**

Реализовать калькулятор.

**Практическое задание 2**

Реализовать взаимодействие с базой данных, т.е. запись переданных данных из формы с последующим выводом на странице.

## Лабораторная работа №9. Скрипты в HTML-документах

Исполняет JavaScript-код браузер. В него встроен интерпретатор JavaScript. Следовательно, выполнение программы зависит от того, когда этот интерпретатор получает управление. Опишем несколько способов размещения кода JavaScript на странице:

1. В теговом контейнере <BODY>...</BODY>.

```
<body>
…
<script> команды скрипта</script>
…
</body>
```

2. В теговом контейнере <HEAD>...</HEAD> - если код скрипта представляет собой функцию, которая вызывается в ответ на какое-либо событие.

```
<head>
…
<script type=”text/javascript”> Здесь находятся команды сценария
</script>
…
</head>
```

3. Во внешнем файле. По аналогии с тем, как стили подключаются к станице с помощью элемента link, сценарии подключаются с помощью элемента script, только файл имеет расширение не .css, а .js.

```
<head>
…
<script type=”text/javascript” src=”my.js”> </script>
…
</head>
```

4. Обработчик события указывается прямо в теге, без заключения в теги <script> </script>

```
<input type="button" value="Нажать"
        onClick="window.alert('Нажмите еще раз')">
```

Выполнение операторов сценария

Существует несколько способов определения момента запуска сценария. Вот некоторые из них:
1. При загрузке документа;
2. Сразу после загрузки документа;
3. В ответ на действия пользователя.

**Задание 1. Включение скриптов JavaScript в HTML страницу**

Цель этого примера показать, как вставляется JavaScript в файл HTML-документа и продемонстрировать назначение тегов <script> и </script>.

```
<html>
<body>
<br>Это обычный HTML документ <br>
<script language = "JavaScript">
document.write("ЭТО и есть JavaScript!")
</script>
<br>
Выходим обратно в HTML
</body>
</html>
```

**Задание 2. Обработка событий**

Событие — это очень важное в программировании на JavaScript понятие. События главным образом порождаются пользователем, явля-ются следствиями его действий. Если пользователь нажимает кнопку мыши, то происходит событие, которое называется Click. Если экранный указатель мыши движется по ссылке HTML-документа, происходит со-бытие MouseOver.

Можно заставить появиться новое всплывающее окно, которое появляется при нажа-тии кнопки. Появление нового окна будет следствием наступления собы-тия Click.

```
<html>
<body>
<form>
<input type="button" value="Нажми сюда" onClick="alert('Ой-ой')">
</form>
</body>
</html>
```

**Задание 3. Обработка функций. Открытие окон.**

С помощью JavaScript можно создать новое окно броузера. В новое открывшееся окно можно загрузить уже существующий HTML-документ, но в нем также можно разместить абсолютно новый, создаваемый по ходу работы, документ. Рассмотрим пример того, как можно создать новое окно броузера и загрузить в него существующий документ.

Следующий скрипт открывает новое окно браузера и загружает в него HTML-страничку.

```
<html>
<head>
<script language = "JavaScript">
function openWin()
{ myWin=open("index.php");}
</script>
</head>
<body>
<form>
<input type = "button" value="Открываем новое окно"
onClick="openWin() ">
</form>
</body>
</html>
```

**Задание 4. Обработка простых форм**

1. Набрать следующий текст (Обработка формы с введением пароля, содержащим > 3 символов)

```
<html>
<head>
<title>Обработка сложных форм</title>
<script language=JavaScript>
function process(nForm)
{
var result,v,n;
str=document.FirstForm.FirstName.value+""+
document.FirstForm.LastName.value+"\n"+"место
жительства:"+document.FirstForm.Address.value;
if((document.FirstForm.Password1.value==document.FirstForm.Password2
.value)
&&(document.FirstForm.Password2.value.length>3)) {
alert(str);
result=confirm("Войти в систему?");
} else {
alert("Пароль набран неправильно!");
result=0
}
if (result==1) {
alert("регистрация в системе прошла успешно!");
v=0;n=0;
if(document.forms[0].check1.checked==1)v++
else n++;
if(document.forms[0].check2.checked==1)v++
else n++;
if(document.forms[0].check3.checked==1)v++
else n++;
if(document.forms[0].check4.checked==1)v++
else n++;
if(document.forms[0].check5.checked==1)v++
else n++;
alert("ответов ДА-"+v+" "+"ответов НЕТ-"+n);
}
else alert("регистрация не прошла");
}
</script>
</head>
<body>
<center>
<form name="FirstForm">
Заполните форму
<p>
имя: <input type=text size=20 name="FirstName">
<br>
фамилия: <input type=text size=20 name="LastName">
<p>
адрес : <textarea name="Address">
г. Владивосток
</textarea>
<p> Мои увлечения
```

```
<input type=checkbox checked=0 name="check1">Компьютеры
<input type=checkbox name="check2">Учеба
<input type=checkbox name="check3">Книги
<input type=checkbox name="check4">Спорт
<input type=checkbox name="check5">Музыка
<p>
Укажите пароль(не менее трех символов):
<br><input type=password size=10 name="Password1">
<br><input type=password size=10 name="Password2">
<p>
<input type="submit" onClick="process(1)">
</form>
</body>
</html>
```

**Задание 5. Обработка списков**

1. Наберите следующий текст

```
<html>
<head>
<title>Обработка списка</title>
<script language=JavaScript>
function newprocess(){
str=document.SForm.city2.value
alert(str);
}
</script>
</head>
<body><center>
<p>Выберите город в котором хотите жить
<form name="SForm">
<p><select name="city2"size=4>
<option value="Уфа" selected>Уфа
<option value="Москва">Москва
<option value="Питер">Питер
<option value="Париж">Париж
</select>
<p>
<input type=submit value="Выбор" onClick="newprocess()">
</form>
</body>
</html>
```

**Задание 6. Придумайте свою форму, на основе предыдущих, включающую 4 текстовых поля, флажки, список.**

Форма должна: производить проверку пароля, подсчитывать количество выбранных ответов во флажках и обрабатывать список

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

**«Дальневосточный федеральный университет»**

(ДВФУ)

**ШКОЛА ЕСТЕСТВЕННЫХ НАУК**

**ФОНД ОЦЕНОЧНЫХ СРЕДСТВ**

по дисциплине «Инженерия интернет систем»

**Направление подготовки – 09.04.04 Программная инженерия**

Магистерская программа «Разработка программно-информационных систем»

**Форма подготовки (очная)**

**Владивосток**

**2018**

# Паспорт
# фонда оценочных средств
# по дисциплине «Инженерия интернет систем»

| Код и формулировка компетенции | Этапы формирования компетенции | |
|---|---|---|
| ПК-3 знание методов оптимизации и умением применять их при решении задач профессиональной деятельности | Знает | Современные методы оптимизации процесса разработки программного обеспечения |
| | Умеет | Применять методы оптимизации при решении задач профессиональной деятельности |
| | Владеет | Приёмами анализа и разработки Интернет-приложений для использования их в различных предметных областях |
| ПК-8 способность проектировать распределенные информационные системы, их компоненты и протоколы их взаимодействия | Знает | Основные компоненты и протоколы, используемые в Интернет-технологиях |
| | Умеет | проектировать Интернет-приложения |
| | Владеет | Инструментами и способами использования современных Интернет-технологий при создании распределенных приложений |
| ПК15 способность проектировать программное обеспечение, имеющее встроенные средства адаптации к изменяемым условиям эксплуатации | Знает | Основные компоненты и протоколы, используемые в Интернет-технологиях |
| | Умеет | Программировать интернет приложения |
| | Владеет | Инструментами и способами использования современных Интернет-технологий при создании распределенных приложений |

| № п/п | Контролируемые разделы/темы дисциплины | Коды и этапы формирования компетенций | | Оценочные средства - наименование | |
|---|---|---|---|---|---|
| | | | | текущий контроль | промежуточная аттестация |
| 1 | Лабораторная работа №1. Основные команды для работы в Bash | ПК-3 ПК-8 ПК-15 | Знания | Опрос УО-1 | Вопросы к зачету № 2, 4, 9 |
| | | | Умения | Л/работа ПР-6 | |
| | | | Владения | С/работа ПР-11 | |
| 2 | Лабораторная работа №2. Установка и настройка веб-сервера | ПК-3 ПК-8 | Знания | Опрос УО-1 | Вопросы к зачету № 1, 3, 5, 6, 22 |
| | | | Умения | Л/работа | |

| | | | | | |
|---|---|---|---|---|---|
| | | ПК-15 | | ПР-6 | |
| | | | Владения | С/работа ПР-11 | |
| 3 | Лабораторная работа №3. Установка настройка PHP-Fpm в связке с Nginx. | ПК-3 ПК-8 ПК-15 | Знания | Опрос УО-1 | Вопросы к зачету № 6-12 |
| | | | Умения | Л/работа ПР-6 | |
| | | | Владения | С/работа ПР-11 | |
| 4 | Лабораторная работа №4. Технология программирования PHP | ПК-3 ПК-8 ПК-15 | Знания | Опрос УО-1 | Вопросы к зачету № 11-13 |
| | | | Умения | Л/работа ПР-6 | |
| | | | Владения | С/работа ПР-11 | |
| 5 | Лабораторная работа №5. Система управления базами данных MySql | ПК-3 ПК-8 ПК-15 | Знания | Опрос УО-1 | Вопросы к зачету № 14-17 |
| | | | Умения | Л/работа ПР-6 | |
| | | | Владения | С/работа ПР-11 | |
| 6 | Лабораторная работа №6. Создание статичной Web-страницы | ПК-3 ПК-8 ПК-15 | Знания | Опрос УО-1 | Вопросы к зачету № 18-20 |
| | | | Умения | Л/работа ПР-6 | |
| | | | Владения | С/работа ПР-11 | |
| 7 | Лабораторная работа №7. Каскадные таблицы стилей | ПК-3 ПК-8 ПК-15 | Знания | Опрос УО-1 | Вопросы к зачету № 21 |
| | | | Умения | Л/работа ПР-6 | |
| | | | Владения | С/работа ПР-11 | |
| 8 | Лабораторная работа №8. Создание динамически формируемой HTML-страницы с использованием языка PHP | ПК-3 ПК-8 ПК-15 | Знания | Опрос УО-1 | Вопросы к зачету № 22-24 |
| | | | Умения | Л/работа ПР-6 | |
| | | | Владения | С/работа ПР-11 | |
| 9 | Лабораторная работа №9. Скрипты в HTML-документах | ПК-3 ПК-8 ПК-15 | Знания | Опрос УО-1 | Вопросы к зачету № 25, 26 |
| | | | Умения | Л/работа ПР-6 | |

## Шкала оценивания уровня сформированности компетенций

| **Код и формулировка компетенции** | **Этапы формирования компетенции** | | **критерии** | **показатели** |
|---|---|---|---|---|
| ПК-3 знанием мето- | знает | Современные | знает среды | способность |

| | | | | |
|---|---|---|---|---|
| дов оптимизации и умением применять их при решении задач профессиональной деятельности | (пороговый уровень) | методы оптимизации процесса разработки программного обеспечения | программирования для web-приложений | дать ответы на вопросы |
| | умеет (продвинутый) | Применять методы оптимизации при решении задач профессиональной деятельности | умеет создавать web-приложения в специальных средах | способность создать web-приложений с использованием PHP-Storm |
| | владеет (высокий) | Приёмами анализа и разработки Интернет-приложений для использования их в различных предметных областях | владеет навыками создания web-приложения в специальных средах | наличие созданных приложений |
| ПК-8 способностью проектировать распределенные информационные системы, их компоненты и протоколы их взаимодействия | знает (пороговый уровень) | Основные компоненты и протоколы, используемые в Интернет-технологиях | знает приемы работы с интернет-браузерами, приемы программирования в PHP | способность дать ответы на вопросы |
| | умеет (продвинутый) | проектировать Интернет-приложения | умеет создавать интернет-приложения с помощью Apache2/Nginx, PHP, MySQL, JavaScript, CSS, HTML | способность создать web-приложени с использованием Apache2/Nginx, PHP, MySQL, JavaScript, CSS, HTML |
| | владеет (высокий) | Инструментами и способами использования современных Интернет-технологий при создании распределенных приложений | владеет навыками создания интернет-приложения с помощью PHP, JavaScript | наличие созданных приложений |
| ПК15 способность проектировать программное обеспечение, имеющее встроенные средства | знает (пороговый уровень) | методы создания информационных компонентов интернет приложений | знает приемы работы с интернет-браузерами, приемы про- | способность дать ответы на вопросы |

| | | | | |
|---|---|---|---|---|
| адаптации к изменяемым условиям эксплуатации | | | граммирования в PHP при создании информационных компонентов | |
| | умеет (продвинутый) | проектировать Интернет-приложения для решения задач обработки данных | умеет создавать интернет-приложения с помощью Apache2/Nginx, PHP, MySQL, JavaScript, CSS, HTML | способность создать web-приложения с использованием Apache2/Nginx, PHP, MySQL, JavaScript, CSS, HTML |
| | владеет (высокий) | Инструментами и способами использования современных Интернет-технологий при создании распределенных приложений для решения задач обработки данных | владеет навыками создания интернет-приложения с помощью PHP, JavaScript | наличие созданных приложений |

## Методические рекомендации, определяющие процедуры оценивания результатов освоения дисциплины

### Промежуточный контроль

Промежуточный контроль осуществляется в конце семестра и завершает изучение дисциплины. Помогает оценить более крупные совокупности знаний и умений, сформированность определенных профессиональных компетенций по дисциплине. Промежуточный контроль проводится в форме зачета, допуск к экзамену возможен для обучающихся, получивших оценку «зачтено» в результате выполнения самостоятельной работы и успешно выполнившие все лабораторные работы.

## Критерии выставления оценки магистранту на зачете

| **Баллы** (рейтинговой оценки) | **Оценка экзамена** (стандартная) | **Требования к сформированным компетенциям** |
|---|---|---|
| 86-100<br><br>76-85<br><br>61-75 | «зачтено» | Оценка «зачтено» выставляется магистранту, если он<br>-глубоко и прочно усвоил программный материал, исчерпывающе, последовательно, четко и логически стройно его излагает, умеет тесно увязывать теорию с практикой, свободно справляется с задачами, вопросами и другими видами применения знаний, причем не затрудняется с ответом при видоизменении заданий, правильно обосновывает принятое решение, владеет разносторонними навыками и приемами выполнения практических задач;<br>- твердо знает материал, грамотно и по существу излагает его, не допуская существенных неточностей в ответе на вопрос, правильно применяет теоретические положения при решении практических вопросов и задач, владеет необходимыми навыками и приемами их выполнения;<br>- имеет знания только основного материала, но не усвоил его деталей, допускает неточности, недостаточно правильные формулировки, нарушения логической последовательности в изложении программного материала, испытывает затруднения при выполнении практических работ. |
| 0-60 | «незачтено» | Оценка «незачтено» выставляется магистранту, который не знает значительной части программного материала, допускает существенные ошибки, неуверенно, с большими затруднениями выполняет практические работы. Как правило, оценка «незачтено» ставится магистрантам, которые не могут продолжить обучение без дополнительных занятий по соответствующей дисциплине. |

## Промежуточный контроль

Промежуточный контроль осуществляется в конце семестра и завершает изучение дисциплины. Помогает оценить более крупные совокупности знаний и умений, сформированность определенных профессиональных компетенций по дисциплине. Промежуточный контроль проводится в форме зачета, допуск к экзамену возможен для обучающихся, получивших оценку «зачтено» в результате выполнения самостоятельной работы и успешно выполнившие все лабораторные работы.

**Оценочные средства для промежуточной аттестации**

**Вопросы к зачету по дисциплине**

1. Основы компьютерных сетей.
2. Утилиты для работы с файлами и директориями
3. Модели работы веб-серверов.
4. Установка дополнительных утилит/библиотек с помощью менеджера пакетов apt.
5. Установка и настройка веб-сервера Apache2.
6. Установка и настройка веб-сервера Nginx.
7. Установка настройка PHP-Fpm в связке с Nginx
8. Основные способы межпроцессорного взаимодействия (unix-сокеты, net-сокеты).
9. Утилиты для работы с процессами, сокетами.
10. Сервисы в Linux.
11. Создание простейшего скрипта на языке PHP.
12. Использование PHP для создания простых скриптов.
13. Работа с файлами с помощью PHP.
14. Реляционные базы данных.
15. Установка и настройка MySql-сервера.
16. Работа с MySql с помощью консольного клиента.
17. Работа с MySql базой данных из PHP-скрипта.
18. Язык гипертекстовой разметки HTML.
19. Основные тэги и атрибуты HTML.
20. Работа с гиперссылками.
21. Использование CSS в оформлении web-страниц.
22. Протокол HTTP.
23. Передача параметров между клиентом и сервером.
24. Формы в HTML.
25. Скрипты в HTML-документах
26. Использование Java Script при создании web-сайта.

**Критерии оценки проектов**

- 100-86 баллов выставляется, если магистрант/группа точно определили содержание и составляющие части задания, умеют аргументировано отвечать на вопросы, связанные с заданием. Продемонстрировано знание и владение навыками самостоятельной исследовательской работы по теме. Фактических ошибок, связанных с пониманием проблемы, нет.
- 85-76 - баллов - работа магистранта/группы характеризуется смысловой цельностью, связностью и последовательностью изложения; допущено не более

1 ошибки при объяснении смысла или содержания проблемы. Продемонстрированы исследовательские умения и навыки. Фактических ошибок, связанных с пониманием проблемы, нет.

- 75-61 балл – проведен достаточно самостоятельный анализ основных этапов и смысловых составляющих проблемы; понимание базовых основ и теоретического обоснования выбранной темы. Привлечены основные источники по рассматриваемой теме. Допущено не более 2 ошибок в смысле или содержании проблемы
- 60-50 баллов - если работа представляет собой пересказанный или полностью переписанный исходный текст без каких бы то ни было комментариев, анализа. Не раскрыта структура и теоретическая составляющая темы. Допущено три или более трех ошибок смыслового содержание раскрываемой проблемы

## Шкала оценивания проектов

| | |
|---|---|
| Менее 60 баллов | Не зачтено |
| От 61 до 75 баллов | зачтено |
| От 76 до 85 баллов | зачтено |
| От 86 до 100 балов | зачтено |

## Текущий контроль

Текущий контроль предполагает систематическую проверку усвоения учебного материала, сформированности компетенций или их элементов, регулярно осуществляемую на протяжении изучения дисциплины, в соответствии с ее рабочей программой.

Состоит в проверке правильности выполнения заданий по самостоятельной работе. Задание зачтено, если нет ошибок. По текущим ошибкам даются пояснения.

Тесты предназначены для проверки знаний по компетенциям. Проверка достижения умений и навыков по компетенциям проверяется выполнением практических работ.

## Примерные тесты для проверки сформированности компетенций

| **ПК-3 знанием методов оптимизации и умением применять их при решении задач профессиональной деятельности** | Современные методы оптимизации процесса разработки программного обеспечения |
|---|---|
| 1. Что из перечисленного является средой разработки с поддержкой языка PHP? | 1. Windows<br>**2. PHP-Storm** |

| | |
|---|---|
| | 3. SSH<br>**4.** PHP-Fpm |
| 2. Какой протокол позволяет производить удалённое управление операционной системой? | 1. PHP<br>2. HTML<br>3. FTP<br>**4. SSH**<br>**5.** Нет правильного варианта |
| 3. Для просмотра Web-страниц в Интернете используются программы: | 1. MicroSoft Word или Word Pad<br>2. MicroSoft Access или MicroSoft Works<br>**3. Internet Explorer или NetScape Navigator**<br>4. HTMLPad или Front Page |

| **ПК-8 способностью проектировать распределенные информационные системы, их компоненты и протоколы их взаимодействия** | Основные компоненты и протоколы, используемые в Интернет-технологиях |
|---|---|
| 1. С помощью какого символа в PHP можно склеить 2 строки в одну? | **1. Точка**<br>2. Плюс<br>3. Звёздочка<br>4. Процент<br>**5.** Данная операция невозможна |
| 2. Каким является язык PHP? | **1. интерпретируемым**<br>2. компилируемым<br>3. прототипируемым |
| 3. Что из перечисленного не является моделью работы веб-серверов? | 1. Event Model<br>**2. Process Model**<br>3. Reach Model<br>4. Worker Model<br>5. **Trigger Model** |
| 4. Наличие каких компонент обязательно для создания TCP-сокета? | **1. IP-адрес**<br>**2. Порт**<br>3. Файл<br>4. Браузер |

| **ПК15 способность проектировать программное обеспечение, имеющее встроенные средства адаптации к изменяемым условиям эксплуатации** | знает приемы работы с интернет-браузерами, приемы программирования в PHP при со-здании информационных компонентов |
|---|---|
| 1. В чём Вы видите назначение CSS? | **1. В создании интерактивных сайтов**<br>**2. В разделении содержания и представления веб-страницы**<br>3. В структуризации контента.<br>4. В создании большой таблицы<br>**5.** Нет правильного варианта. |
| 2. Для чего используется атрибут target тега <a>?(HTML) | 1. задает адрес документа, по которому следует перейти.<br>2. устанавливает имя якоря внутри |

| | |
|---|---|
| | документа.<br>**3. задает имя окна или фрейма, куда браузер будет загружать документ**<br>4. добавляет всплывающую подсказку к тексту ссылки.<br>5. атрибут target недопустим для тега <a>.<br>**6.** Нет правильного варианта. |
| 3. Какие инструменты позволяют редактировать файлы на удалённом сервере? | **1. WinSCP**<br>2. Ajax<br>3. Microsoft Word<br>4. Internet Explorer |